Adaptive Portfolio Management and Stock Price Forecasting Using Long Short-Term Memory (LSTM) Neural Networks

Jacob May
CS682
jacobmay@umass.edu

Matthew McCarthy
CS682
mtmccarthy@umass.edu

Srikrushna Pinaki Budi
CS682
sbudi@umass.edu

Abstract

The problem of effectively trading stocks is an incredibly popular problem, and has been for a long time. With the introduction of computers, and their subsequent skyrocket in power and utility, this problem has taken on new and interesting dynamics. One of these, is to try and predict security prices using machine learning. In this paper we take a well studied machine learning technique for forecasting time series data, Long Short Term Memory (LSTMs), and attempt to use these price predictions in a portfolio management simulation. With the price predictions we produced using LSTM networks, we were able to achieve impressive portfolio management results during our simulations.

1. Introduction

The goal of our project is not solely to accurately predict the price on any given day, but to take advantage of mispricing within the market. We aim to buy profitable stocks when they are under-valued, and sell speculative stocks that are over-valued. To solve this we want to create a system that automatically distributes capital amongst a set of assets with the intention to maximize returns and minimize risk.

First we need to build a price prediction machine learning model. Long Short Term Memory networks are a common modern technique to approach to this problem. LSTMs are an extension of Recurrent Neural Networks that allow for some state data to pass through the sequence without being feed through the computation heavy piece of a traditional RNN. This creates a large save in training hardware needs, allowing for deeper networks that trains faster. More detail on LSTMs can be found in the background section.

After generating models for each stock that are able to accurately predict future prices, we can use this as input in our trading simulations. Each simulation is a time series in which we control the start date, the end date, the time step, and the future prediction distance. We also implemented a custom trading strategy algorithm that decides when to buy and when to sell. After running this automated portfolio manger we are left with a rate of return and a make-up of assets the model thought to be most optimal. Finally, we can compare these we leading public portfolios to see how our LSTM backed trading strategy compares to industry standards.

2. Background

2.1. Choosing LSTM

We initially faced uncertainty about which deep neural network architecture would best suit our portfolio management needs. To guide our decision, we referred to the comprehensive study by Huang et al. [1] This research assesses six different neural network types in various financial areas, including portfolio management. Our choice of Long Short-Term Memory (LSTM) networks was influenced by several factors highlighted in Huang's review:

2.1.1 Proficiency in Time Series Data

LSTMs are designed to process time series data, crucial for our price prediction models used in our portfolio since stock data is time series.

2.1.2 Long Term Memory Capabilities

Unlike other networks, LSTMs excel in maintaining long-term memory, enabling them to handle extensive time series data more effectively which will allow our models to train on many more days and years of stock price data.

2.1.3 Avoidance of Vanishing Gradients

When processing long sequences of data, LSTMs are better equipped to avoid vanishing gradients which will allow us to train our model more effectively over a long sequence.

2.1.4 Previous Research

While Huang's review also indicated substantial research on Feedforward Neural Networks (over half the papers in the analysis), the specific advantages of LSTMs—particularly their long-term memory and gradient stability—made them the more suitable choice for our needs, while also having the second most number of papers to reference.

2.2. LSTM Architecture

We primarily used as reference three papers in the development of the LSTM Price Classifier model used to power our portfolio trading strategy, Moghar et al. [2], Sen et al.n [4] and Akhter [3].

2.2.1 Moghar et al.

Moghar et al. [2] used a nine layer approach with four LSTM layers, each followed by a dropout layers (totaling four dropout) and finally a single dense layer. Each LSTM layer had 96 hidden nodes, and a timestep of 50. They do not mention their dropout rate. They train for a variety of epochs on two stocks and show best performance with around 25 epochs. We found the same implementation for LSTM evaluated by [5]

2.2.2 Sen et al.

Sen et al. [4] used a different approach with fewer layers (2 LSTM and 2 dropout), but more hidden nodes in each layer (256), but also used a timestep of 50. They used a dropout rate of 30%, a batch size of 64 and trained for 100 epochs.

2.2.3 Akhter el al.

Akhter [3] uses three layer approach, the first two LSTM layers have 100 nodes, and the final layer uses 50 nodes. Akhter mentions that overfitting tends to occur around 100 epochs and implements dropout of 30% similar to Sen et al. [4] and adds to this by implementing early stopping. Ahkter uses a batch size of 100 during training. Akhter uses a ReLU activation function. We go on to take advantage of this finding and limit our training to 50 epochs while also taking advantage of the EarlyStopping feature in Keras.

2.3. Portfolio Management

Ahkter used these price prediction models to implement and test a portfolio trading strategy. Ahkter evaluates the the portfolio using rate of return and a measure of risk with the Sharpe ratio. Ahkter evaluates his model compared to other famous portfolio models Markowitz Mean-variance model and Mean Semivariance model. Akhter states that his portfolio has similar expected returns to the Markowitz models but with less risk due to more diversification. Actual returns from certain test periods on specific stocks are not mentioned.

3. Technical Approach

3.1. General Overview

We used the following Kaggle data-set. This provided us with plenty of data for the scope of our experiments and simulations. The stock data contains a separate text file for each stock and it contains the High, Low and Close Prices of that stock for each day until October 2017.

We have nine security assets and cash. The portfolio algorithm is designed in a two-step process. Initially, we create a deep neural network model to predict stock prices for each asset using a mixture of LSTM, dropout, and a single dense layer. After this, we feed our predictions to our trading algorithm and finally evaluate the performance of the portfolio compared to other leading portfolio managers.

3.2. Building a LSTM Model

3.2.1 Data Loading and Processing

Using the Kaggle data-set mentioned above, we were able to load the stock data in a Pandas Data-frame. (figure 1) Using Pandas and Numpy, we split the data in training, validation, and testing sets. (figure 2)

	Date	Open	High	Low	Close	Volume	OpenInt
0	2005-02-25	7.1722	7.3563	7.1457	7.2511	5375043	0
1	2005-02-28	7.2949	7.3388	7.1194	7.1808	4602417	0
2	2005-03-01	7.1808	7.2597	7.1194	7.1808	5648916	0
3	2005-03-02	7.1457	7.5493	7.0931	7.4439	10262412	0
4	2005-03-03	7.4879	7.6632	7.2949	7.3299	5287947	0

Figure 1. Pandas Data-frame of AMTD Data

Figure 2. AMDT Data split into Train, Test, and Validation

3.2.2 Issues with Data Across Stocks

The primary issue we ran into was the scale and integrity of our data. Not all stocks had the same data across time. Some stocks had data as old as 1960 while others were new stocks and had only a few years of data. We addressed this by deliberately selecting a portfolio of stocks where the date range was consistent. We also had to address issues of gaps in data. To address this, we chose to ignore dates in the

test data where any given stock in the portfolio did not have actual pricing during the test period.

We tried splitting the data synchronously by train, then validation, and finally test but this was led to worse performance than train, test, validation split (we use 80, 10, 10 percent splits). This might be because when the validation data is far away from train data in the Time Series, it reduces the over-fitting of the model. When the validation data is near the train data in the Time Series, it might be following similar pattern as training data, and not able to detect over-fitting.

3.2.3 Selected Model

We used the Keras framework to implement our machine learning networks in Python. Using the sequential builder we constructed a model with four LSTM layers and one affine layer. (figure 3/4) We are using the Adam optimizer as well as Early-Stopping to halt training if there is no improvement in validation loss for ten consecutive epochs.

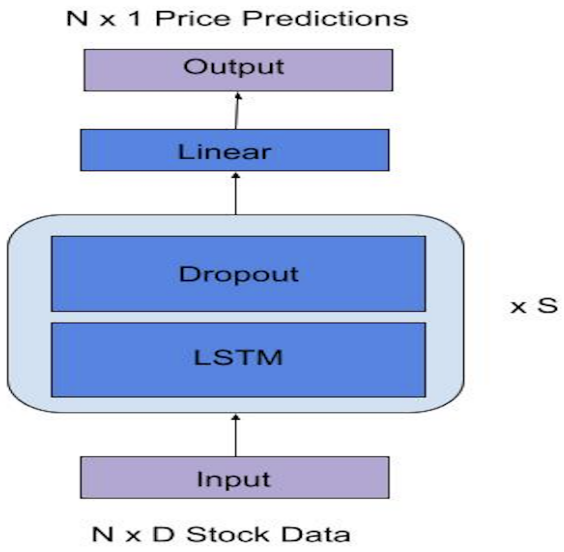

Figure 3. Model Architecture

```
def train(self, model=None, optimizer=None, epochs=50, batch_size=32, loss="mean_squared_error"):
  if not model:
    model = Sequential()
    model.add(LSTM(units=50,return_sequences=True,input_shape=(self.X_train.shape[1], 1)))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50,return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50,return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))

  if not optimizer:
    optimizer = Adam()
  model.compile(optimizer=optimizer,loss=loss)
  early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
  history = model.fit(self.X_train,self.y_train,epochs=epochs,
                      batch_size=batch_size,
                      validation_data=(self.X_val, self.y_val),
                      callbacks=[early_stopping])
  self.model = model
```

Figure 4. Training Function with LSTM Model and Optimizer

Using this model we were able to get solid prediction results for the AMTD stock after only 50 epochs. (figure 4, 5)

```
Epoch 50/50
80/80 [==============================] - 1s 14ms/step - loss: 1.9656 - val_loss: 66.0949
```

Figure 5. Final MSE for Training and Validation Sets for AMTD

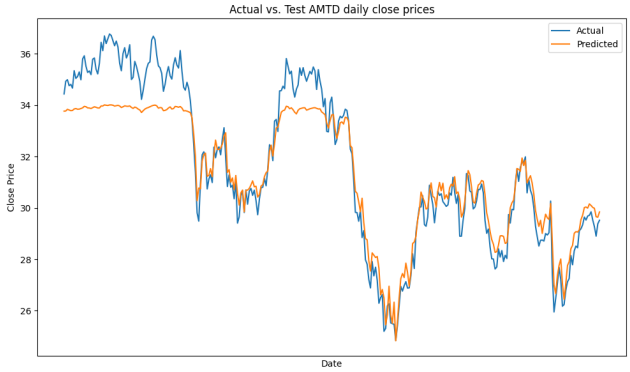

Figure 6. Actual Close Prices vs Predicted Close Prices for AMTD

We chose to use default hyperparameters for activation (tanh), recurrent activation (sigmoid) and no recurrent dropout since there is a performance boost in Keras when using these default values. It enables use of cuDNN fast implementation and allows for faster training. Keras LSTM Documentation

3.2.4 Experimentation with Loss Functions

We experimented with two different loss functions, mean absolute error, and mean squared error. We compared the pricing performance visually and chose to use mean squared error since it seemed to perform a bit better and is standard for regression tasks.

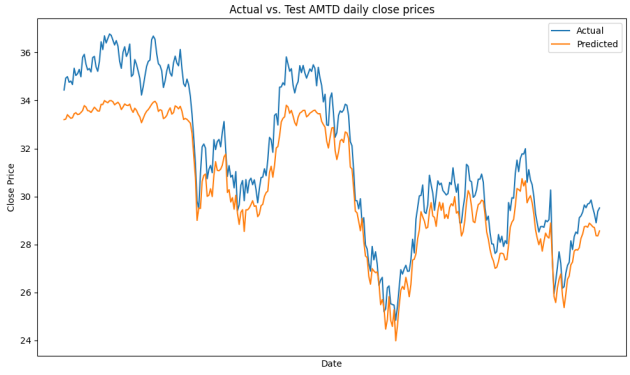

Figure 7. AMTD using Mean Absolute Error Loss

3.2.5 Experimentation with Dropout

In all previous papers referenced a dropout rate of 30% was used. We decided to experiment with various dropout rates from 10-40%. We compared pricing performance visually and with an evaluated loss function using keras.Model.evaluate(). Results were inconclusive between stocks. For some stocks the best loss was clearly 20% dropout and for others 20% dropout performed terribly compared to 10%, 30%, or 40%. Our conclusion is that there is no generic dropout rate that works well for all stocks across time periods and that if increased performance is desired, specific dropout rates should be experimented for each stock.

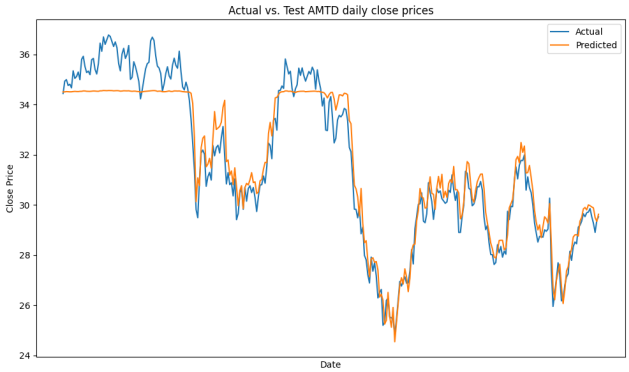

Figure 8. AMTD using 10% dropout

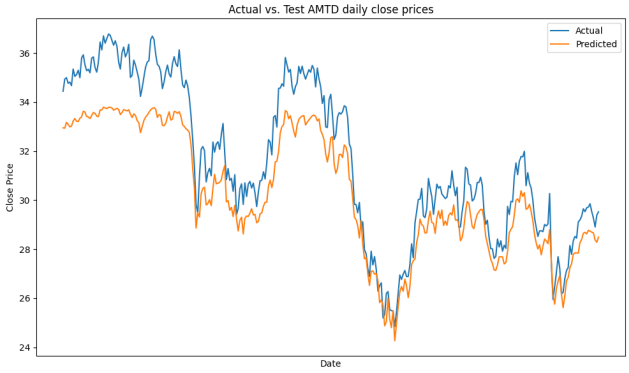

Figure 9. AMTD using 30% dropout

3.2.6 Experimentation with Layer Normalization

We attempted to increase performance by adding in layer normalization layers in between the dropout and LSTM layers. We did not see any increase in performance using layer normalization and did not pursue any attempts to tune those layers.

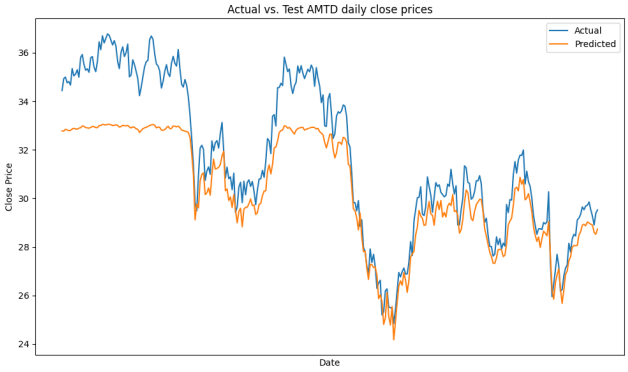

Figure 10. AMTD using 40% dropout

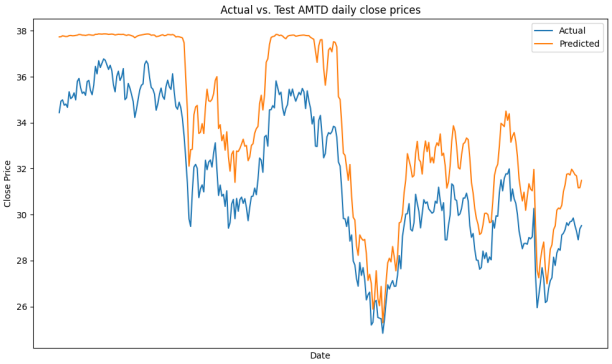

Figure 11. AMTD Layer Normalization Between LSTM and Dropout layers

3.2.7 Experimentation with Number of LSTM Layers

In the papers referenced, the number of LSTM layers varied. We experimented with two, four, and six layers of LSTM mixed with dropout each with a single dense layer at the end. There were no drastic differences between these variations, but the four LSTM layer variation performed the best on the AMTD stock.

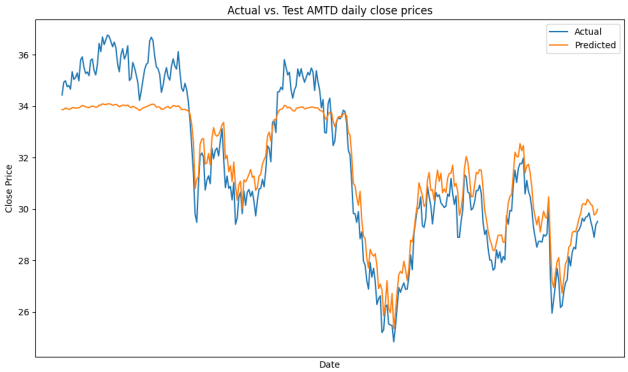

Figure 12. AMTD 2 Layers of LSTM

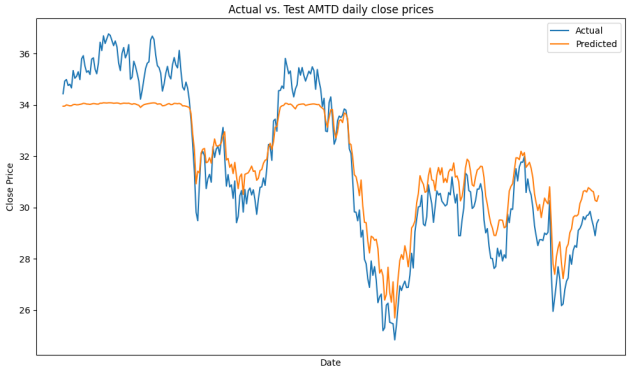

Figure 13. AMTD 6 Layers of LSTM

3.2.8 Experimentation with Bias

We attempted to disable the bias in all the LSTM Layers. We saw mixed results because the AMTD model it was resulting in little less test prediction loss. But for the CERN model it was increased by a large amount. And for some stocks like STAR, it remained mostly same. Therefore we decided on keeping the bias because there was no large decrease in the test losses and it is a best practice to keep the bias term.

3.3. Automated Portfolio Manager

The second half of our project is to use our trained models as a guide to manage a simulated portfolio. To do this we needed to build a time series simulation system that provides a flexible interface of designing experiments, and implements our chosen trading strategy

3.3.1 Simulation Interface

The system allows for robust configuration. We are able to configure start and end dates, time step for each trading attempt, look forward gap, as well as any initial asset state we desire. This allowed us to carefully simulate over only true testing data and not cheat by predicting over our training data for each model. It provides enough flexibility to test both short sighted and long term trading philosophies. In our best portfolio, we adopted a medium-long term outlook trading only every 20 days, and running for a two year period.

3.3.2 Trading Strategy

A simple yet effective strategy is to sell any stocks that are overvalued, and buy any stocks that our undervalued. This makes sense as stock prices should always converge to their true value, and this strategy will sell high and buy low and thus create a good return. The problem is how to determine what the ”true value” of the stock is. To do this, we predict the price using our LSTMs based on a specified jump into the future. The further the look ahead, the more long term the strategy is thinking. We found success looking twenty days into the future.

In order to prevent the portfolio from becoming solely invested into one stock, we introduced a parameter of minimum hold amount. This is the minimum the portfolio must hold of every stock at any time. We will see qualitatively why this is important in the results section when we discuss Sharpe ratios, but the intuition is that forcing diversity on our model reduces risk.

3.3.3 Link to Code

A link to the code for this paper can be found at the following github link.

4. Results

4.1. Stock Prediction

Using the above network architecture we trained and tested ten different stock models. We were able to achieve a range of decent to really well fitting models for all ten stocks. As we will discuss in the trading results, even the decent fitting models with some clear area’s of wrong predictions work well enough to create a successful portfolio All loses are calculated using mean squared error.

4.1.1 AMTD

We were able to train a good model for the AMTD security with training loss of 1.9213 and validation loss of 66.1083.

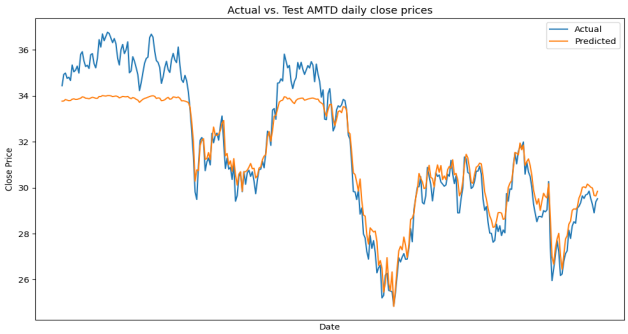

Figure 14. Actual Close Prices vs Predicted Close Prices for AMTD

4.1.2 CMS

We were able to train a good model for the CMS security with training loss of 0.9719 and validation loss of 97.1346.

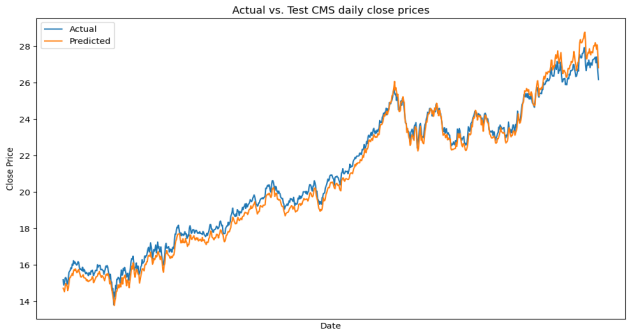

Figure 15. Actual Close Prices vs Predicted Close Prices for CMS

4.1.3 COT

We were able to train a good model for the COT security with training loss of 0.6306 and validation loss of 0.1755.

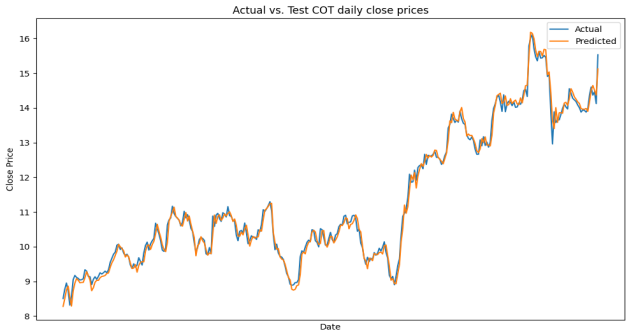

Figure 16. Actual Close Prices vs Predicted Close Prices for COT

4.1.4 CERN

We were able to train a decent model for the CERN security with training loss of 7.5461 and validation loss of 15.6564.

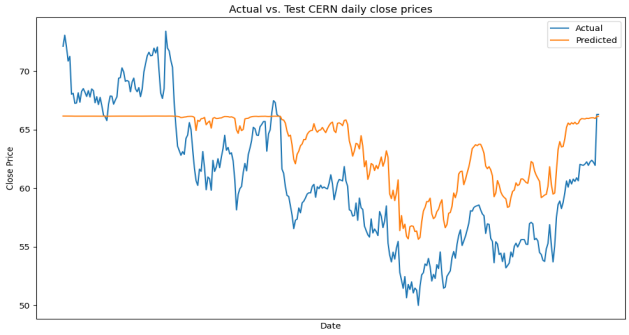

Figure 17. Actual Close Prices vs Predicted Close Prices for CERN

4.1.5 STAR

We were able to train a good model for the STAR security with training loss of 4.1497 and validation loss of 0.4697.

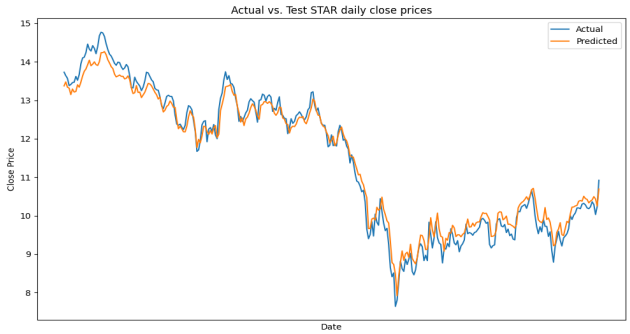

Figure 18. Actual Close Prices vs Predicted Close Prices for STAR

4.1.6 SONS

We were able to train a good model for the SONS security with training loss of 2.7020 and validation loss of 0.0905.

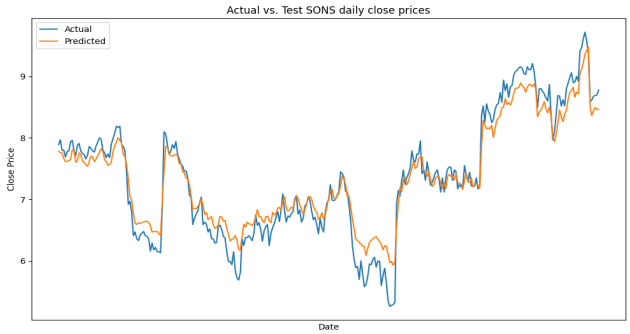

Figure 19. Actual Close Prices vs Predicted Close Prices for SONS

4.1.7 FR

We were able to train a good model for the FR security with training loss of 5.0066 and validation loss of 0.5260.

4.1.8 UFI

We were able to train a good model for the COT security with training loss of 1.3742 and validation loss of 0.4448.

4.1.9 RDCM

We were able to train a good model for the COT security with training loss of 0.3516 and validation loss of 1.3672.

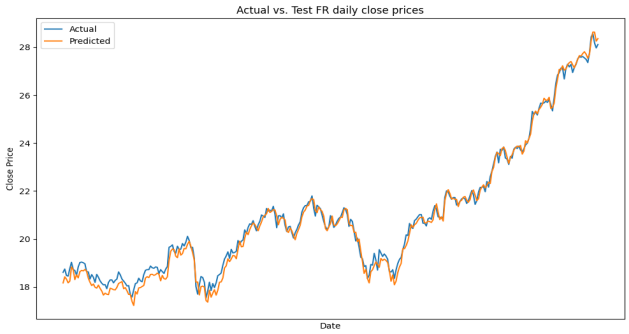

Figure 20. Actual Close Prices vs Predicted Close Prices for FR

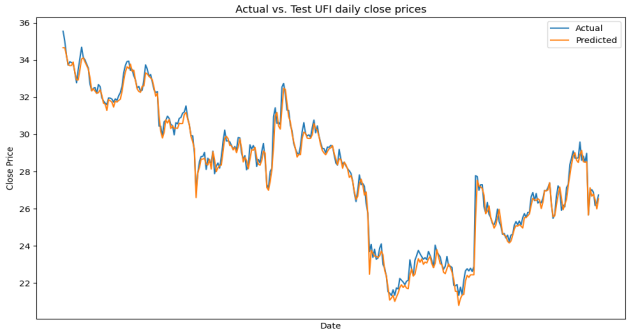

Figure 21. Actual Close Prices vs Predicted Close Prices for UFI

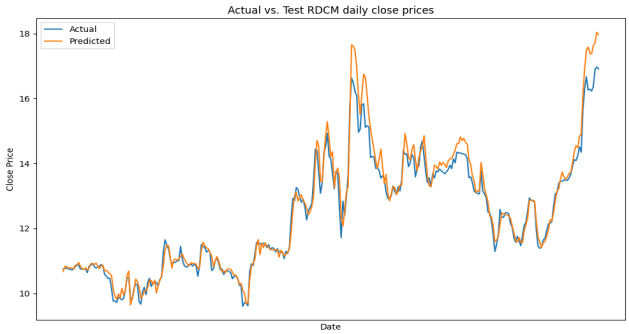

Figure 22. Actual Close Prices vs Predicted Close Prices for RDCM

4.1.10 SOHU

We were able to train a good model for the COT security with training loss of 22.7131 and validation loss of 1.5055.

4.2. Stock Trading

After seeing the results of the above prediction models, we felt the accuracy was good enough to feed into our trading algorithm. This turned about to be true, as we were able to get a really impressive average return rate. For all comparisons in this section we have chosen two well known stock portfolios that have large amounts of data available to use. These are Warren Buffet's portfolio and Ray Dalio's portfolio. For here on we will reference these competing portfolio's as buffet and dalio respectively.

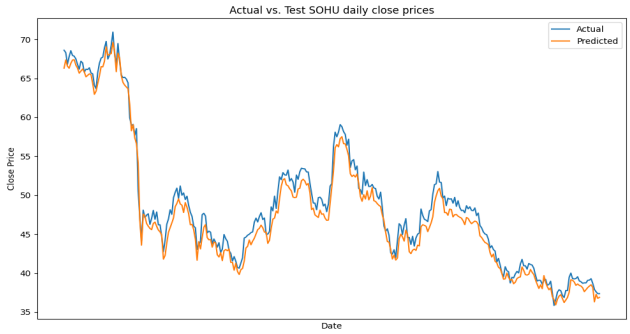

Figure 23. Actual Close Prices vs Predicted Close Prices for SOHU

4.2.1 Initial Conditions

The setup for our stock trading simulation as follows: We initialize the portfolio to contain 25 shares of every asset, as well as $2500 cash. We set the minimum holding amount to 15 shares. We wanted our portfolio to focus on longer term gains so only made trades every twenty days, which represents the approximately twenty days per month the stock market is open. At each trading tick, we look forward another twenty days to try and predict the future prices. We continue in this loop for two years (4/30/2014 - 4/30/206).

4.2.2 Return Rate

The rate of return over time for a trading portfolio is defined as:

$$(\text{Final Value} - \text{Start Value})/\text{Start Value}$$

This number represents the percentage growth over the chosen time slice. Using our models and trading algorithm we obtained a .37 rate of return. Comparing to the buffet and dalio portfolio we get the following graph,

This is a really high return rate. There are a few things to consider. First we are only trading between ten stocks which is a much simpler than the entire world market. Secondly our time period is not short but it also is not very long at only two years. It's likely we the model is exploiting a riskier strategy and capitalizing during a bull market. Further research into longer term testing could show this.

4.2.3 Sharpe Ratio

The Sharpe ratio is a financial metric that is commonly used as a measure of the risk involved with a certain trading strat-

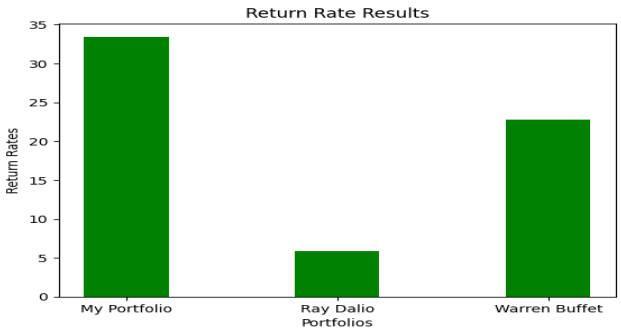

Figure 24. Return Rate versus buffet and dalio

egy. It is calculated as:

$$\frac{\text{Average Return Rate} - \text{Risk Free Return Rate}}{\text{Return Rate Standard Devation}}$$

The risk free return rate represents how much return is expected on a zero risk investment. Commonly used as an example of a zero risk investment are US Treasury bills, or T-Bills. The rate for two year T-bill is 4.2% Using this, we got a final Sharpe ratio of 2.103. Compared to buffet and dalio we get the graph

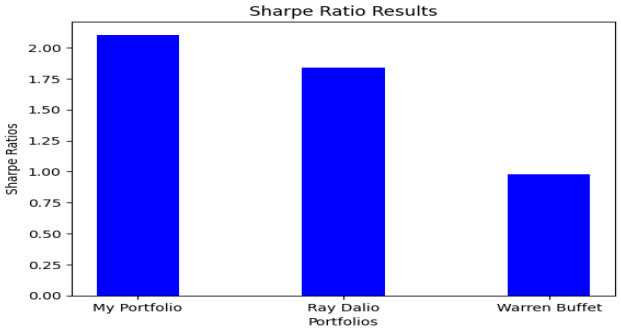

Figure 25. Sharpe Ratio versus buffet and dalio

Generally a Sharpe ratio between $1 - 2$ is considered good, while > 3 is excellent. Our results show a good Sharpe ratio. This is a direct factor of our minimum holding amount, which acts as risk controller forcing a more diverse and thus less risky stock makeup. To improve our Sharpe ratio, we could add more risk controllers into the logic of our trading algorithm.

5. Conclusions & Future Improvements

This paper uses RNN based on LSTM built to predict the price of stocks on any given day. We tested this on stock data of AMTD, CMS, COT, CERN, STAR, SONS, FR, UFI, RDCM and SOHU. And these models have shown promising results, with high testing accuracies. We have performed some experiments on the hyper parameters of the LSTM Architecture such as loss functions, dropout rates, layer normalization, etc. and adopted the best set of hyper parameters in our model architecture by comparing the test time loss.

For the Automated Portfolio Manager we using our custom Time Series Simulation Class and a simple Trading Strategy. The Trading Strategy we used is to sell any stocks that are overvalued and buy undervalued stocks while holding the minimum quantity of each stock to diversify the portfolio. We have simulated the Portfolio Manager using all the above mentioned 10 stocks over a fixed two year testing period with a 20 day look ahead prediction of stock prices.

We also performed experimentation on other stocks like CMS, MSFT, IBM, etc. but we selected the above mentioned Stocks manually for our Portfolio considering their prediction model performance. In the future we can automated logic to select the stocks based on the amount of stock data present, prediction model performance and the correlation between the stocks.

In our experimentation, we found that tuning across many stocks using generic hyperparameters was not a feasible solution. For future work we encourage the use of specific hyperparameters per model.

Our trading algorithm took great advantage of predictions on certain stocks and this led to little diversification in the final portfolio. Despite a large return, it would be more optimal to encourage the trading algorithm to diversify assets. In the future, others might set a maximum percentage of the portfolio that can be allocated to a single asset.

Our models were trained on data from 2005 to 2017. In the future, others could extend this data set far beyond 2005 in the past and pick up on very long historical patterns of the market as a whole and use the latest information from the stock market as part of the test data.

References

[1] Jian Huang. Deep learning in finance and banking: A literature review and classification. *Frontiers of Business Research in China*, 2020. 1

[2] Adil MOGHAR. Stock market prediction using lstm recurrent neural network. 2020. 2

[3] Akhter Mohiuddin Rather. Lstm-based deep learning model for stock prediction and predictive optimization model. *EURO Journal on Decision Processes*, 2021. 2

[4] Jaydip Sen. Stock portfolio optimization using a deep learning lstm model. 2021. 2

[5] Lamiaa Zrara. Portfolio optimization using deep learning for the moroccan market, 2020. 2