Using Deep Learning for Predicting Stock Trends

Arvand Fazeli

Submitted in partial fulfilment
of the requirements for the degree of

Master of Science

Department of Computer Science
Brock University
St. Catharines, Ontario

Abstract

Deep learning has shown great promise in solving complicated problems in recent years. One applicable area is finance. In this study, deep learning will be used to test the predictability of stock trends. Stock markets are known to be volatile, prices fluctuate, and there are many complicated financial indicators involved. While the opinion of researchers differ about the predictability of stocks, it has been shown by previous empirical studies that some aspects of stock markets can be predictable to some extent. Various data including news or financial indicators can be used to predict stock prices. In this study, the focus will be on using past stock prices and using technical indicators to increase the performance of the results. The goal of this study is to measure the accuracy of predictions and evaluate the results. Historical data is gathered for Apple, Microsoft, Google and Intel stocks. A prediction model is created by using past data and technical indicators were used as features in the model. The experiments were performed by using long short-term memory networks. Different approaches and techniques were tested to boost the performance of the results. To prove the usability of the final model in the real world and measure the profitability of results backtesting was performed. The final results show that while it is not possible to predict the exact price of a stock in the future to gain profitable results, deep learning can be used to predict the trend of stock markets to generate buy and sell signals.

Acknowledgements

I would like to extend my deepest gratitude to my supervisor Professor Sheridan Houghten. Thank you for your encouragement, patience and kind support through this process.

I would also like to thank Professor Brian J.Ross and Professor Ke Qiu for their dedication and hard work.

I would like to thank my wife for her support and continued encouragement.

Contents

1 Introduction **1**

1.1 Predicting Stock Returns 1

1.2 Thesis Structure 2

2 Background **3**

2.1 Financial Background 3

2.1.1 Technical Analysis 3

2.1.2 Backtesting 7

2.2 Deep Learning 8

2.2.1 Artificial Neural Networks 8

2.2.2 Recurrent Neural Networks 9

2.2.3 Long-Short Term Memory 10

2.2.4 Training Neural Networks 11

3 Literature Review **16**

3.1 Deep Portfolio 16

3.2 Textual Analysis 17

3.3 Fundamental Analysis 17

3.4 Technical Analysis 18

3.5 Hyperparameter Optimization 20

3.6 The Importance of Backtesting 20

4 Methodology **22**

4.1 Data 22

4.2 Training the model 23

4.2.1 Multivariate Time Series 24

4.2.2 Features 24

4.2.3 Feature Scaling 24

4.2.4 Stacked Long Short Term Memory 24
4.2.5 Reducing Overfitting 26
4.2.6 Overall Architecture 26
4.3 Frameworks and Libraries 28
4.4 Experiments 28
4.4.1 Initial Experiments 28
4.4.2 Final Experiments 29
4.4.3 Hyperparameter Optimization 30
4.4.4 Backtesting 32

5 Results and Discussion **33**
5.1 Calculating Returns 33
5.2 The Effect of Technical Indicators 38
5.3 Hyperparameter Optimization Results 38
5.3.1 Analysis of results 38
5.4 Final Experiments 44
5.4.1 Comparing Results 44
5.5 Testing Other Stocks 45

6 Conclusion and Future Work **46**

Bibliography **52**

Appendices **53**

A Results of other stocks. **53**
A.1 MSFT (Microsoft) 53
A.2 GOOG (Google) 55
A.3 INTC (Intel) 56

List of Tables

2.1 Activation functions 13

4.1 Features 26
4.2 Parameter space for optimizing 30

5.1 The effect of technical indicators. 38
5.2 Ten best results of hyperparameter optimization 40
5.3 Best hyperparameters found after optimization 40
5.4 Comparing the results of conducted experiments. 45
5.5 Comparing ROI for different stocks. 45

List of Figures

2.1 Example of MACD and its histogram [34]. 6
2.2 Buy (green arrow) and sell (red arrow) signals 8
2.3 Perceptron [35] 9
2.4 LSTM Network [37] 11
2.5 Error surface (E) of a linear neuron with two input weight (w1, w2) [36] 12

4.1 Records of AAPL stock price for different days 23
4.2 AAPL closing price for the period of five years 23
4.3 Technical indicators calculated and plotted for initial data. 25
4.4 Architecture of the model 27
4.5 Buy (green marks) and sell (yellow marks) signals. 29
4.6 Talos Hyperparameter tuning workflow [17] (MIT License) 31

5.1 Validation loss for 100 epochs 34
5.2 Original and predicted data for MACD Histogram. 34
5.3 Predicted MACD Histogram with zero line 35
5.4 Log of trades using Backtrader. 36
5.5 Backtesting using Backtrader. Buy and sell signals are generated for the test period (after red line). 37
5.6 Change of MSE during the hyperparameter optimization. 39
5.7 The effect of activation functions on the MSE during hyperparameter optimization 41
5.8 The effect of different optimizers on the Mean Squared Error(MSE) during hyperparameter optimization 42
5.9 The effect of loss function and optimizer on the performance of the model during hyperparameter optimization. 43
5.10 Loss of the model after hyperparameter optimization 44
5.11 Original and Predicted data 45

A.1 Original and Predicted data 53
A.2 Predicted MACD Histogram with cross line 54
A.3 Backtesting the data for test period. 54
A.4 Original and Predicted data 55
A.5 Predicted MACD Histogram with cross line 55
A.6 Backtesting the data for test period. 56
A.7 Original and Predicted data 57
A.8 Predicted MACD Histogram with cross line 57
A.9 Backtesting the data for test period. 58

Chapter 1

Introduction

The goal of this chapter is to briefly explain the problem of predicting stock returns and the proposed solution. The structure of the thesis is found at the end of this chapter.

1.1 Predicting Stock Returns

A stock is a type of security that represents a claim on a part of the corporation's assets and earnings, and a stock market is a place where shares of publicly listed companies are traded. There has been much research on the predictability of stock markets, and although researchers have different opinions, many empirical studies show that some aspects of stock markets can be predicted [30].

There are multiple ways to predict the volatile prices of stocks. Information, including news, tweets, technical indicators, and fundamental indicators, could be represented in such a way to be analyzable by computers. The processed information will later be used to find patterns for predicting future prices. One of the most famous indicators are technical indicators. Technical indicators are mathematical outputs that use prices from the past to gain better insight about future price movements.

For making investment decisions machine learning models can be incorporated to make predictions. A sub-field of machine learning is deep learning. Deep learning is inspired by the structure and function of the brain, and has revolutionized pattern recognition and machine learning in recent years [22]. Deep learning has been the main cause for major improvements in speech recognition, image recognition, and other areas. While it may be hoped to gain similar results in time series prediction, researchers have pointed out that neural networks can be difficult to train, and configurations of parameters must be tuned to improve the performance of deep learning

[40].

In this study, deep learning along with technical indicators are used to predict the price of stocks. A suite of long short-term memory (LSTM) networks is developed for a range of time series prediction. Different architectures are tested to improve the performance, and after the initial experiments, we propose a new solution to create a profitable model: instead of predicting the prices, we focus on predicting the trends. In the last chapter, the difference between the values of LSTM prediction and backtesting is explained and we test the profitability of the model by using backtesting to reach practical results.

1.2 Thesis Structure

The chapters in this thesis are as follows. Chapter 2 gives an introduction to deep learning and technical analysis. LSTM networks are explained and technical indicators are also described. Chapter 3 reviews previous work related to the thesis's goal and the accomplishments made before by other researchers. Chapters 4 and 5 are the primary chapters describing the conducted experiments and results. In Chapter 4, we describe the dataset, and the processing of the data. We also describe the experiment that is designed for building the model and training the dataset. Chapter 5 elaborates experimental results from using LSTM network. Chapter 6 is the conclusion of the thesis and discusses possible future work.

Chapter 2

Background

The purpose of this chapter is to give a brief overview of technical analysis and technical indicators. We will review important technical indicators such as MACD, RSI and Williams %R. A brief introduction of deep learning is later given and LSTM networks are explained.

2.1 Financial Background

A stock is a share that claims an ownership on part of the corporation's assets and earnings. A stock market is a place where these shares are traded. Although there are multiple ways to predict stock markets, the most common ones use either technical indicators, which focus on historical trading data, or fundamental data, which focuses on financial statements such as revenue of a company. Efficient market hypothesis (EMH) [48], states that markets are efficient in reflecting the information about stocks, therefore neither technical analysis nor fundamental analysis can be used to predict the prices. This theory has its own criticisms. In [49], the authors conclude that predictable patterns appear in stock returns and the market is not perfectly efficient.

2.1.1 Technical Analysis

Technical analysis uses past market data to predict the direction of prices [45]. Technical analysis is based on using statistical methods to identify patterns. Other tools such as charts are also used to predict trends [45].

Technical analysis is based on three assumptions [18]:

1. The market discounts everything: Technical analysts believe that the price of stock reflects everything that could affect a company and it is not necessary to

take other factors into consideration.

2. Price moves in trends: A stock price tends to continue past trend movement. Based on the beliefs of technical analysts, prices move in short, medium or long-term trends. This implies that price tends to follow a past trend rather than moving erratically.

3. History tends to repeat itself: Technical analysts believe that investors repeat the same behaviour that is based on emotions such as fear and greed. Because of this, repeated patterns can be identified on charts.

Technical indicators are mathematical calculations that use past price and volume to identify the direction and strength of market trends. They can be broken down into four major types:

1. Trend: Trend indicators, also known as oscillators, indicate the direction in which the price is moving. Moving Average Convergence Divergence (MACD) is a trend indicator.

2. Momentum: Momentum indicators measure how strong the trend is. Relative Strength Index (RSI) is an example of momentum indicator.

3. Volume: Volume is a measurement of the number of units being bought and sold. It is an indication of how strong the movement is.

4. Volatility: It is a degree of variation of prices over a period of time with no indication of direction of price movement.

It is worthwhile mentioning that a few indicators are more favored than the others and have proven to be more useful by past empirical studies. In [32], the authors evaluate the profitability of the MACD and Relative Strength Index (RSI), and concludes that these indicators are profitable for some stocks. The author of [46] states that the efficiency of a back propagation neural network was most improved by addition of the technical index term MACD. In [31] the authors find that the RSI and the MACD outperform the buy-and-hold strategy.

Relative Strength Index (RSI)

To identify overbought or oversold signals, we can use the Relative Strength Index (RSI). The RSI is a momentum indicator that can signal oversold or overbought securities [15]. The RSI can signal oversold or overbought securities. RSI ranges

between 0 and 100; a stock is usually considered overbought when RSI goes above 70 and oversold when it goes below 30. Some analysts use other data ranges such as 80 and 20 or 90 and 10. RSI is typically used on a 14-day time frame and is calculated by the following formula:

$$RSI = 100\frac{100}{(1+RS)}$$
$$RS = \frac{AverageGain}{AverageLoss}$$

Moving Average Convergence Divergence

MACD is a trend indicator to reveal changes between two moving averages of a security price [9]. It is calculated by subtracting 26-day exponential moving average (EMA) from the 12-day EMA. The exponential moving average (EMA) is a weighted moving average (WMA) that gives more weight to recent data. The result will be a 9-day EMA of the MACD, also referred to as signal line. This line can be used as a buying signal when the MACD crosses above its signal line. MACD helps investors understand whether the uptrend or downtrend is getting stronger or weaker [9]. The following formula is used for calculating MACD:

$$MACD = 12\ day\ period\ EMA - 26\ day\ period\ EMA.$$

MACD Histogram

MACD is usually displayed along with a histogram, known as MACD histogram. As shown in Figure 2.1, when the MACD is below the signal line, the histogram (displayed as bars) will be below the baseline and when it is positive the values are reflected on the MACD histogram [9].

The MACD histogram measures the difference between MACD and its signal line (the 9-day EMA). It was created by Thomas Aspray in 1986, and similar to MACD it is an oscillator that moves in positive or negative range. MACD histogram was developed to show crossovers in MACD and generate trading signals [10]. An example is shown in Figure 2.1 and as we can see as the trend changes and moves upwards the MACD Histogram turns positive. Its calculation is as follows:

$$Signal\ Line = \text{9dayEMA}\ of\ \text{MACD}.$$
$$MACD\ Histogram = MACD - SignalLine$$

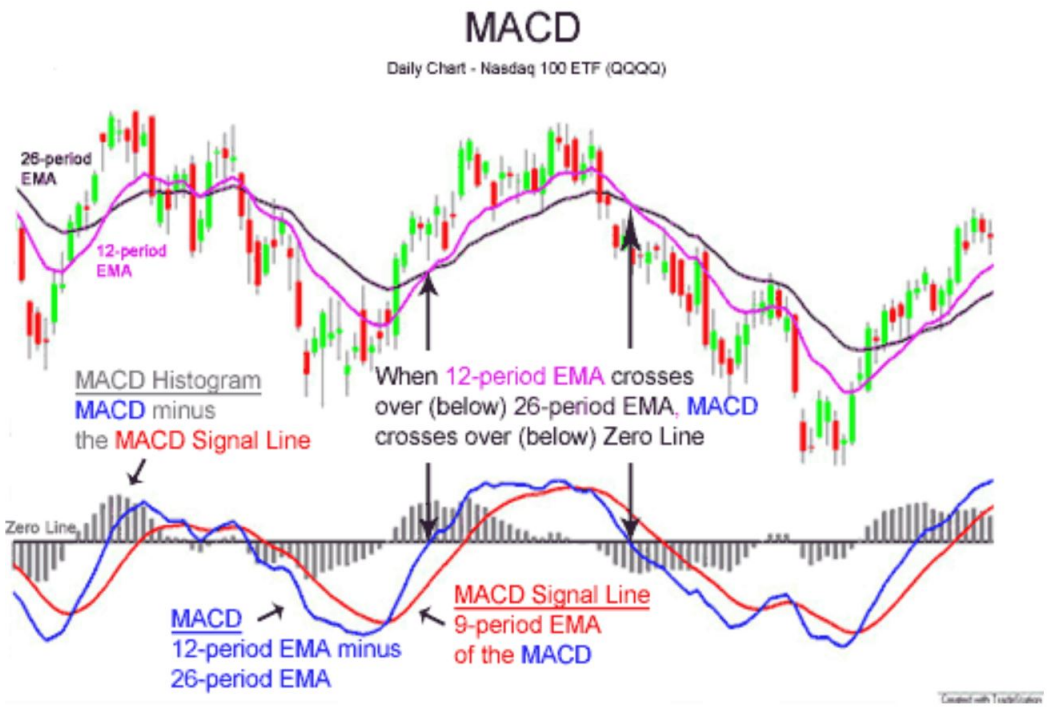

Figure 2.1: Example of MACD and its histogram [34].

The MACD histogram can be used as a potential buy signal when it is below the zero line and begins to converge towards the zero line. It can also be used as a potential sell signal when it is above the zero line and begins to converge towards the zero line [11].

Williams %R

Williams %R can be used to find entry and exit points in the market, it compares a stocks closing price to the high-low range over a specific period, typically 14 days or more [24]. This indicator is helpful in showing the difference between the period high and closing price within the range of days. Williams %R is calculated by the following formula:

$$Williams\%R = \frac{Highest\ High - Closing\ Price}{Highest\ High - Lowest\ Low} * (-100)$$

Highest high is the highest price over the trading period and lowest low is the lowest price over the same period.

Volatility

Volatility is the rate at which the price increases or decreases for a given set of returns. It is used as an indication to the amount of risk related to a security's value. High volatility value is an indication that the price can fluctuate drastically in either direction. Mathematically, it is the standard deviation calculated over a time period. Standard deviations (σ) are measures of how spread out data is. A high standard deviation indicates high volatility. It is important to mention that volatility does not measure the direction of price movements.

$$\text{Standard Deviation } = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}}$$

$x_i =$ Value of the i^{th} point in the dataset

$\overline{x} =$ The mean value of the dataset

$n =$ The number of data points in the dataset

2.1.2 Backtesting

Backtesting is used to measure the performance of a trading strategy. Backtesting works by simulating trades with past data to determine if the trading strategy is profitable or not. A trading strategy is a strategy to trade stocks based on predefined rules [19]. A passive trading strategy is buy and hold, in which the investor buys the stock and holds them for a period of time with the hope that stock will increase in price. Another strategy can be to go long on stocks when there is an upward trend and short on stocks that have a downward trend. If an investor goes long on a stock position, it means that he has bought the stock with the expectation of an increase in value for future time. In contrast, if the investor shorts on a stock he hopes to generate profit from a drop in price. In shorting, the investor first borrows and sells the shares. Later when the price of the stock falls, he repurchases the shares and returns them and gains profit from the drop in price. For this purpose we need to create buy and sell signals, as shown in Figure 2.2; these signals can be generated by a system. If the results of backtesting are positive it can be an indicator that the trading strategy is successful. After training our neural network, we generate buying and selling signals based on the trading strategy. For backtesting we use a platform to simulate and test the strategy. The results give us insight about the performance of our approach in the real world. Based on the results we can modify our approach

Figure 2.2: Buy (green arrow) and sell (red arrow) signals

or the model to improve the results.

2.2 Deep Learning

Deep learning is a sub-field of machine learning. It is a technique that lets computers improve with experience and data. In deep learning, each layer trains a set of data received from the previous layers. It can be described as a feedforward network with many hidden layers.

2.2.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are composed of multiple units also known as perceptrons [52]. Each perceptron imitates biological neurons of the human brain. As seen in Figure 2.3, a perceptron first receives inputs x, and all inputs are multiplied by weights w. At the next step the output is determined by comparing it to a threshold: if the weighted sum $\sum_j w_j x_j$ is less than the specified threshold the output will be zero, otherwise it will be one.

A multi-layer perceptron is called an artificial neural network. Artificial neural networks are composed of multiple perceptrons arranged in layers. The first layer receives inputs and passes it to the middle layers which are called the hidden layers. Values calculated in one layer are passed to the next layers with the first layer taking inputs and the last layer producing outputs. There are multiple types of neural network with different architectures, used in different situations for various

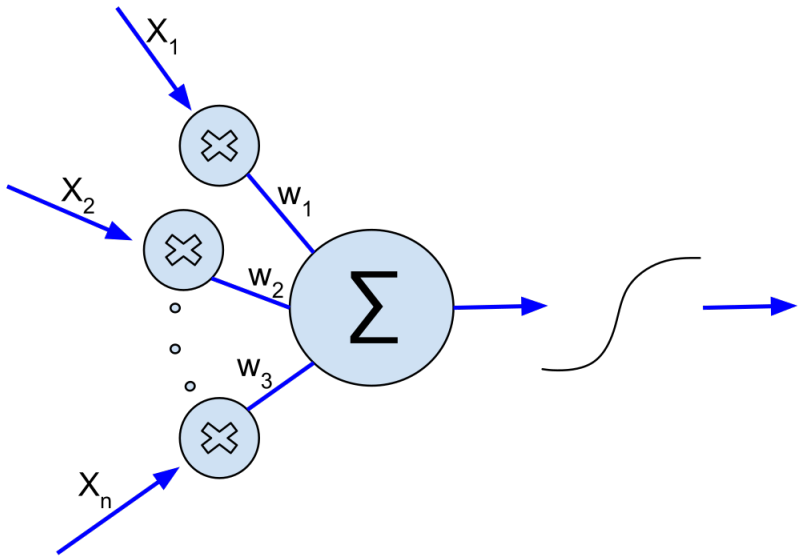

Figure 2.3: Perceptron [35]

purposes. The most basic type of neural net is the feedforward neural network, in which information is just passed forward. More information about neural networks can be found in [52]. Another commonly used type of neural network is the recurrent neural network, which is usually suitable for time series data. Recurrent neural networks (RNN) use internal memory to process a sequence of data. There is also convolutional neural network which is widely used for image recognition. Selecting the suitable architecture depends on the problem's domain and specific needs.

2.2.2 Recurrent Neural Networks

RNNs are multi-layer networks where the connections between the nodes create a directed cycle. RNNs take as their input not just the current input but also what they have perceived previously in time. The decision a recurrent neural network reaches at time step $t-1$ affects the decision it will reach one moment later at time step [41]. Forward propagation begins with a specification of the initial state $h^{(0)}$. Then, for each time step from $t=1$ to $t=\tau$, the following equation can be applied:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = tanh(a^{(t)})$$
$$o^{(t)} = c + Vh^{(t)}$$
$$y^{(t)} = softmax(o^{(t)})$$

where b and c are bias vectors, and weight matrices U ,V and W, respectively are for input-to-hidden, hidden-to-output and hidden-to-hidden connections [41]. Bias units are a set of weights that help to shift the activation function. During processing, the previous hidden state is passed to next step of the sequence. The hidden state acts as the neural network's memory, otherwise known as a transition matrix. The weight matrices are used to determine the importance of present and past hidden state. The errors they generate will return via backpropagation (explained in Section 2.2.4) and be used to adjust their weights until it reaches a minimum value [1].

2.2.3 Long-Short Term Memory

The problem of preserving long-term information and skipping short-term input has existed for a long time. One early approach to address this issue was the long-short term memory (LSTM) introduced by Hochreiter and Schmidhuber [43].

To control a memory cell a number of gates are needed. An output gate are needed to read out the entries from the cell. An input gate is needed to read data into the cell. To reset the contents of the cell, a forget gate is used. The motivation is to design a system that can remember and ignore inputs as needed. In addition to the three gates, memory cells are introduced that take the same shape as the hidden state. This can be considered as a fancy version of states to record additional information [57].

LSTM can add or remove information from cell states. It can make modifications by multiplying and adding. When information flows through cell states, LSTM can selectively remember or forget things. Gates can control what information is let through by using a sigmoid neural net layer and a pointwise multiplication operation [20].

The first step in an LSTM is to determine the information that is going to be thrown away using the forget gate layer, (also known as sigmoid layer). The output is generated based on h_{t-1} (hidden state) and x_t. The forget gate (f_t) is calculated as follows:

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$

where σ is the sigmoid function, W_f is the weight for the gate, x_t is the input for current time and b_f is the bias value for the gate. Bias is the output when there is no input. The final result is a number between 0 and 1. Later, a number is stochastically generated and if it is lower than the predefined probability, we forget the gate. At

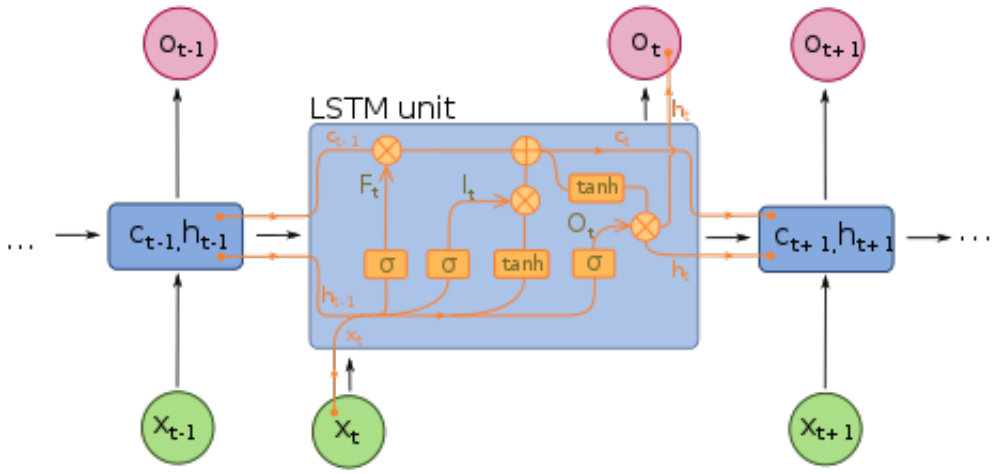

Figure 2.4: LSTM Network [37]

the next step, the input gate layer is used to decide which values will be updated and the *tanh* function creates new candidates known as $\widetilde{C}_t$. After that C_{t-1} gets updated by C_t. The following two equations are applied:

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

$$\widetilde{C}_t = \tanh(W_c.[h_t, x_t] + b_c)$$

At the last step, we run a sigmoid layer to decide the parts of cell states are going to be filtered and after that the cell state is used as an input for the *tanh* function. The result is multiplied by sigmoid gate to output the needed parts [20], using the following equation:

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

The complete architecture is shown in Figure 2.4.

2.2.4 Training Neural Networks

By training neural networks, we are trying to solve an optimization problem. Each connection between neurons, also known as units, initially has an arbitrary weight, and the goal is to optimize the weights within the model. These weights will be constantly updated in order to reach optimal values. The approach that determines

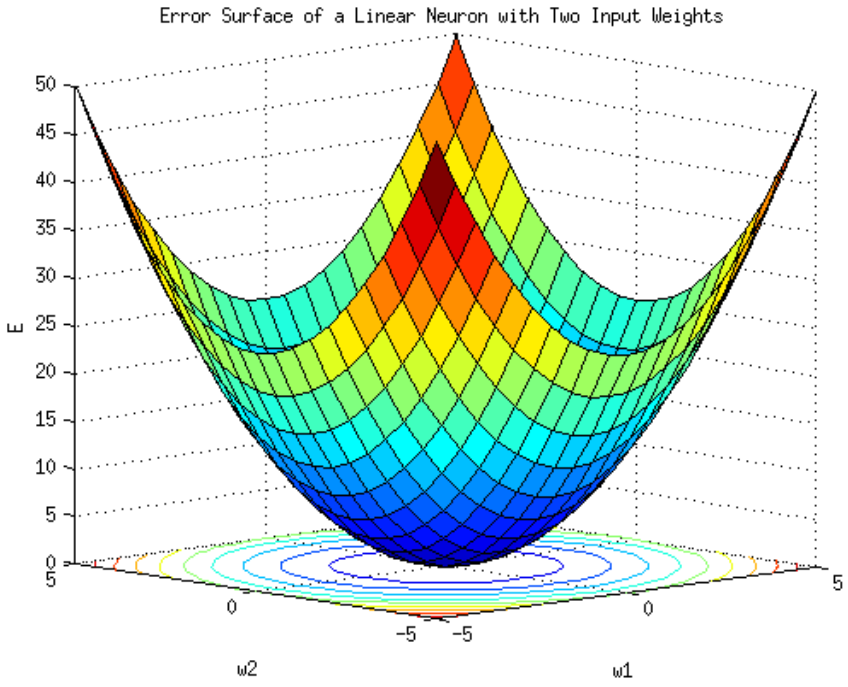

Figure 2.5: Error surface (E) of a linear neuron with two input weight (w1, w2) [36]

how these values should be updated is dependant on the optimization algorithm, and the goal is to minimize the loss function. An example of an optimization algorithm is stochastic gradient descent (SGD) [29]. Backpropagation is used to find the steepest descent direction in an optimized approach. The training process is iterative; during each iteration the weights are updated. As shown in Figure 2.5, the slope of the error surface is used to choose the direction of the next step. For training the model the data is split into three sets: training, validation and test sets. The validation dataset is used for tuning the model's hyperparameters [23].

Activation Function

The activation function is used to determine the output of the neurons of the neural networks. It maps the resulting values to a value between 0 to 1, or depending on the activation function, it can map to 1 to -1.

An activation function commonly used in deep neural networks is rectified linear unit (ReLU) [50]. ReLU function converts the values to zero if the input has a value below zero and when the value increases the output will have a linear relationship with the input of the variable. Leaky RelUs introduce a small negative slope which

Name	Function
ReLU	$\phi(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$
Leaky ReLU	$\phi(x) = \begin{cases} \alpha x & x \leq 0 \\ x & x > 0 \end{cases}$

Table 2.1: Activation functions

prevents the output from being zero when $x < 0$ [3]. As an example, Leaky ReLU may be $y = 0.03x$ when $x < 0$. The formulas are shown in Table 2.1.

There is also the softmax function [41], which is used before the output layer. The softmax function is often used to normalize the input value [41]. It is calculated as follows:

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$$

where exp is the exponential function.

Optimization Algorithm

Choosing the right optimization algorithm can have huge impact on the performance of the model. The classic optimization method is stochastic gradient descent (SGD), which is an iterative method that randomly selects samples to find the optimal values [57] . Another popular optimization method is Adam. Adam, calculates the learning rate the based on the first and second moment of the gradients [44]. The method requires less memory and is suited for problems with large data and/or parameters. It is also a suitable choice for non-stationary objectives and problems with very noisy gradients [44]. Nadam optimizer incorporates Nesterov accelerated gradient into Adam, which allows larger decay rate compared to momentum. Nesterov accelerated momentum makes bigger moves into direction of the previous accumulated gradient and then measure the gradient [38]. In regular momentum the gradient is calculated by running the following iterative scheme :

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} - \nabla_\theta f(\boldsymbol{\theta}_{t-1})$$
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \gamma_t \mathbf{v}_t$$

where γ_t is a learning rate schedule, β is the damping coefficient, which determines how quickly the momentum vector decays. Nesterov momentum improves convergence and stability and calculates the results with a momentum-based method with

the following formula [47]:

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} - \nabla_\theta f\left(\boldsymbol{\theta}_{t-1} + \gamma_{t-1}\beta \mathbf{v}_{t-1}\right)$$
$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \gamma_t \mathbf{v}_t$$

Loss Function

A loss function is a measurement of how well our model is performing. A high loss function implies that the predictions are not working. An optimization problem tries to minimize the loss function. A basic example of a loss function is mean squared error (MSE). To calculate MSE, as shown in the following formula, we square the difference between the predictions (y_i) and the actual values ($\tilde{y}_i$), and average it out across the dataset.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Another loss function is Huber loss, which is mostly used for regression and is less sensitive to outliners than MSE [6]. Huber loss is preferred in certain cases for which there are both large outliers; an outlier is a data point that differs significantly from other observations. Huber loss accepts the hyperparameter δ, which is a configurable parameter. The definition is as follows:

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

where $f(x)$ is the predicted value and y is the actual value.

Reducing Overfitting

Deep neural networks are powerful tools to learn complex tasks. However, the model may be overfitted. When the difference between the training error and test error is too big, overfitting has happened [41]. There are couple of ways to avoid overfitting, and in our approach we will be using dropout and added noise.

The dropout technique is essentially a regularization method used to prevent overfitting while training neural nets. By applying dropout to the hidden layer, we are essentially removing each hidden unit with probability p. This can result in a network with a subset of original neurons [57].

Another approach to improve generalization error and to optimize the structure of the mapping problem is to add random noise. The amount of noise is a configurable

hyperparameter. Noise is only added during the training and not during the validation or testing of the dataset.

Hyperparameter Optimization

Hyperparameter optimization aims to find a set of hyperparameters in order to minimize the generalization error [33]. This is a difficult task due to the dimensionality of the hyperparameter space. However, in the machine learning field, often a subset of the hyperparameters usually play a significant role with high impact [56]. The parameters are optimized by inputting the data and measuring the error until the best set is found. The hyperparameter optimization that is used in our approach is grid search. Grid search is an exhaustive search through a manually predefined subset of hyperparameter space. Grid search constructs all possible combinations for a set of parameters. As an example we can consider the following parameters:

$$p_1 = [a, b, c]$$
$$p_2 = [1, 2, 3]$$
$$p_3 = [\alpha, \beta, \theta]$$

Grid Search will create 3^3 combinations, including: $[a, 1, \alpha], [a, 1, \beta], ...[c, 3, \theta]$, and processes all these combinations.

Chapter 3

Literature Review

Predicting the stock market has been the subject of many studies. Dealing with the wide variety of data sources to create a prediction tool is a daunting task. However, deep learning has shown great advantages in processing non-stationary data, and has been used more recently in the finance realm. In this chapter we will review previous works with the focus of using deep learning for stock prediction.

3.1 Deep Portfolio

In [42], the authors propose a new method to build deep learning hierarchical decision models that can include complex features. To do so, a framework needs to be set up to train the data. After that, a four-step algorithm is defined for model construction to build deep portfolios and create an automated process to select portfolios.

The dataset is divided into three subsets: training, validation and testing. To avoid overfitting, a regularization penalty is added. For large datasets, mini-batch stochastic gradient descent is used to perform optimization. To implement the framework a four-step process is used, consisting of auto-encoding, calibration, validation, and verification. The algorithm aims to create deep hierarchical compositions of portfolios that are constructed in the encoding step. Deep factors can be represented as compositions of financial put and call options for the univariate activation. During the calibration a non-linear portfolio is created by using the returns, risk free rate and portfolio weights. At the first step, each stock is ranked by the measure of its similarity to other stocks (communal information), for choosing stocks, the 10 most communal (similar) stocks are chosen. Some non-communal stocks are also picked. This can be treated as adding non-learned features. Market-cap is used for index weight. Model selection (i.e. verification) is conducted through comparison. As an

example, this method is tested on the IBB Index. The results show that the validation error is higher in the test set. By reviewing the verification results it was suggested that to achieve reliable predictions a deep portfolio of at least forty stocks is needed.

It was shown that deep learning may have the potential to dramatically improve the predictive performance in conventional applications.

3.2 Textual Analysis

Textual analysis is an approach that involves understanding the text to gain information. This approach could be used to determine whether an expression is positive, negative or neutral. In [55], the author analyzes the influence of news articles. The information is gathered from websites like Twitter or Reddit. After downloading the headlines, the stock trend is correlated with headlines to find the effect of the headlines on the trends. The problem is reduced to predicting whether a stock price rises or falls. Recurrent neural networks were used to map the function between sentiment values and the target price. The news headlines are first preprocessed and passed as inputs to the neural network. The performed experiments tried to predict the stock prices using information from both numerical analysis and textual analysis. The numerical analysis was performed using LSTM. This resulted in a MSE of 0.00045. After that, textual analysis was performed on the news headlines. The author claims 78% accuracy in predicting their influence on the stock prices. When the results from textual analysis are augmented over the predictions from numerical analysis, the model resulted in 0.00037 MSE. The thesis concludes that adding textual information from news to the stock price data would substantially improve the prediction accuracy.

3.3 Fundamental Analysis

In [28] the authors propose an investment strategy for creating portfolios based on predicted future fundamental indicators. The authors suggest that the long term performance of an investment depends on how the stock is currently priced with respect to its future fundamental indicators. In previous experiments, the authors concluded that predicting the relative returns directly with RNN is not feasible and does not out-perform a linear model. In a quantitative strategy, stocks are sorted and portfolios are created with stocks with the highest points. Stocks can be valued by taking fundamental data into account. High value factor ratios are called *value stocks* and those with low ratios are called *growth stocks*. Portfolios of stocks which

give more weight to *value stocks* have significantly outperformed portfolios that give more weight to *growth stocks* over the long run. Fundamental data, such as revenue, operating income and debt, are gathered. Computed features of the reported data are analyzed, based on these values. Using deep learning, future fundamentals are forecasted based on a trailing 5 year window. Quantitative analysis demonstrates a significant improvement in MSE over a naive strategy. They run a simulation to assess future financial reports, and by applying earning yields (EBIT/EV) during a 12 month period they achieve a 44% annual return.

3.4 Technical Analysis

Technical data is gathered directly from the market and can be used for trading by generating buy, sell and hold signals.

In [25], the authors use deep learning to predict one-month-ahead stock returns in a cross-section of the Japanese stock market, and compare the performance of this method to shallow neural networks and machine learning models. The cross-section tries to find the reasons why stock A has higher or lower returns than stock B. This is in contrast with time-series in which the average returns change over time. The predictive stock returns (scores) have been calculated from the information of the past five points of time for 25 factors (features) for the MSCI Japan Index. By inputting data of these factors and passing them through many layers, useful features have been extracted and prediction accuracy for future stock returns has been improved. As a measure of the performance, the authors use rank correlation between the actual out-of-sample returns and their predicted scores and directional accuracy. In the end, three approaches have been compared. Conventional three-layer neural networks, support vector regression and random forests are identified as representative machine learning techniques. Before describing the methodology, the authors study related works in this area. The dataset for this test was gathered from the MSCI Japan Index. Twenty-five factors including: book-to-market ratio, earnings-to-price, dividend yield, sales-to-price, cash flow-to-price, etc. are used to perform the calculations.

The problem has been defined as a regression problem, and mean squared error is used as a loss function. The model is trained by using the latest 120 sets of training from the past 10 years. For measuring the performance, rather than using the values of loss function directly, rank correlation coefficient (CORR) and directional accuracy (direction) are used because these are more relevant measures of performance than the

loss function. CORR measures the strength and direction of association between two ranked variables and rank correlation between the actual out-of-sample returns and their predicted scores is used. Direction accuracy compares the forecast direction to the actual realized direction. The performance of a long-short portfolio is compared with support vector regression and random forests. The long-short portfolio strategy is to buy the top stocks with higher prediction scores and sell the bottom stocks with lower prediction scores.

A total of 8 patterns of deep neural networks (DNN) with 8 layers (DNN8) and with 5 layers (DNN5) are examined and TensorFlow is used for the implementation. It is seen that the loss of the model increased when fewer layers were used and that a higher number of layers resulted in higher CORR and direction.

Next, an ensemble methodology is used to combine different machine learning models to determine if the results would improve, but it is observed that the improvement is limited. In conclusion it is observed that with more layers the prediction accuracy of cross-sectional stock returns increases. Four patterns of DNN outperform the CORR of both support-vector regression and random forests. For future research recurrent neural networks, which are a suitable choice for time series data, were recommended.

In [54], the researchers use an artificial neural network for creating a trading system by using technical indicators. To predict the stock market among 100 technical indicators, the three most important ones are chosen and a multilayer perceptron is used to predict buy-sell signals by analyzing time-series data. A multilayer perceptron (MLP) is a class of feedforward artificial neural network. The data is gathered from Dow30 stocks for the period from 1997 to 2017. To perform the experiment, data was gathered from Dow30 stocks, was obtained from “Yahoo! Finance”. The aim is to predict buy, and sell points of the stock prices by using a multilayer perceptron. To model the data, hold, buy and sell signals are generated based on peak and valley points. The training is performed for prices between dates of 1997 to 2006 and test data is from 2007 to 2017. It may be hard to beat the buy and hold strategy over long periods of time, however in comparison to buy and hold the model provides mixed results, with some being better and some worse. The average annualized return is 10.3%, while the average annualized return of buy and hold strategy is 13.83%. The paper suggests that individually fine tuning the technical indicator parameters for each stock might improve overall performance. The experiments show that even without optimizing the model, the results are comparable to the buy and hold strategy.

In [39], the authors use deep learning to explore its potential in a time series prediction problem. Average monthly statistics for the S&P 500 split by industry are gathered from January 1990 until October 2015.

LSTM networks are used to determine the directional movements. They create a set of features for training and create a classification problem, response variable Y^s_{t+1} for each stock s and date t that can take on two different values. Class 0 is realized if the one-period return R^1, s_{t+1} of stock s is smaller than the cross-sectional median return of all stocks in period $t+1$. Class 1 is realized if the one-period return of s is larger than or equal to the cross-sectional median. The prediction is the probability of $P^s_{t+1|t}$ for each stock s to either out-or under-perform cross-sectional median in period $t+1$. After that, they sort the values and go long on stocks with the highest probability and short on stocks with the lowest probability. With daily returns of 0.46 percent and the Sharpe ratio (a measurement to understand the return of an investment compared to its risk) of 5.8 prior to transaction costs, they find LSTM networks outperform memory free classification methods and show that deep learning can be deployed in this domain.

3.5 Hyperparameter Optimization

One research example that considered optimizing parameters and design choices for the correct hyperparameter optimization is [53]. The authors test the importance of different network design choices and hyperparameters. By evaluating 50,000 different configurations, they found that some parameters, for example the last layer of the network, have a large impact on performance. Other parameters, such as the number of LSTM layers are of minor importance. To summarize their conclusions, variational dropout was on all tasks superior to no-dropout or naive dropout. Adam and Adam with Nesterov momentum (Nadam) (explained in Section 2.2.4) usually performed the best of those examined. SGD (explained in Section 2.2.4) did not produce good results in most of the cases. During the experiments, they looked at one dimension for a certain hyperparameter. However, hyperparameters can influence each other and the individual best options may not necessarily lead to the global optimum.

3.6 The Importance of Backtesting

As mentioned before, backtesting works by simulating trades with past data to determine if the trading strategy is profitable or not. It is a process to see how accurately

the method would have predicted actual results. While there are articles that mention the results of their loss function as a success, it should be mentioned that generating good results for the loss function does not necessarily mean that the proposed approach would be profitable in the real world. Therefore it is better to simulate the strategy and perform backtesting. In [26], the authors search for accuracy using different statistical measures. After tuning the hyperparameters, they reach the conclusion that it may not be possible to predict the adjusted closing price solely based on the open, high, low, close and adjusted closing price. They run the LSTM algorithm based on backtesting data. Although the LSTM model had an accuracy of 80% on predicting the adjusted closing price, they suggest that it was a naive conclusion that their model can do an excellent prediction of the market. It was seen that predictions did not perform well when feeding unseen data in an iterative process.

Chapter 4

Methodology

In this chapter we will propose our approach for solving the prediction problem. Our initial goal was to predict the price of a stock s at time t in future, but due to unsatisfactory results we modified our approach to concentrate on trend prediction. Later we optimize the hyperparameters of the model to improve the results. We will describe the metrics for evaluating the performance of the model and the details of the new approach.

4.1 Data

Our data is chosen from one of the companies from S&P 500. We select a stock that has a shift in major in trends for the last 120 days, which is equivalent to our test data size. Data is downloaded from 3/13/2014 until 3/12/2019 for Apple Stock (AAPL) from "Yahoo! Finance" to conduct the experiments. Our data is the stock price over approximately five years and as shown in Figure 4.1. The data consists of six columns:

1. Open: The price the stock started trading at when the exchange opened.
2. High: The highest price the stock has seen during the day.
3. Low: The lowest price the stock has seen during the day.
4. Close: The stock price at the last close of the market.
5. Volume: The volume is the number of shares that changed hands during a given day.
6. Adjusted Close: The adjusted closing price also factors for dividends.

	Date	Open	High	Low	Close	Adj Close	Volume
2	2014-03-13	76.777145	77.094284	75.594284	75.807144	67.034889	64435700
3	2014-03-14	75.541428	75.841431	74.714287	74.955711	66.281975	59299800
4	2014-03-17	75.385712	75.709999	75.121429	75.248573	66.540955	49886200

Figure 4.1: Records of AAPL stock price for different days

Each data in a column is converted into an array, and technical indicators are created based on these values.

4.2 Training the model

The data, plotted in Figure 4.2, shows the stock price of Apple (AAPL) indexed by the date starting from 3/13/2014 until 3/12/2019. Training the model is a process that includes comparing the actual value with the value generated by randomly assigned weights. Backpropagation is used to train the network, see Section 2.2.4. During this process the weights of neurons are updated based on the previous epoch or iteration.

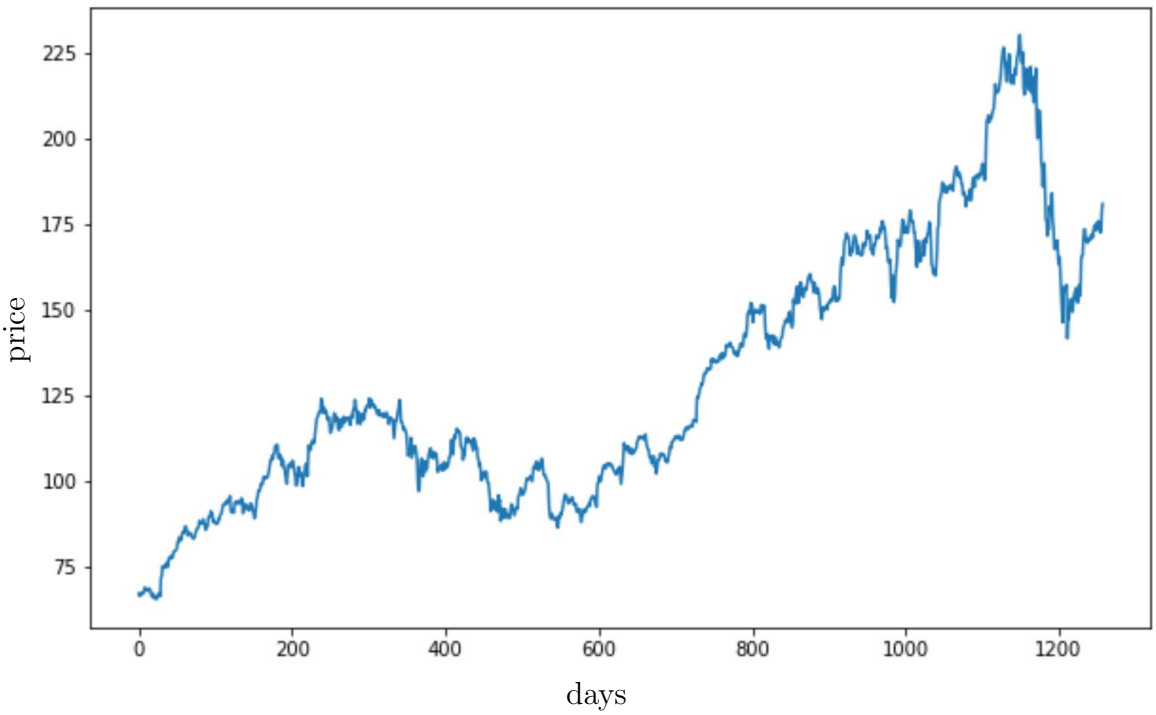

Figure 4.2: AAPL closing price for the period of five years

During training, the number of epochs is counted by multiplying number of iterations by batch size. An epoch, which is useful for periodic evaluation is one pass over

the entire data [21].

4.2.1 Multivariate Time Series

By using LSTM networks it is possible to forecast data using multiple input variables. In a univariate time series, the forecast depends on one time dependant variable. Unlike univariate time series, multivariate time series can have multiple variables. Each of these variables depend on their past value and can be dependent on other values.

4.2.2 Features

To create the model we use a list of features shown in Table 4.1. Three of these features are technical indicators (RSI, Volatility, Williams %R). These indicators were created using the initial data and are plotted in Figure 4.3. We will later examine the effect of these indicators on the performance of the model.

4.2.3 Feature Scaling

To standardize the range of features, we used scaling. To scale the features we used min-max scaling, with a range of -1 to 1. By using feature scaling we normalize the range of values The formula is:

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

In the formula, x is an original value and x' is the normalized value.

4.2.4 Stacked Long Short Term Memory

Recurrent layers can be stacked on top of each other. In a gated recurrent unit(GRU), the hidden state is passed from one layer to the other. This makes the GRU to learn transformations [27]. GRUs performances are generally on par with LSTMs [27]. Stacked LSTMs can be defined as multiple LSTM layers used in sequential order. The first layers of LSTM return their full output sequences, but the last one only returns the last step in its output sequence, thus dropping the temporal dimension [5]. In our model four layers of LSTM are used to improve performance.

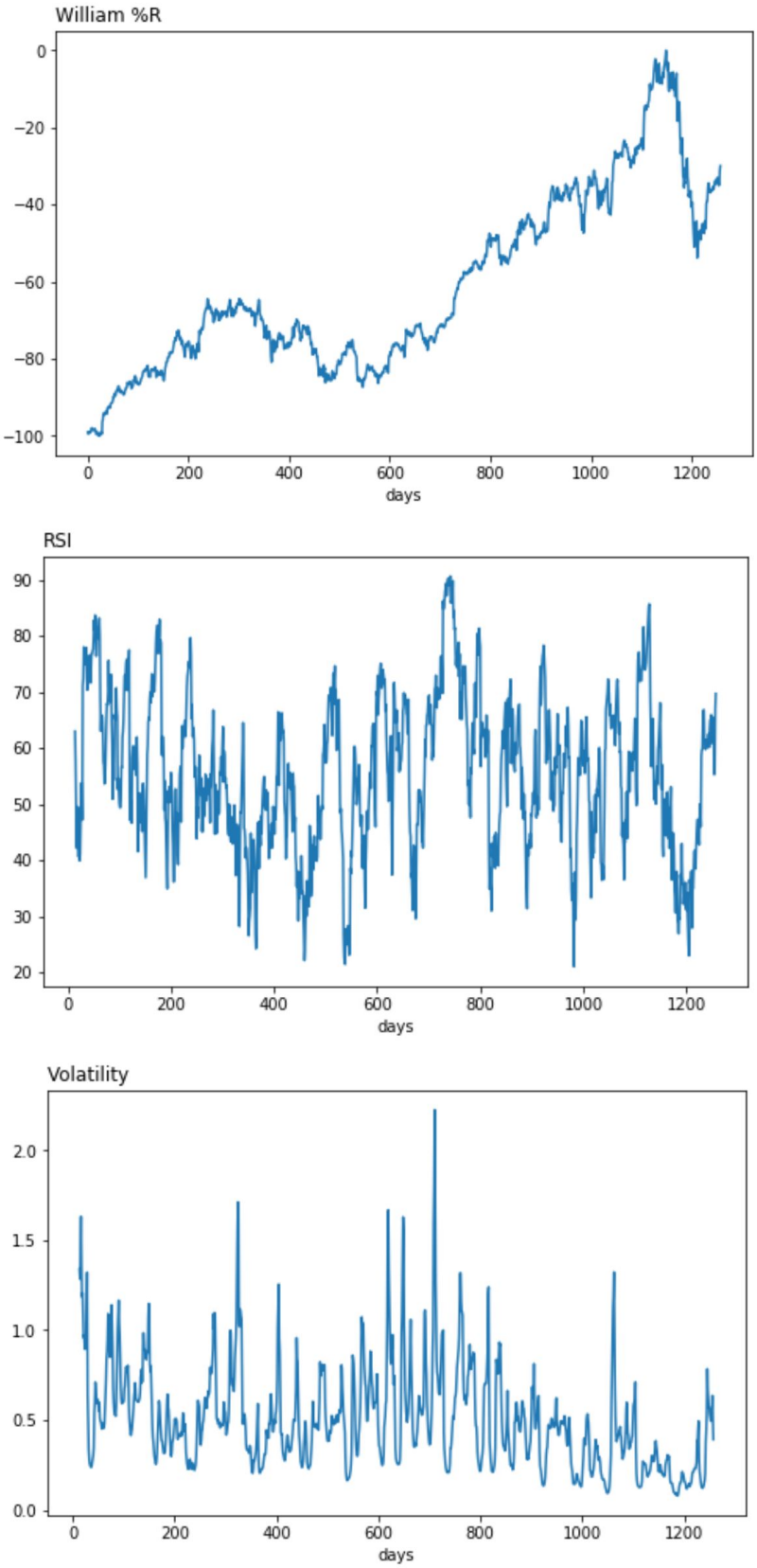

Figure 4.3: Technical indicators calculated and plotted for initial data.

	Name
1	Opening Price
2	High Price
3	Low Price
4	Closing Price
5	Adjusted Closing Price
6	Volume
7	Volatility
8	Williams %R
9	RSI

Table 4.1: Features

4.2.5 Reducing Overfitting

As mentioned in Section 2.2.4, dropout prevents overfitting and the term refers to removing a hidden unit temporarily from the network. Dropout in our model is a configurable value that indicates the number cells to be dropout during the process.

Gaussian Noise is added during training in order to regularize the layer. The amount of noise added is a configurable hyperparameter. Injected gradient noise causes improvement in different models [51].

4.2.6 Overall Architecture

The overall architecture of the network is shown in Figure 4.4 and consists of four layers of LSTM networks. To create the input data, features are scaled and sliding windows are created. After that, the data is reshaped for input of LSTM network. The first layer consists of 100 units, with the dropout of 0.2 initially set. Later this value will be optimized through hyperparameter optimization. We add the gaussian noise of 0.05 to the model and batch normalization added to the model. The second Layer has 200 units with the same dropout value and the third and fourth layers are the same as the second layer. The input format is $(batch\,size, times\,steps, input\,dimension)$ and the batch size (none) will be set later in training of the model.

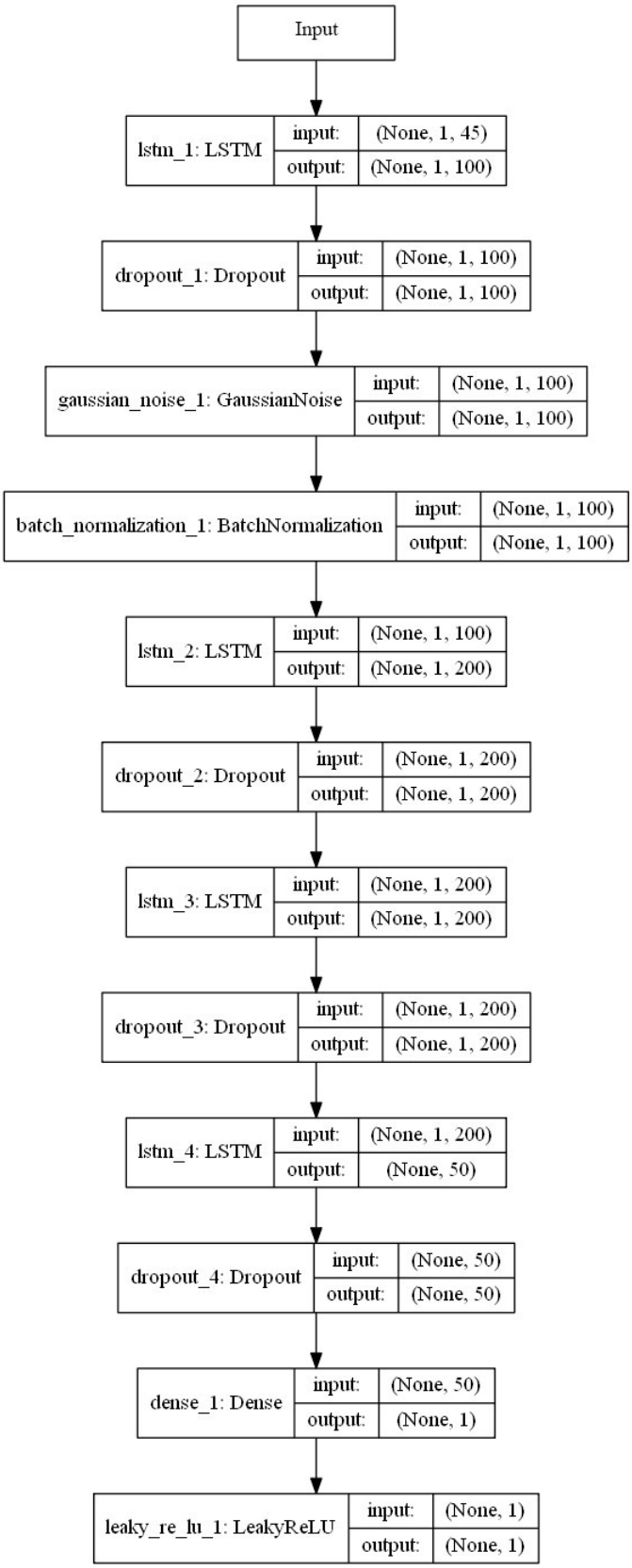

Figure 4.4: Architecture of the model

4.3 Frameworks and Libraries

To train the model we used Keras framework [8]. Keras is a Python framework, which provides neural networks API by running on top of TensorFlow, CNTK, or Theano [8]. Our framework will also be using TensorFlow [13] on top of Keras. TensorFlow is a free software library focused on machine learning. It uses graph structures and each edge between nodes is a multidimensional tensor. Other Python libraries such as Pandas [14], Numpy [4], Scikit-learn [12], etc. were also used to process the information. Other libraries such as Backtrader [2] and Talos [16], which will be discussed later, were used for backtesting and hyperparameter optimization.

4.4 Experiments

We begin our experiments by trying to predict the price of a stock in future time. The stock we examine is AAPL and we will later examine the effect technical indicators have on the loss of our model and then we will optimize the hyperparameters through grid Search and measure the performance of the model. To measure the performance of the model in the real world we will be using backtesting. To further examine the functionality of the final model, we will test the approach on three other stocks.

4.4.1 Initial Experiments

Our initial goal was to predict the price of a stock at a certain point in time. After creating the model and optimizing it, we generated the buy and sell signals as shown in Figure 4.5. After that backtesting was performed, and surprisingly, even when the commission was set to 0, the results showed that the value of the portfolio decreased. We tried to penalize the model for loss of value and add threshold but again the results were not satisfactory. One reason for the failure of the experiments was that the loss function was set for the price instead of the profit, in other words a lower MSE did not result in higher profit. The other reason could be that the neural networks failed to find an accurate pattern in day to day price changes.

We changed our experiments with the next focus being on predicting the price change. The aim was to predict if the price change would be positive or negative for each day. But again, after training the model, the results of the backtesting indicated that the initial portfolio decreased in value.

The third phase of the initial experiments focused on predicting the RSI. That resulted in just one buying signal.When the trading strategy was changed with closer

overbought and oversold signals, it did not result in an increase in the portfolio value.

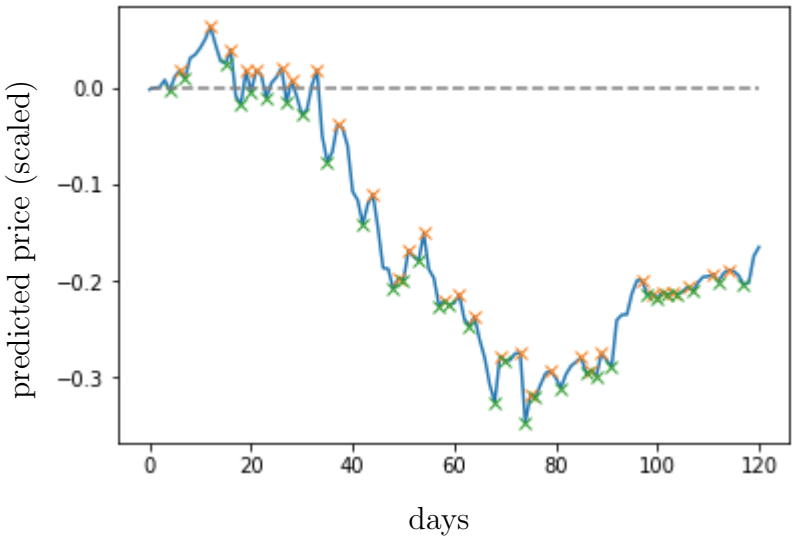

Figure 4.5: Buy (green marks) and sell (yellow marks) signals.

4.4.2 Final Experiments

Our later experiments focused on predicting the trend of the market. One good indicator for predicting the trends is the MACD histogram. As mentioned before in previous chapters, a trading strategy that uses the MACD histogram can be used to generate a buying signal when the MACD histogram moves from zero to a positive value and a sell signal when the price change crosses below zero.

The data was split into test, training and validation sets. Ten percent of the data, which equals approximately to 120 days, starting from 2018/09/10, is set as test data, and the rest is used for training. The validation will be 10% of the training data.

Before passing the data we must reshape our inputs. The input of the LSTM has three dimensions: number of samples, time samples and number of features. In the model, the mean squared error function was used as the loss function, and LeakyRelu was set as the activation function. Adam (Section 2.2.4) was set initially as the optimization algorithm. Once we have a suitable model to predict a day ahead, we tune the hyperparameters and select the best set of configurations.

The number of epochs is set to 100 and batch size to 32. An epoch is a pass through the whole data. Because the optimization is an iterative process, it is necessary to go

over the training set multiple times. The data is divided into smaller batches before being fed to the neural networks. We will also be reducing the learning rate when our metric (MSE) has stopped improving. This was done because as noted in [7] : Models often benefit from reducing the learning rate by a factor. The ReduceLROnPlateau callback in Keras monitors a quantity and the learning rate will be reduced if no improvement is seen for a defined number of epochs.

4.4.3 Hyperparameter Optimization

After creating a working model, we optimize hyperparameters. To do so we must choose which hyperparameters we want to optimize during the process. Based on previous research, we know that a few parameters have a bigger impact on the results than the others. The parameters we focused on are shown in Table 4.2. Based on previous research (including [53]), these values were selected to create a parameter space for Grid Search.

Parameter	Values
Dropout	From 0.1 to 0.5 in 5 steps
Optimizer	Adam, Nadam, SGD
Loss Function	Huber Loss, Mean Squared Error
Activation Function	ReLU, LeakyReLU

Table 4.2: Parameter space for optimizing

There are three different optimization strategies: grid search, random search and probabilistic reduction. We will be using Grid Search which scans the data with a set of predefined hyperparameters. It builds a model on each parameter configuration and ultimately selects the model with best performance. The framework that we will be using to optimize the hyperparameters is Talos [16]. Talos is an open source framework, and is used for hyperparameter optimization with Keras models. After an experiment is started, a scan object is created which will be used in the main program. By using the parameter space, the framework yields the next permutation through multiple iterations until all permutations of the parameter space are processed. The hyperparameter tuning workflow of Talos is shown Figure 4.6.

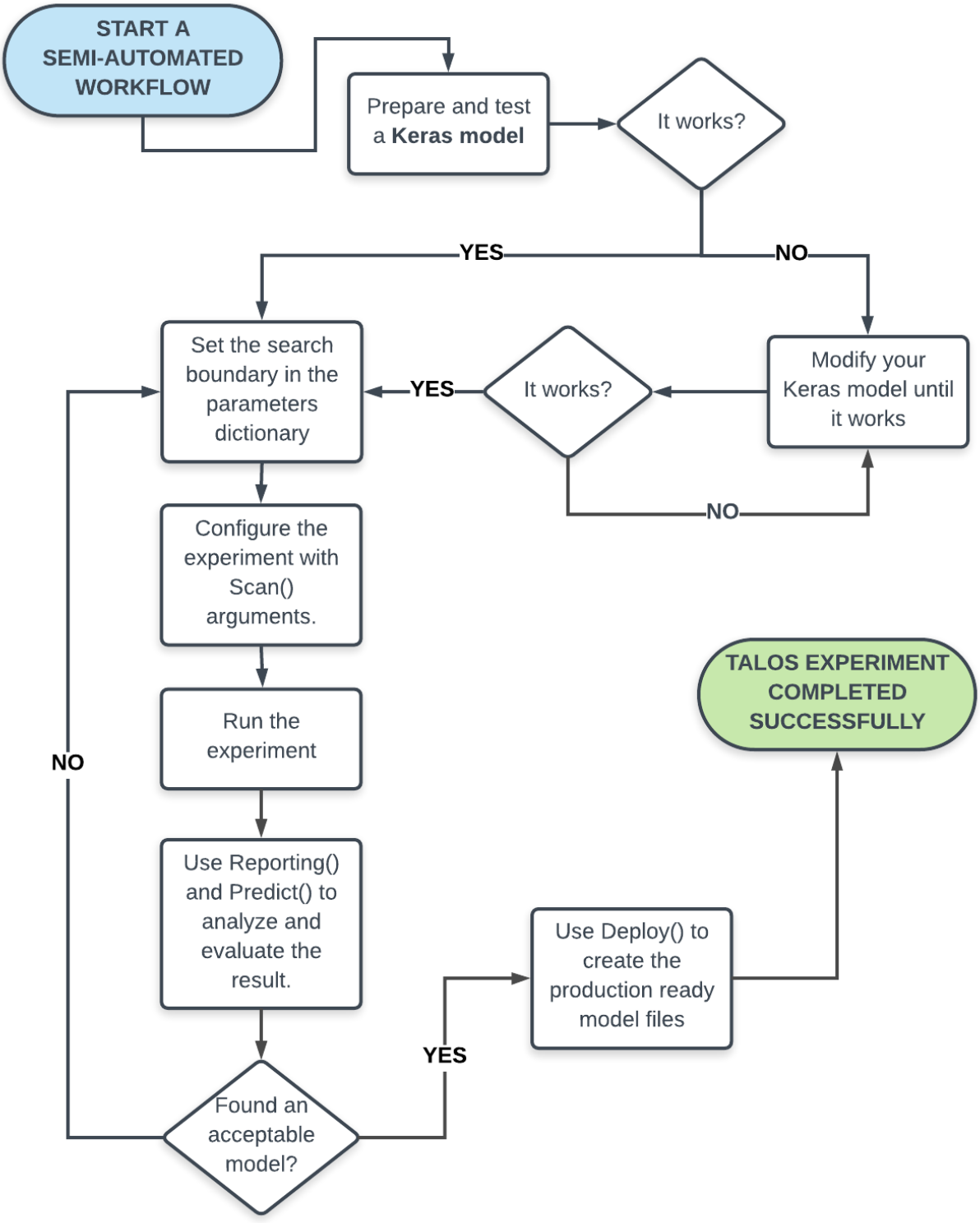

Figure 4.6: Talos Hyperparameter tuning workflow [17] (MIT License)

4.4.4 Backtesting

To perform backtesting we used BackTrader [2]. BackTrader is an open source framework that allows us to focus on writing reusable trading strategies and test our trading strategy. Parameters such as commission, interest rate, etc. can be configured through the program. We chose a simple trading strategy that goes long on prices (explained in Section 2.1.1). When the MACD Histogram moves from a negative value toward a positive value we go long on the stock. We also consider a threshold of 5% for crossing the zero-line, this will account for mistakes. If the portfolio's value is higher than the initial amount and the buy and hold strategy, we can say the approach was successful.

Chapter 5

Results and Discussion

In this chapter we present the results of the experiments. The result are visualized and compared with each other.

5.1 Calculating Returns

The focus of the experiment, as explained in Section 4.4.2, was on predicting the MACD histogram and using that along with a trading strategy. When the MACD histogram crosses above zero we buy and sell when it crosses below zero. The mean squared error, as shown in Figure 5.1 for our experiment without using technical indicators was 0.04057. In Figure 5.1, the first date of the test data starts with index 0, which is equivalent to 2018/9/10 and the final day is 2019/03/12. The results of actual and predicted data with scaled values are compared in Figure 5.2.

To calculate how much this model yields profit, it is necessary to identify days that cross above or below zero to generate buy and sell signals. Based on the data visualized in Figure 5.3, these days would be [6 11 23 26 40 43 44 58 72 75 125]. Since we are only considering to go long the buy signals are [11 26 43 58 75] and the sell signals are [23 40 44 72 125]. To remove sudden fluctuations in our data, we add a threshold of 5% for the zero-line in the MACD histogram. Therefore the prices have to cross the above or below the threshold line. As a result, we trade on certain days; these days are the buy and sell signals that will be fed into Backtrader as input data.

Backtrader has a list of configuration that can be set. For our testing purpose we are only interested in setting the commission value since it may have the highest impact on the final result. In the first experiment, the commission is set to zero and later the final value is calculated after optimizing the model with a commission to a flat rate of $10, which is a typical price offered for online trading. After feeding

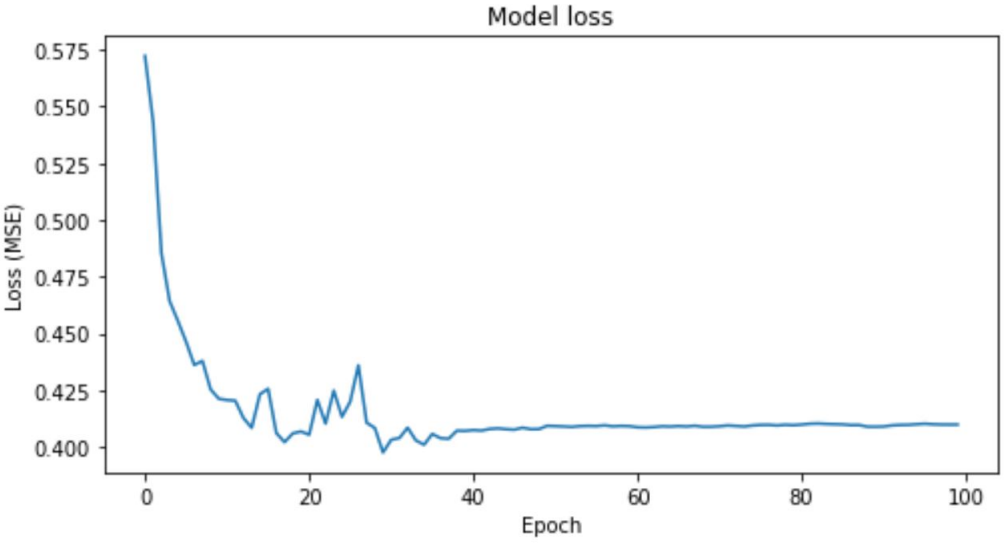

Figure 5.1: Validation loss for 100 epochs

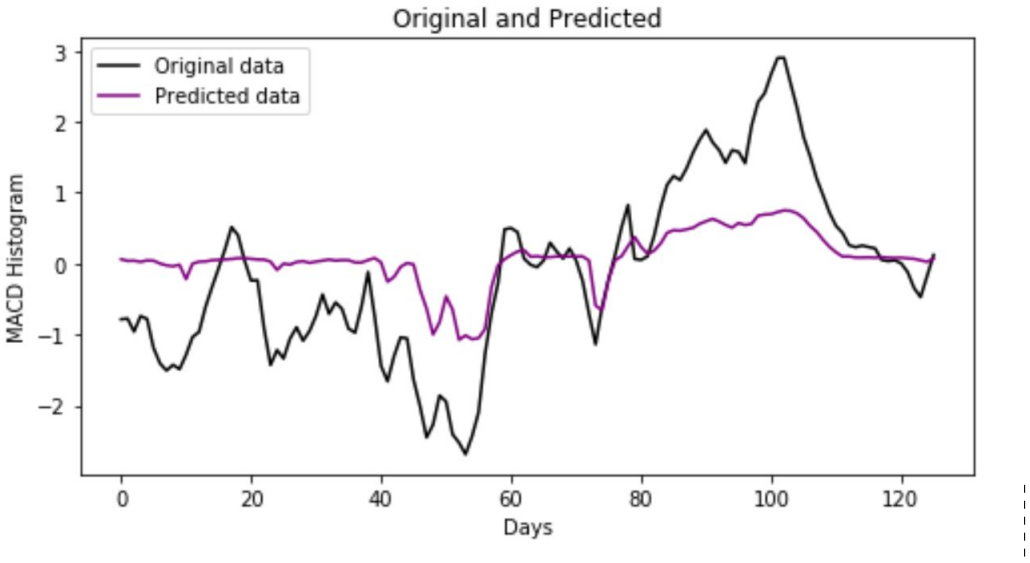

Figure 5.2: Original and predicted data for MACD Histogram.

the input data to Backtrader, the program starts simulating the trading. The log of trades in shown in Figure 5.4, and we can see the cost of each trade. The size for each trade is set to 500. The cost of the investment will be equal to the size of trade multiplied by the value of the equity on the first purchase. The return on investment (ROI) is calculated by the following formula:

$$ROI = \Big(\frac{Net\ profit}{Cost\ of\ investment} * 100\Big)$$

The results of backtesting are shown in Figure 5.5. Although the final portfolio shows a small amount of loss, when compared to the price of stock, the values indicate that the results can be optimized to create a profitable model.

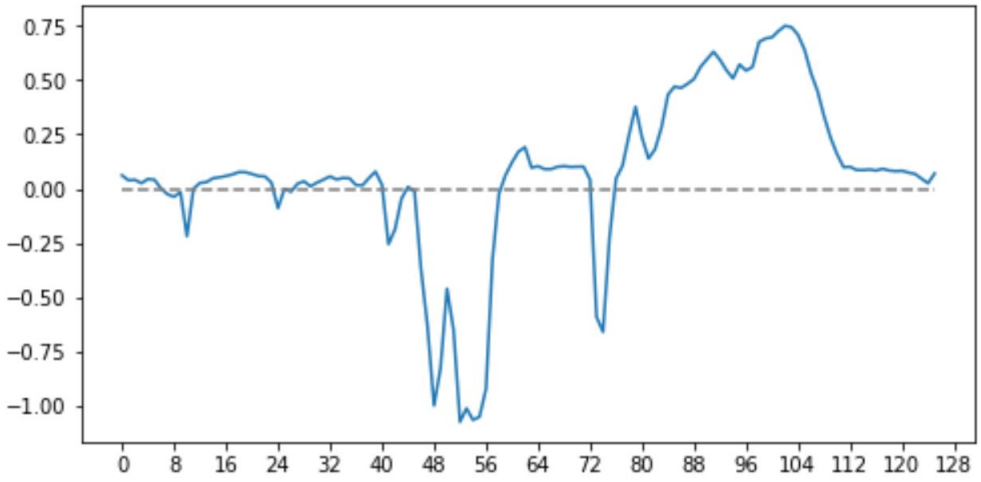

Figure 5.3: Predicted MACD Histogram with zero line

```
Starting Portfolio Value: 1000000.00
2018-09-25, BUY CREATE, 219.63 - Index 11.00
2018-09-26, BUY EXECUTED, Price: 218.45, Cost: 109225.00, Comm 10.00
2018-10-11, SELL CREATE, 211.98 - Index 23.00
2018-10-12, SELL EXECUTED, Price: 217.88, Cost: 109225.00, Comm 10.00
2018-10-12, OPERATION PROFIT, GROSS -285.00, NET -305.00
2018-10-16, BUY CREATE, 219.59 - Index 26.00
2018-10-17, BUY EXECUTED, Price: 219.73, Cost: 109865.00, Comm 10.00
2018-11-05, SELL CREATE, 199.26 - Index 40.00
2018-11-06, SELL EXECUTED, Price: 199.59, Cost: 109865.00, Comm 10.00
2018-11-06, OPERATION PROFIT, GROSS -10070.00, NET -10090.00
2018-11-08, BUY CREATE, 206.80 - Index 43.00
2018-11-09, BUY EXECUTED, Price: 203.89, Cost: 101945.00, Comm 10.00
2018-11-09, SELL CREATE, 202.82 - Index 44.00
2018-11-12, SELL EXECUTED, Price: 197.39, Cost: 101945.00, Comm 10.00
2018-11-12, OPERATION PROFIT, GROSS -3250.00, NET -3270.00
2018-11-30, BUY CREATE, 177.14 - Index 58.00
2018-12-03, BUY EXECUTED, Price: 182.97, Cost: 91485.00, Comm 10.00
2018-12-21, SELL CREATE, 149.51 - Index 72.00
2018-12-24, SELL EXECUTED, Price: 146.95, Cost: 91485.00, Comm 10.00
2018-12-24, OPERATION PROFIT, GROSS -18010.00, NET -18030.00
2018-12-27, BUY CREATE, 154.89 - Index 75.00
2018-12-28, BUY EXECUTED, Price: 156.23, Cost: 78115.00, Comm 10.00
2019-03-12, SELL CREATE, 180.22 - Index 125.00
Final Portfolio Value: 980290.00
ROI:
-18.045319295033188
```

Figure 5.4: Log of trades using Backtrader.

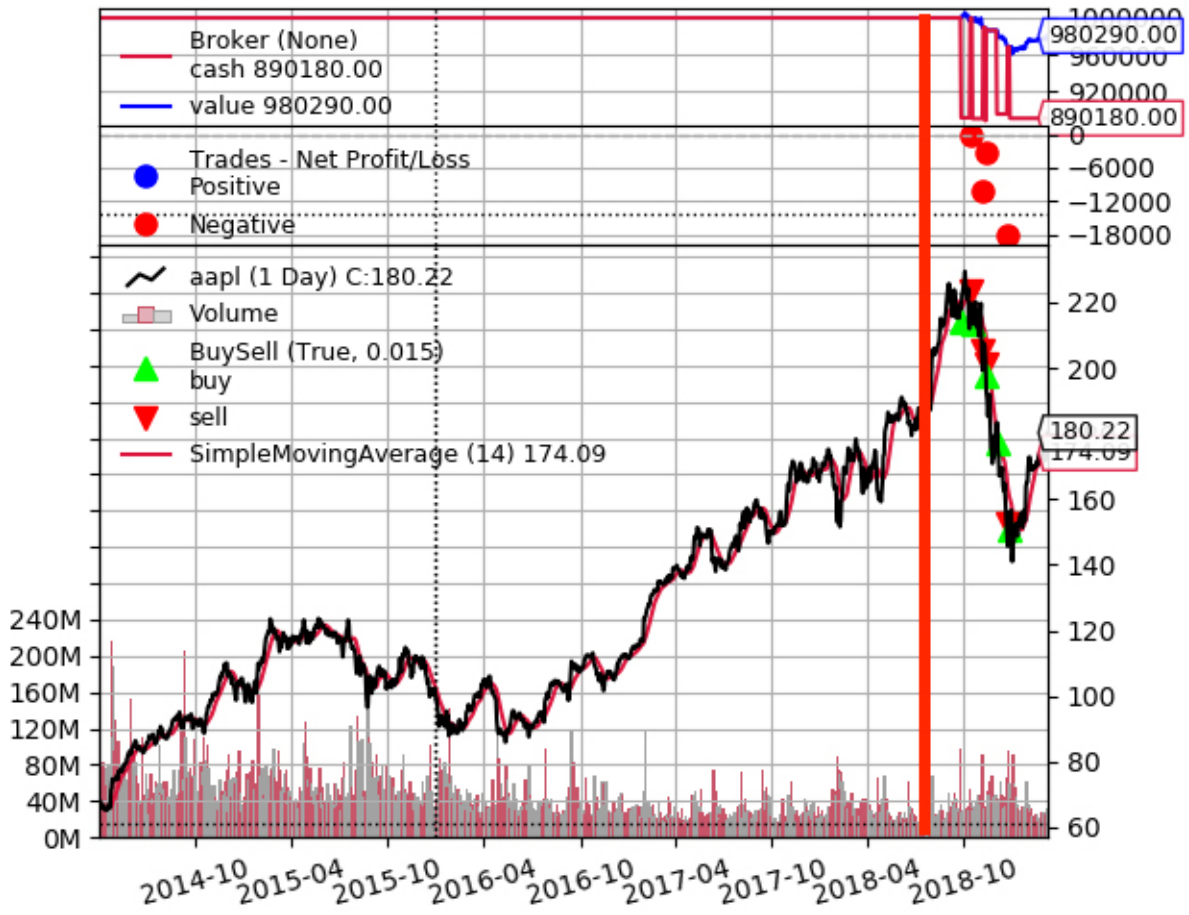

Figure 5.5: Backtesting using Backtrader. Buy and sell signals are generated for the test period (after red line).

5.2 The Effect of Technical Indicators

In this experiment, the goal is to examine the use of technical indicators on the performance of our model. We will examine the use of three main technical indicator: RSI, William %R and Volatility. In Table 5.3, we see the effect of these indicators on the loss value. As we can see, by using only RSI we can decrease the loss of the model to the lowest amount.

Technical Indicator	MSE
Volatility	0.02699
William %R	0.03845
RSI	0.02053
All Indicators	0.02598
No Indicator	0.04057

Table 5.1: The effect of technical indicators.

5.3 Hyperparameter Optimization Results

To further improve the results and also to examine which hyperparameters have better effects on the model for future research, hyperparameter optimization is performed. As shown in Figure 5.6, after running the experiment for 60 different combinations based on the hyperparameter space, Mean Squared Error fluctuates between best to worse. The the set of parameters with the lowest MSE is the best result. The results of the experiment will be analyzed to find the best hyperparameters.

5.3.1 Analysis of results

By looking at Figure 5.8, it can be seen that by using LeakyReLU the results improve and MSE comes below 0.1 while by using the ReLU function MSE goes well above 0.2. To summarize the results, the top ten best results are selected and displayed in Table 5.2. The best parameter space is the first row with the MSE of 0.00484. The best set of parameters is shown in Table 5.3. The model is evaluated with these parameters and the results will be calculated to examine the performance. And the parameters of the best result are shown in Table 5.3. The optimization functions are

compared in Figure 5.8. The results of activation and optimization functions can be seen side by side in Figure 5.9.

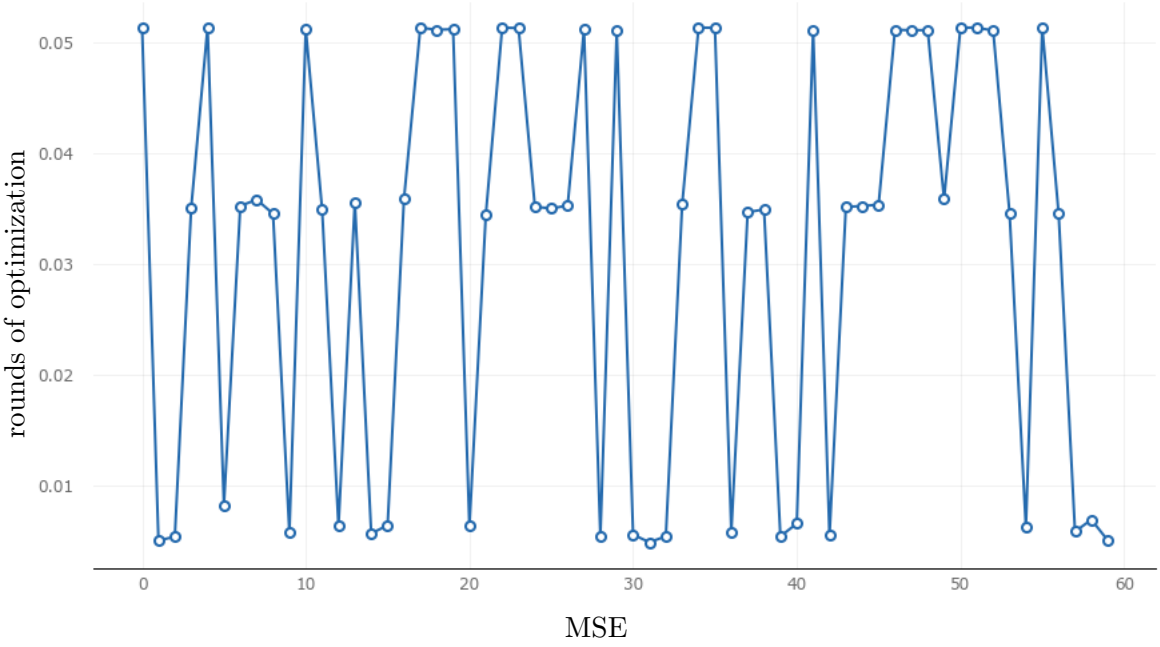

Figure 5.6: Change of MSE during the hyperparameter optimization.

MSE	Dropout	Optimizer	Loss function	Activation
0.004845492	0.1	Nadam	MSE	LeakyReLU
0.004993705	0.1	Nadam	Huber loss	LeakyReLU
0.005018013	0.18	Nadam	MSE	LeakyReLU
0.005358382	0.1	Adam	Huber loss	LeakyReLU
0.005375063	0.18	Nadam	MSE	LeakyReLU
0.005381555	0.1	Ndam	Huber loss	LeakyReLU
0.005391314	0.26	Adam	Huber loss	LeakyReLU
0.005471755	0.18	Aadam	MSE	LeakyReLU
0.005537316	0.26	Nadam	MSE	LeakyReLU
0.00564124	0.34	Nadam	MSE	LeakyReLU

Table 5.2: Ten best results of hyperparameter optimization

Parameter	Value
Loss Function	MSE
Activation Function	LeakyReLU
Optimization Function	Nadam
Dropout	0.1

Table 5.3: Best hyperparameters found after optimization

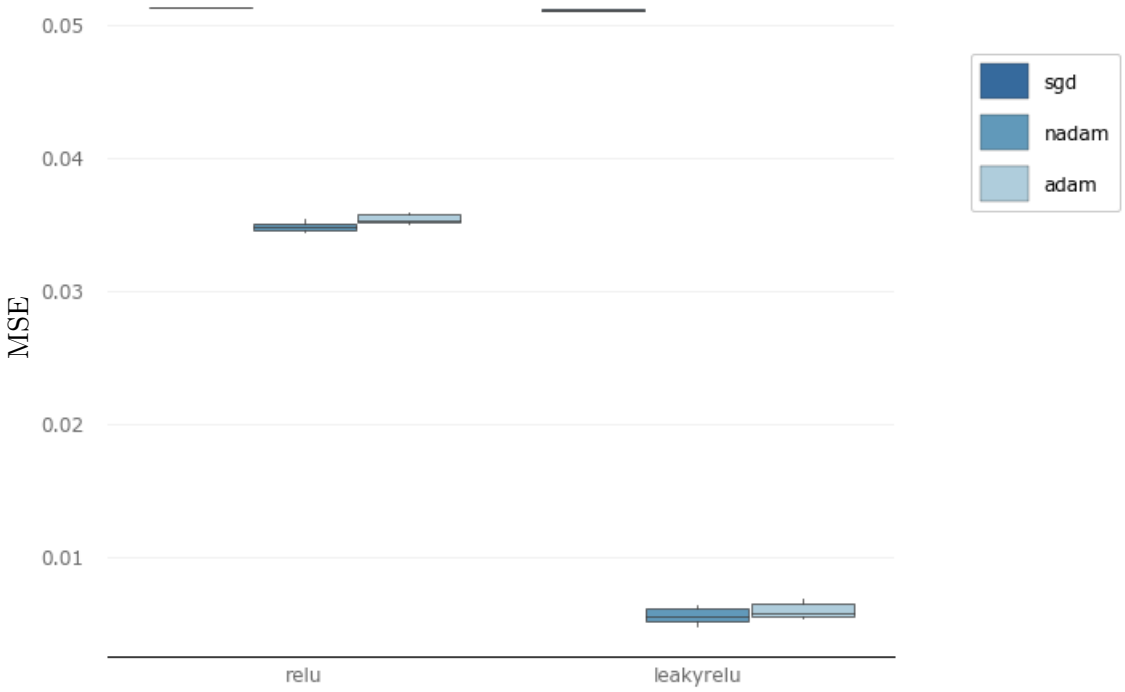

Figure 5.7: The effect of activation functions on the MSE during hyperparameter optimization

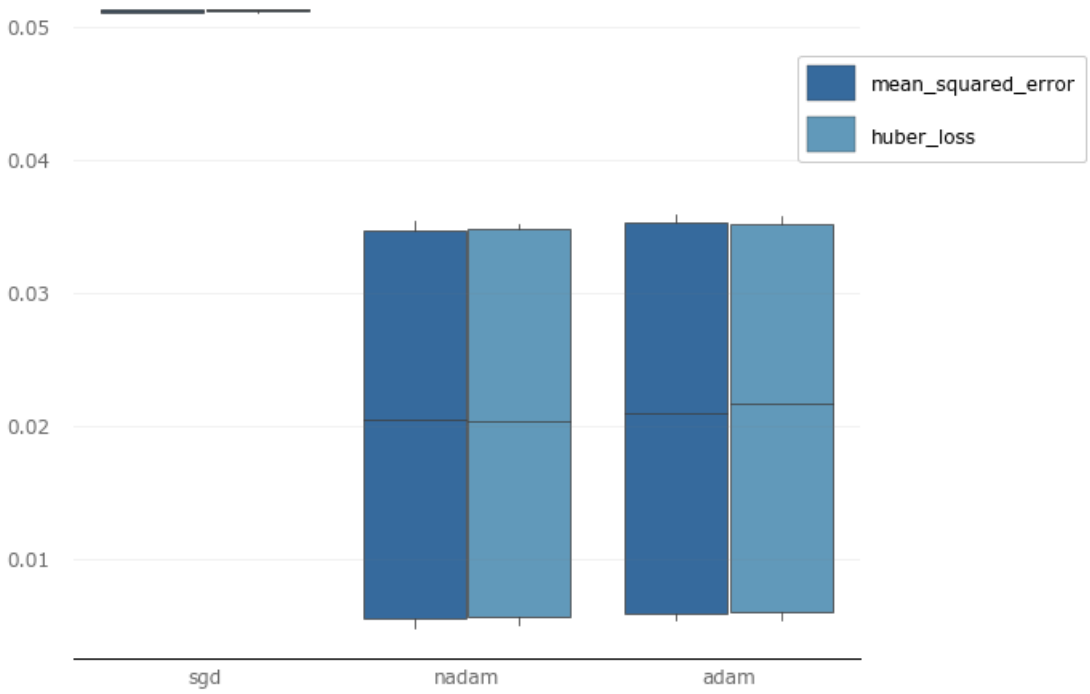

Figure 5.8: The effect of different optimizers on the Mean Squared Error(MSE) during hyperparameter optimization

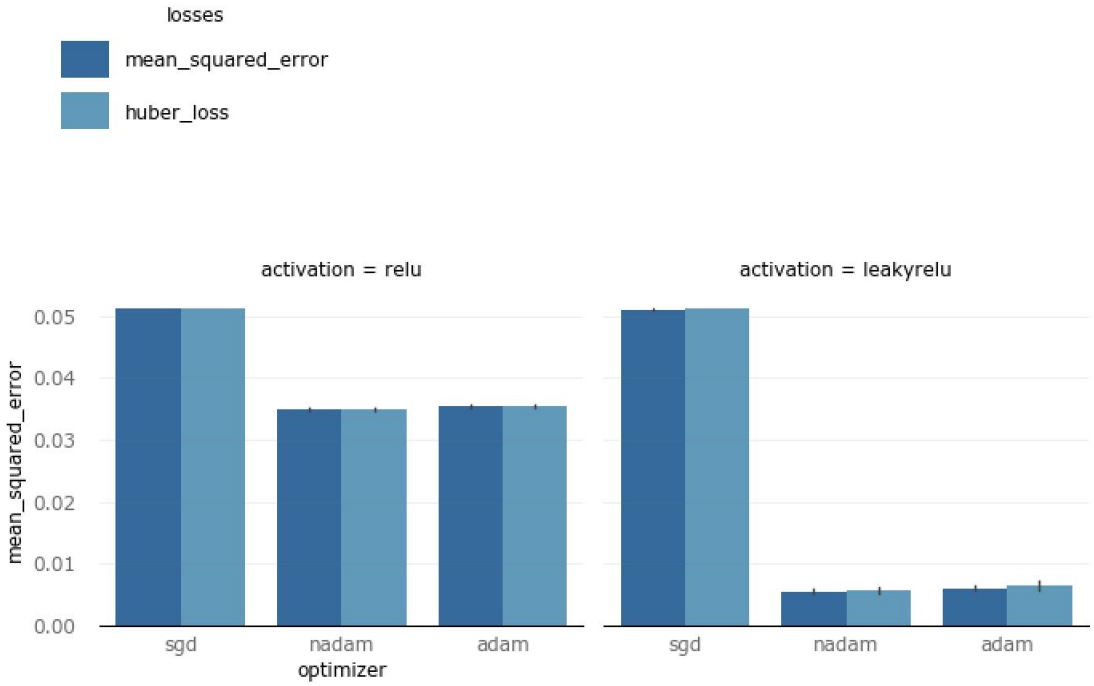

Figure 5.9: The effect of loss function and optimizer on the performance of the model during hyperparameter optimization.

5.4 Final Experiments

After selecting the best parameters from optimizing the hyperparameters and modifying our model, we run the experiments again to see how it increases the value of our portfolio. In Figure 5.10, we see the loss of the model has decreased by 0.54128436. Figure A.9 shows the actual and predicted data.

The days that crossed the zero line are [77 78 82 111] Analyzing the predicted results to generate buy and sell signals we determine [77 82] for buying and [78 111] for selling. Our final portfolio value increased to $1015075.00, which shows that optimizing the hyperparameters has a positive impact for this single testing set. Based on the beginning purchase and the end value, the return on investment for 128 days is equal to 6.1%. It should also be noted that the ROI with the buy and hold strategy for the same period of time is equal to -16.37%.

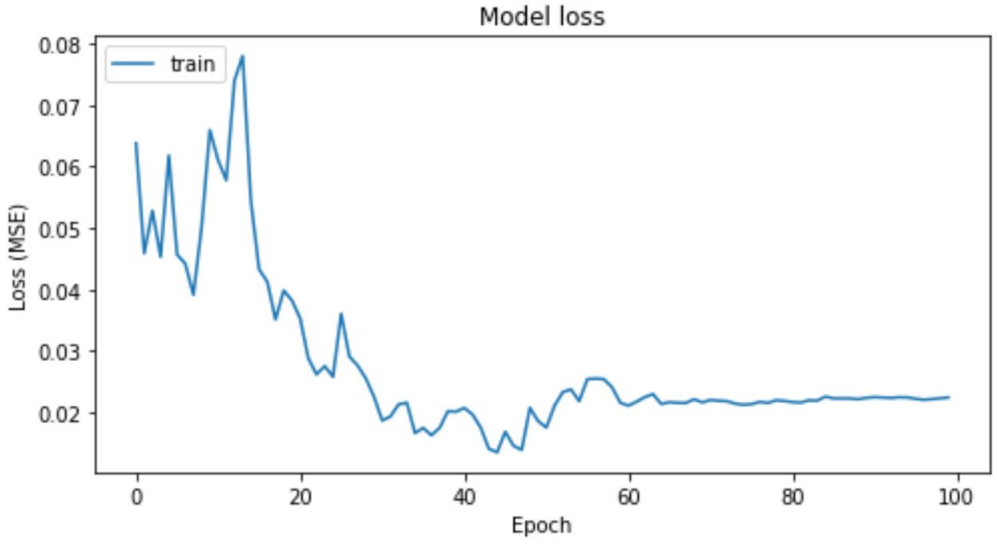

Figure 5.10: Loss of the model after hyperparameter optimization

5.4.1 Comparing Results

To summarize the results and compare the results of hyperparameter optimization, the results are shown in Table 5.4.

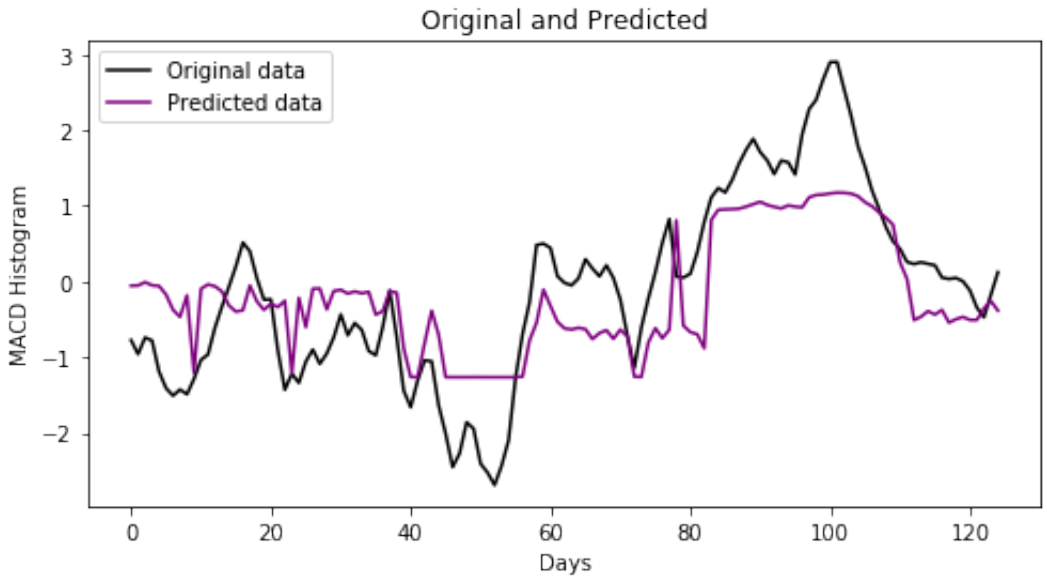

Figure 5.11: Original and Predicted data

Expriment	MSE	ROI
Before Optimization	0.040576865	-16.37%
After Optimization	0.004845492	6.67%

Table 5.4: Comparing the results of conducted experiments.

5.5 Testing Other Stocks

To test our approach we select three more stocks from the information technology sector of S&P 500. We use the same process and select the hyperparameters from our previous experiment. The final results are shown in Table 5.5. The beginning balance is the amount of money spent to buy 500 shares. We also redid the last experiment with the commission included.

Company Name	Symbol	Commission	ROI	Buy & Hold
Apple	AAPL	$10	6.61%	-16.37%
Microsoft	MSFT	$10	25.99%	27.51%
Google	GOOG	$10	5.769%	9.22%
Intel Corp	INTC	$10	16.9%	2.34%

Table 5.5: Comparing ROI for different stocks.

Chapter 6

Conclusion and Future Work

In this thesis, we used long-short term memory networks to predict the trend of markets and generate buy and sell signals to create a profitable model. We started our research with the initial goal to predict the price of a stock at a certain point in time. Although we were able to reduce the loss of the model, it was seen that during backtesting the portfolio decreased in value. This issue indicates the importance of backtesting and changing the metrics of the model to measure profitability instead of predicting the price. Later we modified our approach, but again, we were confronted by failure. We next changed the approach of the model to predict technical indicators and apply a trading strategy based on the selected technical indicator. One approach that showed promising results was predicting the MACD histogram, which is a suitable indicator to predict the trend of the market.

We chose Apple's stock and created an LSTM network to test our approach and later conducted the experiment for other stocks chosen from S&P 500 . The network's architecture was initially improved by studying previous research and Backtesting was setup to evaluate the model's outcome in the real world.

With our research we addressed a few topics: 1) It was seen that it was not possible to use LSTM networks to predict the prices of a stock in order to create a profitable portfolio. 2) We examined the effect of RSI, William %R and Volatility on the loss of the model. It was shown that by using only RSI, the model's loss was reduced, which contributed to the performance of the model. 3) We measured the effect of hyperparameter optimization with the use of Grid Search and testing 61 combination to identify the best set of hyperparameters. It was shown that choosing a different optimizer and activation function had a substantial effect on the loss of the model. By reducing the loss value it was possible to increase the return on investment from a negative value to 6.67%. This paves the way for future research with the focus of

identifying stocks with higher returns and creating portfolios with multiple stocks.

Our results show that deep learning can be integrated with technical analysis to create a profitable portfolio by choosing the correct technical indicator. It should also be noted that the results of the portfolio can be further optimized by choosing more complicated trading strategies.

The achieved results lay the ground for further research. Future research could examine the effect of different input data. The data can be gathered from other sectors and different stock markets. Other technical indicators can also be used to decrease the loss value. The research could be further expanded to modify the formulas for technical indicators and also test which indicators have a higher correlation with other factors such as volatility, etc. to categorize the prediction models. Different neural network architectures could also be used to achieve better results. Another topic would be to integrate the methodology with other analyses such as sentiment analysis [58]. With the help of sentiment analysis, it may be possible to generate buy signals sooner and integrate that to the built model. Other research direction is the use of fundamental indicators such as revenue. The combination of these methods along with the examination of the approach on a wider area of stocks can be a suitable choice for future research.

Bibliography

[1] A.i. wiki. https://skymind.ai/wiki/lstm. Last accessed 10 April 2019.

[2] Backtrader. https://www.backtrader.com/. Last accessed 26 June 2019.

[3] Convolutional neural networks for visual recognition. http://cs231n.github.io/neural-networks-1//. Last accessed 11 May 2019.

[4] Fundamental package for scientific computing with python. https://numpy.org/. Last accessed 1 September 2019.

[5] Getting started with the keras sequential model. https://keras.io/getting-started/sequential-model-guide/. Last accessed 6 April 2019.

[6] Huber loss function. https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#huber. Last accessed 08 May 2019.

[7] Keras documentation. https://keras.io/callbacks/. Last accessed 20 May 2019.

[8] Keras: The python deep learning library. https://keras.io. Last accessed 10 September 2018.

[9] Macd. https://www.investopedia.com/terms/m/macd.asp. Last accessed 6 April 2019.

[10] Macd historgram. https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:macd-histogram. Last accessed 6 April 2019.

[11] Macd historgram in technical analysis. https://commodity.com/technical-analysis/macd/. Last accessed 6 April 2019.

[12] Machine learning in python. https://github.com/scikit-learn/scikit-learn. Last accessed 1 September 2019.

[13] An open source machine learning framework. https://github.com/tensorflow/tensorflow. Last accessed 1 September 2019.

[14] Python data analysis library. https://pandas.pydata.org/. Last accessed 1 September 2019.

[15] Relative strength index - rsi. https://www.investopedia.com/terms/r/rsi.asp. Last accessed 10 September 2018.

[16] Talos documentation. https://autonomio.github.io/docs_talos/#introduction. Last accessed 25 May 2019.

[17] Talos workflow. https://github.com/autonomio/talos/wiki/Workflow. Last accessed 28 May 2019.

[18] Technical analysis: The basic assumptions. https://www.investopedia.com/university/technical/techanalysis1.asp. Last accessed 10 April 2019.

[19] Trading strategy. https://www.investopedia.com/terms/t/trading-strategy.asp. Last accessed 02 May 2019.

[20] Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Last accessed 10 April 2019.

[21] What does sample, batch, epoch mean? https://keras.io/getting-started/faq/#what-does-sample-batch-epoch-mean. Last accessed 6 April 2019.

[22] What is deep learning? https://machinelearningmastery.com/what-is-deep-learning/. Last accessed 10 September 2018.

[23] What is the difference between test and validation datasets? https://machinelearningmastery.com/difference-test-validation-datasets/. Last accessed 10 May 2019.

[24] Williams %r. https://www.investopedia.com/terms/w/williamsr.asp. Last accessed 10 September 2018.

[25] M. Abe and H. Nakayama. Deep learning for forecasting stock returns in the cross-section. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 273–284. Springer, 2018.

[26] W. Ahmed and M. Bahador. The accuracy of the lstm model for predicting the s&p 500 index and the difference between prediction and backtesting, 2018.

[27] C. Ahuja and L. Morency. Lattice recurrent unit: Improving convergence and statistical efficiency for sequence modeling. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[28] J. Alberg and Z. C. Lipton. Improving factor-based quantitative investing by forecasting company fundamentals. *stat*, 1050:13, 2017.

[29] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[30] E. Chong, C. Han, and F. C. Park. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Elsevier*, 2017.

[31] T. Chong and W. Ng. Technical analysis and the london stock exchange: testing the macd and rsi rules using the ft30. *Applied Economics Letters*, 15(14):1111–1114, 2008.

[32] T. Chong, W. Ng, and V. Liew. Revisiting the performance of macd and rsi oscillators. *Journal of risk and financial management*, 7(1):1–12, 2014.

[33] M. Claesen and B. De Moor. Hyperparameter search in machine learning. In *Proc. of the 11th Metaheuristics International Conference*, pages 1–5, 2015.

[34] Wikimedia Commons. File:macdpicwiki.gif — wikimedia commons, the free media repository, 2015. [Online; accessed 10-June-2019].

[35] Wikimedia Commons. File:perceptron moj.png — wikimedia commons, the free media repository, 2016. [Online; accessed 10-June-2019].

[36] Wikimedia Commons. File:error surface of a linear neuron with two input weights.png — wikimedia commons, the free media repository, 2017. [Online; accessed 15-May-2019].

[37] Wikimedia Commons. File:long short-term memory.svg — wikimedia commons, the free media repository, 2018. [Online; accessed 3-May-2019].

[38] T. Dozat. Incorporating nesterov momentum into adam. 2016.

[39] T. Fischer and C. Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, October 2018.

[40] T. Fletcher. *Machine learning for financial market prediction*. PhD thesis, University College London, 2010.

[41] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[42] J. B. Heaton, N. G. Polson, and J. H. Witte. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1):3–12, 2017.

[43] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[44] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[45] C. D. Kirkpatrick and J. R. Dahlquist. *Technical Analysis: The Complete Resource for Financial Market Technicians. Financial Times Press*. Wiley, 2006.

[46] M. Klassen. Investigation of some technical indexes in stock forecasting using neural networks. In *WEC (5)*, pages 75–79. Citeseer, 2005.

[47] J. Lucas, S. Sun, R. Zemel, and R. Grosse. Aggregated momentum: Stability through passive damping. *CoRR*, abs/1804.00325, 2018.

[48] B. G. Malkiel. Efficient market hypothesis. In *Finance*, pages 127–134. Springer, 1989.

[49] B. G. Malkiel. The efficient market hypothesis and its critics. *Journal of economic perspectives*, 17(1):59–82, 2003.

[50] V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[51] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. Adding gradient noise improves learning for very deep networks. *stat*, 1050:21, 2015.

[52] T. Rashid. *Make Your Own Neural Network*. CreateSpace Independent Publishing Platform, USA, 1st edition, 2016.

[53] N. Reimers and I. Gurevych. Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. *CoRR*, abs/1707.06799, 2017.

[54] O. B. Sezer, A. M. Ozbayoglu, and E. Dogdu. An artificial neural network-based stock trading system using technical analysis and big data framework. In *Proceedings of the SouthEast Conference*, pages 223–226. ACM, 2017.

[55] A. Tipirisetty. Stock price prediction using deep learning. page 60, 2018.

[56] H. Tobias, N. Navarro-Guerrero, S. Magg, and S. Wermter. Speeding up the hyperparameter optimization of deep convolutional neural networks. *International Journal of Computational Intelligence and Applications*, 17(02):1850008, 2018.

[57] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning*. 2019.

[58] Y. Zhao, B. Qin, T. Liu, et al. Sentiment analysis. *Journal of Software*, 21(8):1834–1848, 2010.

Appendix A

Results of other stocks.

In this Appendix we show the results of other stocks.

A.1 MSFT (Microsoft)

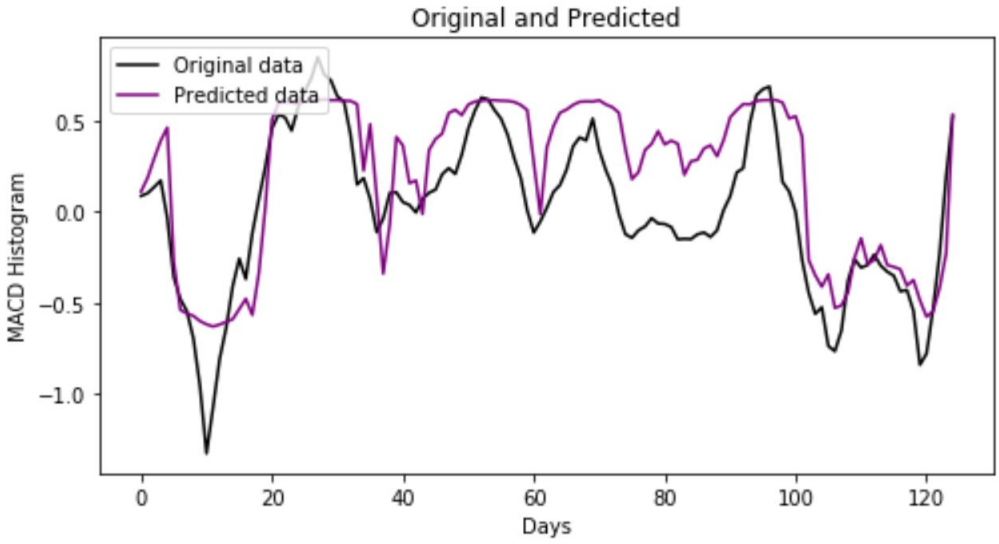

Figure A.1: Original and Predicted data

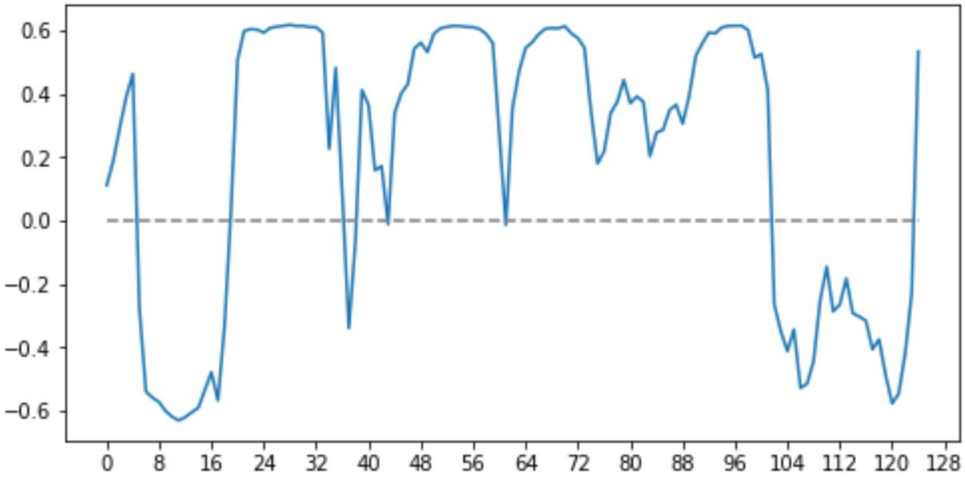

Figure A.2: Predicted MACD Histogram with cross line

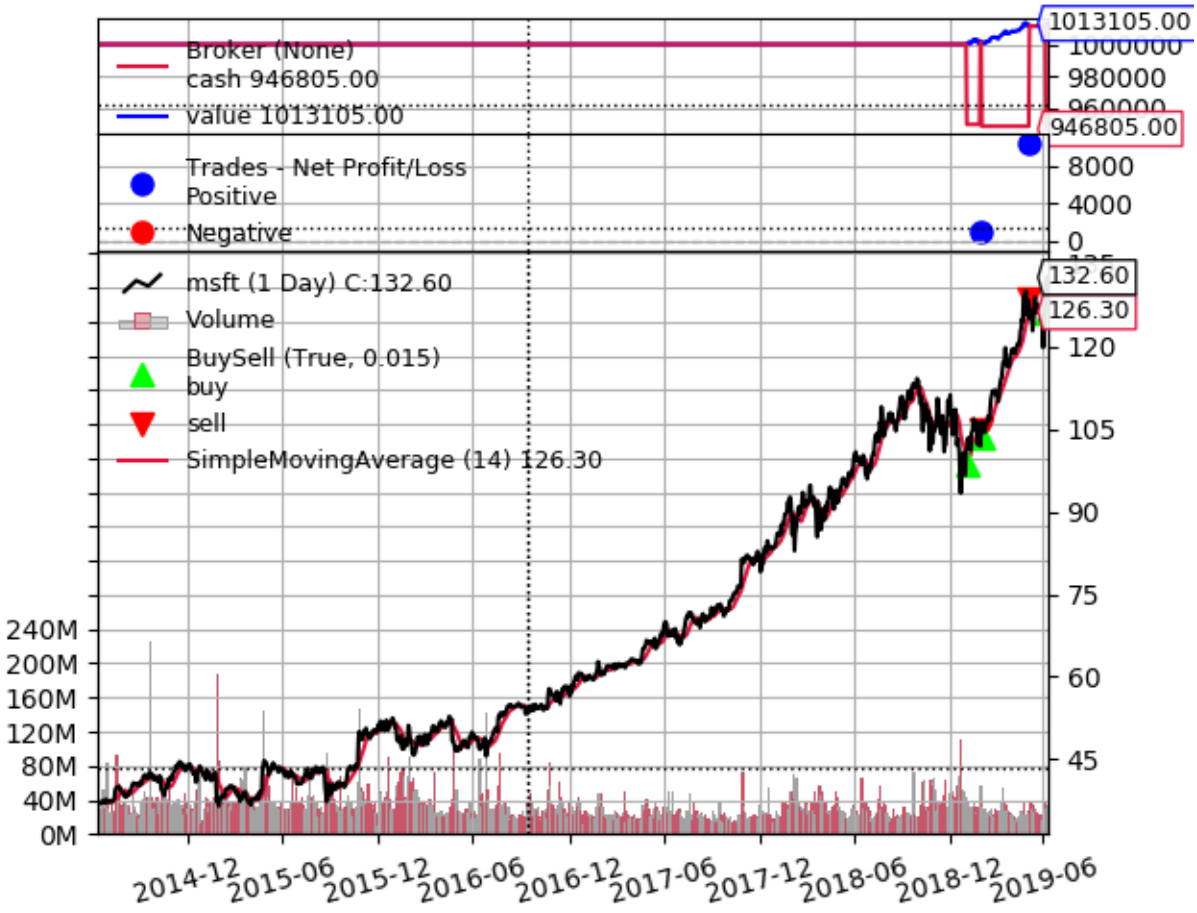

Figure A.3: Backtesting the data for test period.

A.2 GOOG (Google)

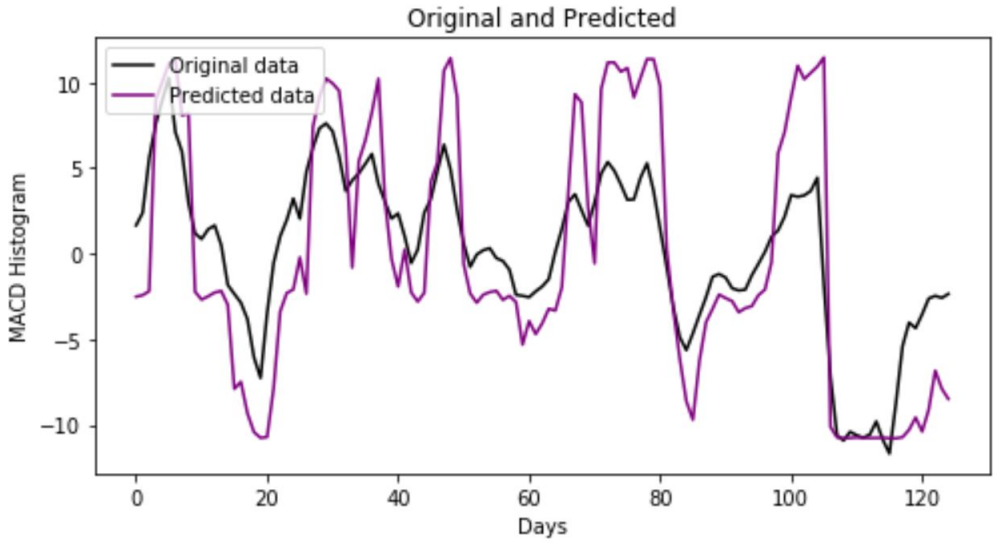

Figure A.4: Original and Predicted data

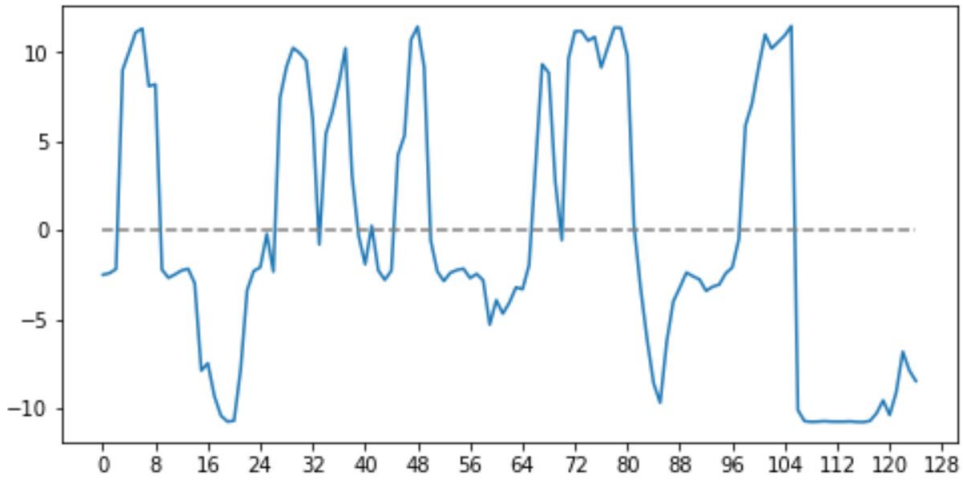

Figure A.5: Predicted MACD Histogram with cross line

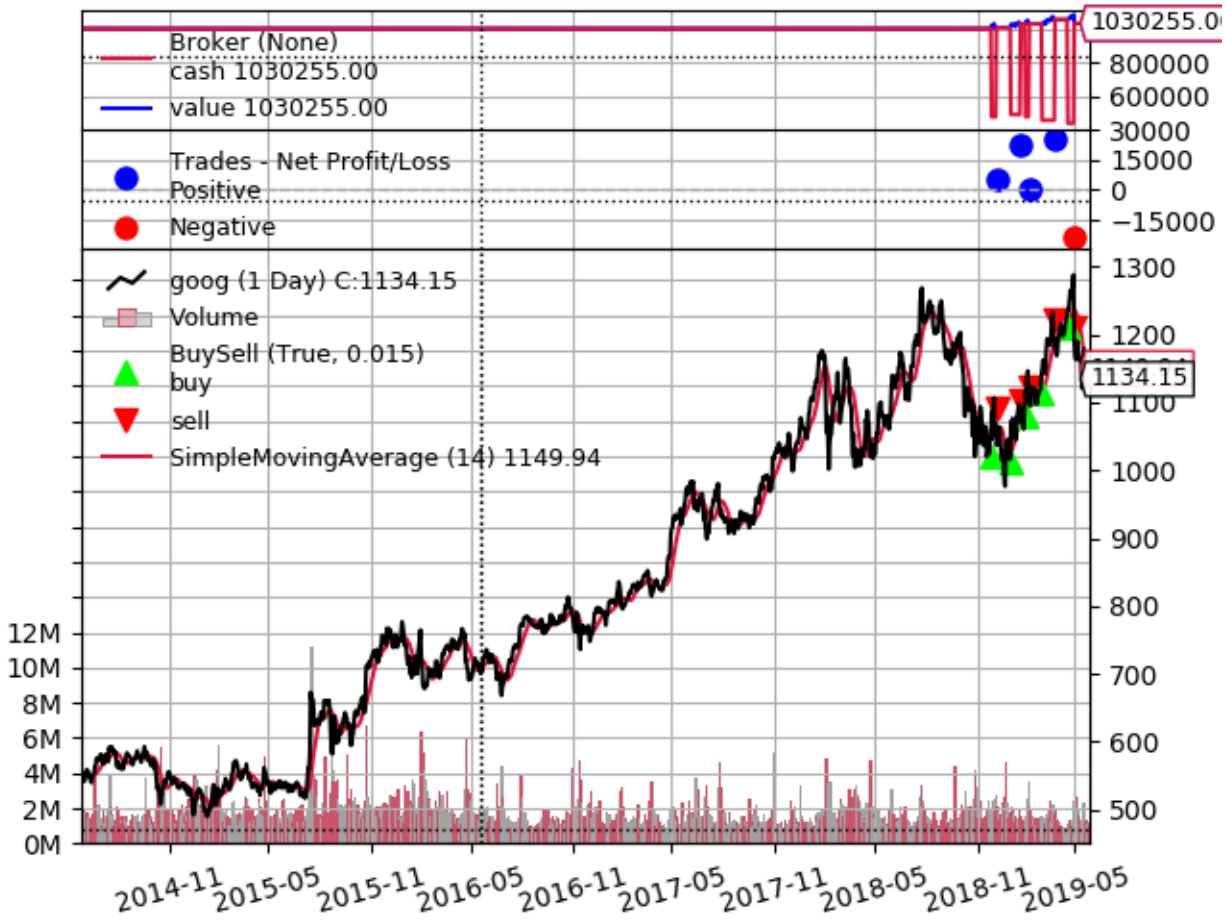

Figure A.6: Backtesting the data for test period.

A.3 INTC (Intel)

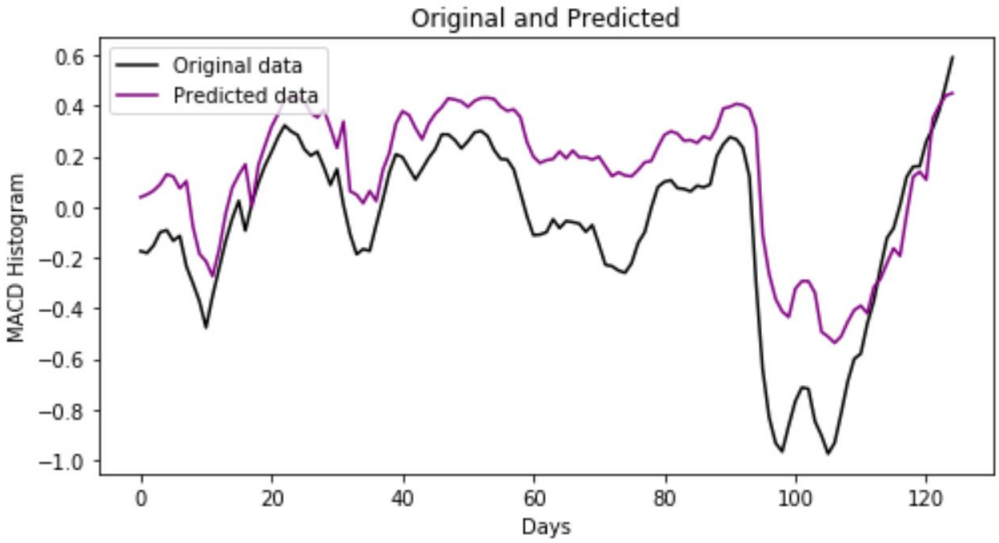

Figure A.7: Original and Predicted data

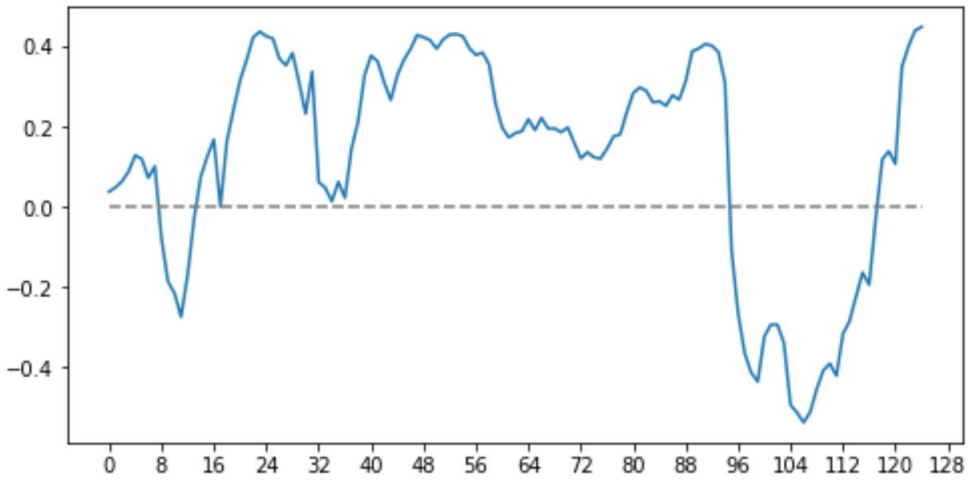

Figure A.8: Predicted MACD Histogram with cross line

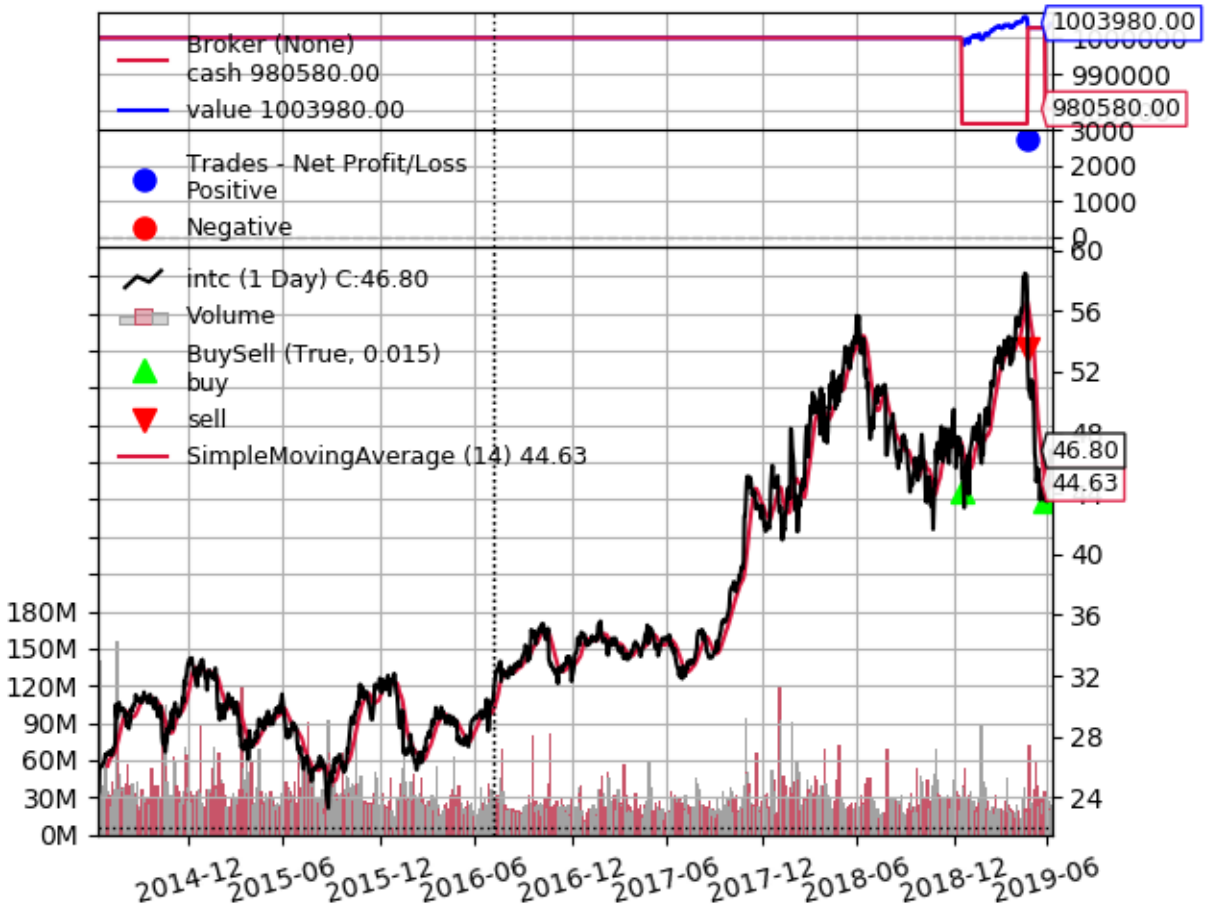

Figure A.9: Backtesting the data for test period.