LSTM Neural Network for Trading Cryptocurrencies

COMP4971C – Independent Work (Fall 2021)

Ng Yuk Pan

Supervised by Dr. David Rossiter, HKUST

Abstract

Neural networks are able of discovering and earning hidden correlations in data, in a way similar to that of the human brain. By using past and current data, it can predict the future trend and values of a time series, such as cryptocurrency prices. In this project, we aim to utilize LSTM neural network in machine learning to learn cryptocurrency trading. Starting by learning on a given dataset, the program should be able to remember part of the data, train, learn and evaluate its own actions, before moving onto the next loop. By learning patterns and changes in price of cryptocurrencies, the program makes predictions. They are combined with a 'buy low, sell high' strategy, which is validated in a mock trading environment. The main goal is to beat conventional trading strategies. In the BTC/USDT market, the program provides on average 15% more return than HODL strategy, with a maximum of 65.5% drawdown.

Table of Contents

Table of Figures

Introduction

With the emergence of cryptocurrencies, many people try to engage in the trading market in order to earn a profit. Due to its inherent volatility and other characteristics, it is difficult for people to accurately trade and predict the price change of cryptocurrencies by learning its hidden patterns. Especially, for FOMO (Fear Of Missing Out) or HODL (Hold On for Dear Life) traders, when there is a sharp decline in price, general panic may cause traders to sell at a great loss. Moreover, prices change during night time, sometimes by great margins, which an average trader would miss.

In order to achieve optimal trading performance, implementation of a neural network can be used to learn the patterns and changes in price of cryptocurrencies, thus calculating the buy / sell amount and decision. A neural network only requires computational power numbers, thus it will not be affected by conventional obstacles such as feelings and sleep.

In this project, we aim to utilize neural networks in machine learning to learn cryptocurrency trading. Starting by learning on a given dataset, the program should be able to train, learn and evaluate its own actions, before moving onto the next loop. The model will mainly be using a simple 'buy low, sell high' strategy. The main goal is to beat conventional strategies, which will be explained later.

Data

In terms of data selection, hourly data is selected as it has a higher frequency which means that the program can have more data to better capture the trends and patterns in the price data.

In terms of crypto-fiat pair, **BTC/USDT** is chosen due to the following reasons.

BTC, or Bitcoin, is one of the most widely known cryptocurrencies on the market. It alo has the highest market cap and the highest trading activity among cryptocurrencies.

USDT, or Tether, is a stablecoin, which refers to a coin backed by a real-world currency and mostly unaffected by the market of cryptocurrency. It also allows seamless, high-frequency trading within most exchanges, whereas conventional fiat may require long waiting times to accept transactions.

Obtaining the data

In order to obtain the historic data for BTC/USDT, hourly data from 17/08/2017 04:00 GMT to 01/09/2021 00:00 GMT is manually downloaded from https://data.binance.vision/ and combined into a single CSV file.

Opening Time	Open	High	Low	Close	Volume (BTC)	Closing Time
17/08/2017, 04:00 GMT	4,261.48	4,313.62	4,261.32	4,308.83	47.181009	17/08/2017, 05:00 GMT
17/08/2017, 05:00 GMT	4,308.83	4,328.69	4,291.37	4,315.32	23.234916	17/08/2017, 06:00 GMT
17/08/2017, 06:00 GMT	4,330.29	4,345.45	4,309.37	4,324.35	7.229691	17/08/2017, 07:00 GMT
17/08/2017, 07:00 GMT	4,316.62	4,349.99	4,287.41	4,349.99	4.443249	17/08/2017, 08:00 GMT
17/08/2017, 08:00 GMT	4,333.32	4,377.85	4,333.32	4,360.69	0.972807	17/08/2017, 09:00 GMT

Figure 1: The first 5 hours of BTC/USDT data

The program is then provided with the “Close” column of the data for training.

Plotting the data

The data provided to the model is as follows. The training set consists of 70% of all data (up to 16/06/2020 00:00 GMT), and the validation set consists of the last 30% of the data (16/06/2020 00:00 GMT onwards).

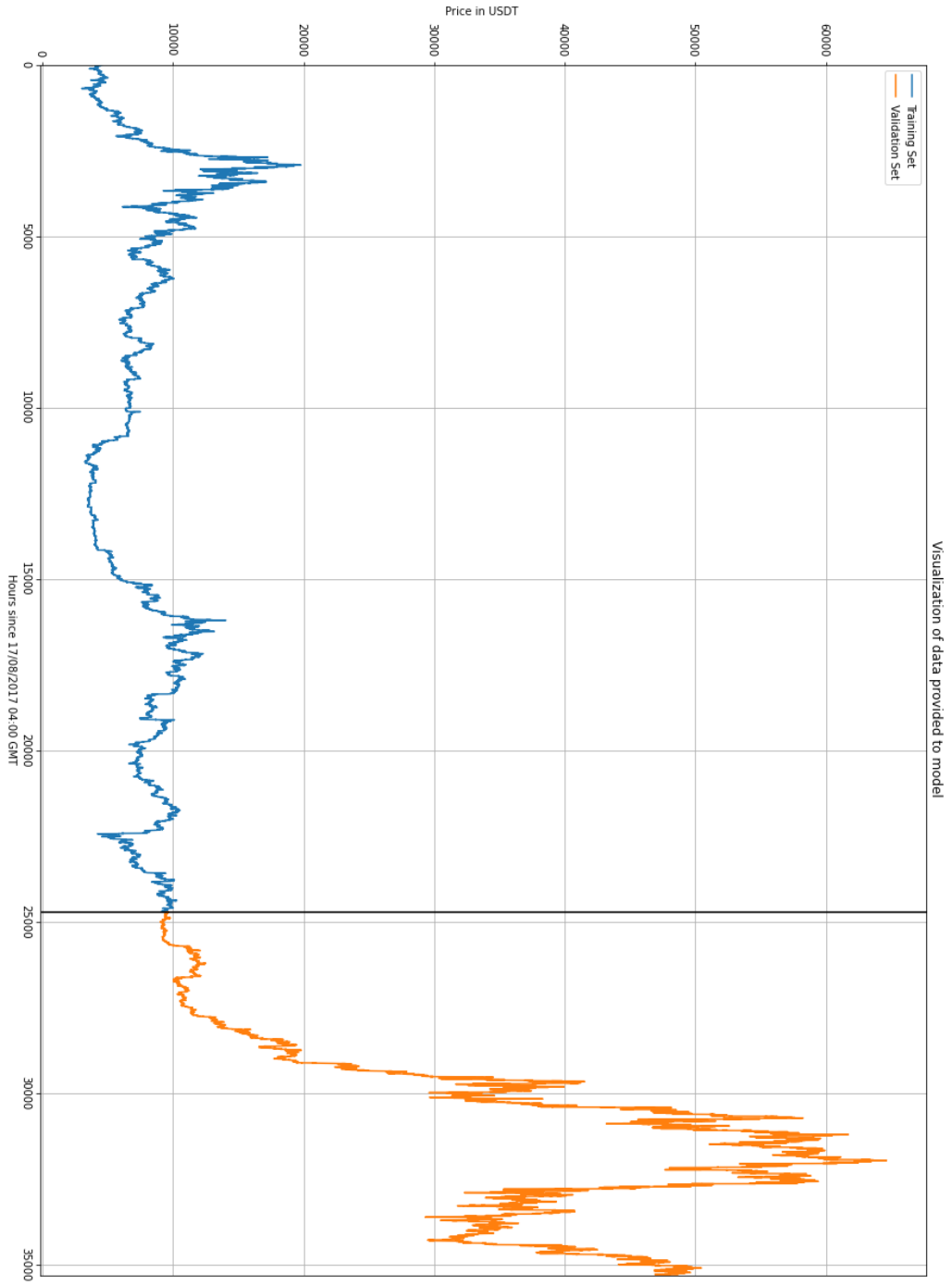

Figure 2: Visualization of data provided to the neural network model

Neural Networks

Conventional Neural Network (CNN)

A CNN uses the inputs given at a specific time to produce outputs. During training of a CNN, it has no way to access previous inputs, e.g. Given the data at time t, the CNN model cannot reference data from time $t-1$ to assist its predictions. Therefore, they are less able to learn the trends and time series hidden in data such as cryptocurrency prices. Therefore, CNNs will not be considered in the project.

Recurrent Neural Network (RNN)

A RNN differs from a CNN in that it has a 'hidden state' in addition to the inputs given, and it outputs a new 'hidden state' which contains information about the data in previous time steps. By passing the hidden state from $t-1$ as part of the input in time t, the RNN model can learn the trends and time series in the provided data better than in CNNs. Therefore, RNNs often consist of multiple layers to take advantage of the 'hidden state'.

The 'hidden state' and inputs are multiplied with their respective weights, then summed together, before passing into a tanh function to rectify all numbers into the range of 0 to 1.

The network performs pointwise operation according to the following equation of

$$h_t = \tanh(W_{ih}x_t + W_{hh}h_{t-1}),$$

where W_{ih} is the weight for the input and W_{hh} is the weight for the output.

A typical RNN cell can be visualized as follows:

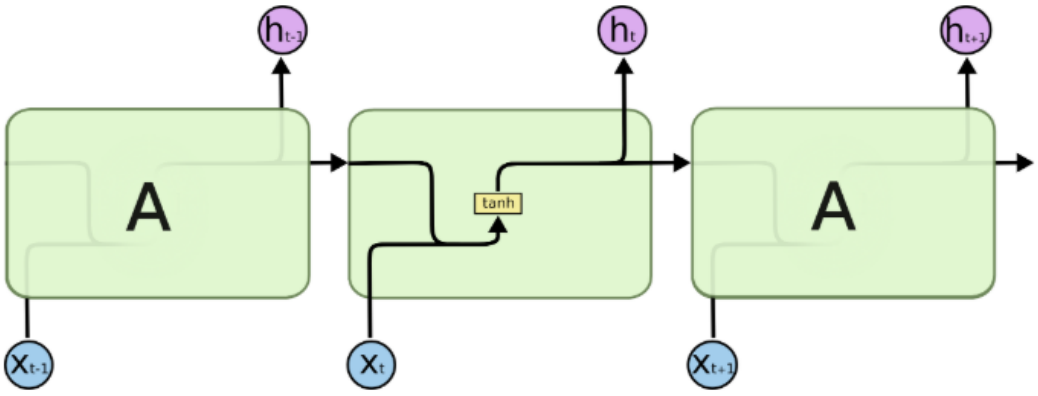

Figure 3: A visualization of RNN.

Where A is a single RNN cell, x is the input and h is the output.

Long Short Term Memory (LSTM)

The LSTM is a variation of RNN. The inputs of a LSTM consists of 3 different tensors: The input tensor of time (x_t), the hidden state from time $t-1$ (h_{t-1}), and the cell state from time $t-1$ (c_{t-1}).

There are 4 main operations in a LSTM cell, performed in order: (All operations are pointwise) [1]

1. Forget old information
2. Input new information
3. Update cell state
4. Output hidden state

Firstly, by using a sigmoid function on h_{t-1} and x_t with weights, the LSTM cell determines how much data is retained in the cell state. (1)

$$f_t = \sigma\left(W_{if}x_t + W_{hf}h_{t-1}\right)$$

The resulting tensor f_t has values between 0 and 1, which represents how much data should be forgotten from the previous cell state.

Then, using sigmoid and tanh functions on h_{t-1} and x_t with different weights, the LSTM cell determines the data added to the cell state. (2)

$$i_t = \sigma(W_{ii}x_t + W_{hi}h_{t-1}) \qquad g_t = \sigma\left(W_{ig}x_t + W_{hg}h_{t-1}\right)$$

The resulting i_t represents the magnitude in which new data influences the cell state, and the resulting g_t represents the values of the new data.

Thirdly, the cell state is updated according to f_t, i_t and g_t. (3)

$$c_t = (c_{t-1}f_t + i_t g_t)$$

The LSTM first forgets the old data as determined in f_t, then adds data according to i_t and g_t.

Finally, the output hidden state is produced by using a sigmoid function on h_{t-1} and x_t with weights, and multiplying the results with the tanh of the new cell state. (4)

$$o_t = \sigma(W_{io}x_t + W_{ho}h_{t-1}) \qquad h_t = o_t \tanh(c_t)$$

[1] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM cells can be visualized as follows:

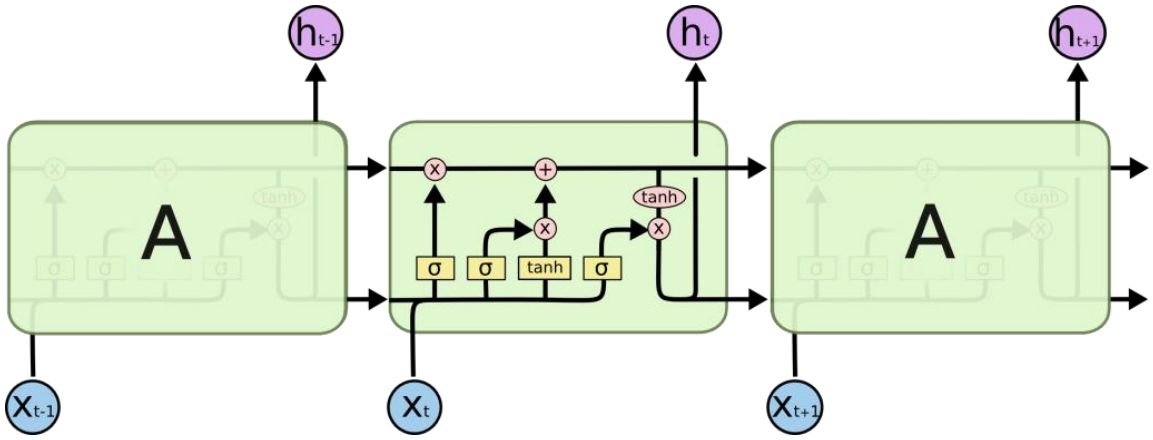

Figure 4: A visualization of LSTM.

Where A is a single LSTM cell, x is the input and h is the output.

The LSTM model has a benefit of being able to remember very long sequences by utilizing the cell state, while being able to adapt quickly by utilizing the current hidden and input tensors effectively as an output. Due to its ability to better store and 'remember' sequences from previous time steps, while utilizing recent data effectively, a LSTM is picked for the purposes of the project. PyTorch has an integrated library, torch.nn.LSTM[2], which is an easy solution to the implementation of LSTM models.

[2] https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Overview of software

The code is based on PyTorch, running Python 3 in Google Colab (cloud) GPU Runtime from https://colab.research.google.com/.

The GPU used in prediction is half of a NVIDIA Tesla K80.[3]

The model is trained on hourly historic spot candlestick data of BTC/USDT pair from Binance.

Model Structure

In this project, a model with the following structure is used:

The model consists of a 3-layer LSTM with a hidden size of 48, which is then connected to a linear layer which reduces the sequences from 4 points to 1.

The model uses a total of 12 batches with 96 normalized data points per batch, where the data is split into 24 sequences of 4 data points for the prediction process. A single number is predicted to be next in the sequence of 4 data points, and the loss is calculated and back-propagated as one training loop.

The model then uses a random sampling method on the validation data. A prediction is then made, and the loss is tracked to understand its performance and prevent over-fitting, where the model only remembers the data but not generalizing.

For evaluation of the performance of the LSTM model, a mean-square-error loss function is used. It allows the model to produce smaller errors, but punishes exponentially for large mistakes.

After the model is trained, it is locked into evaluation mode, to prevent frequent trading from influencing the decisions of the model.

The final model configuration was decided through trial and error.

Model	Input Size	Hidden Size	Output Size	Batch Size	Layers	Sequence Length	Notes
1	1	32	1	1	2	1	Error too large
2	1	64	1	12	2	1	Error too large
3	4	64	4	1	2	7	Predictions inaccurate
4	8	64	8	12	3	8	Model takes too long to run
Final	4	48	1	12	3	24	Best performing

[3] A NVIDIA K80 has 2 GPU cores. In Google Colab Free, only 1 GPU core is allocated per user.

Trading Simulation

Model Evaluation

During the trading phase, the model is fed chronological sections of data randomly. The data points are provided, the model makes a prediction, and the trading algorithm uses the difference between the normalized values of the current and previous time step to trade.

A positive number reflects the price will go up; a negative number reflects the price will go down. This 'trade ratio' is used to decide whether to buy or sell, as well as how much. The de-normalized predicted value is used as the 'trade price' that the program will try to buy / sell at.

After the 'trade price' and 'trade ratio' are produced, the model is (optionally) unloaded and deleted to prevent data leakage. A series of checks are performed to validate the trade. If the checks succeed, the trade will be operated according to the trade price and ratio; else, there will be no operations, which completes one loop.

The program will be compared to simple strategies, such as HODL (Hold On for Dear Life), which means the trader buys at the first possible moment and never sells no matter the price change, and a momentum strategy, which takes the difference between current and previous price, and applies it to the trading algorithm.

Testing Restrictions

The testing is done with the following restrictions:

1. The starting capital of each program will be 1,000 USD and no crypto.
2. A trade fee of 0.1% is deducted from each trade separately.
3. Any trade below 10 USD or 0.0001 BTC will be ignored.
4. Any trade where its trade price is outside the 'low' and 'high' price of the following hour will be ignored.
5. Any trade attempting to sell more than all owned assets will be corrected to 95% of said asset.

Results

The results are as follows: (Legends are provided in the graphs)

Total Value

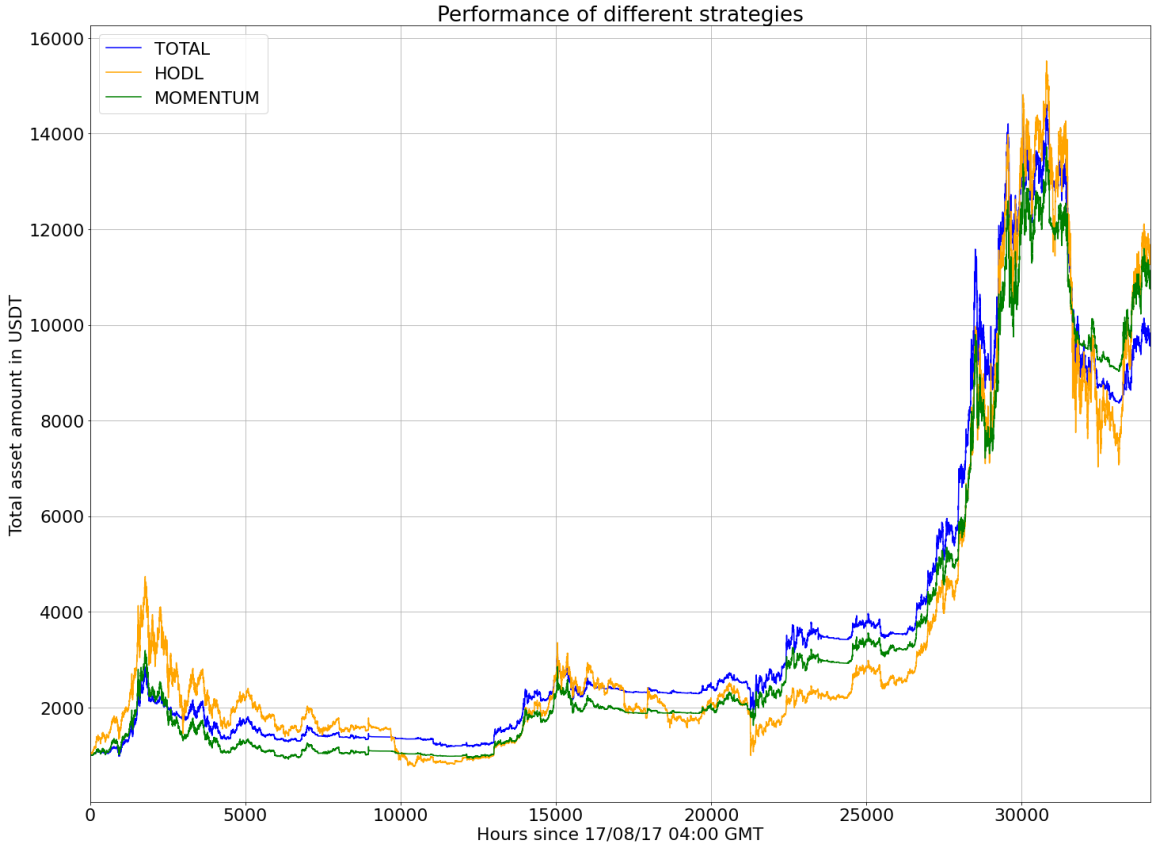

Figure 5: Initial results

From the graph, we can see clearly that in most cases, especially during fluctuations, the program and momentum strategy perform reasonably well.

When we take a closer look, we can see that during sections such as points 2,000-4,000 and 9,000-11,000, the program can avoid losing money by selling, while momentum trading reacts much slower and HODL drops significantly.

However, HODL can capture all the upward trends of the price fully, while momentum strategy and the model perform significantly worse during these times.

Overall, our program performs better most of the time, but during the great fluctuations near the end of the graphs, both trading options performed better than the program.

From the above, we can tell that our program is underperforming when compared with other strategies. However, due to the simplicity of such trading strategies, we can incorporate part of their strategies into the code to assist the prediction. The following graph contains an example of simply taking half of the prediction from the program and momentum strategy each:

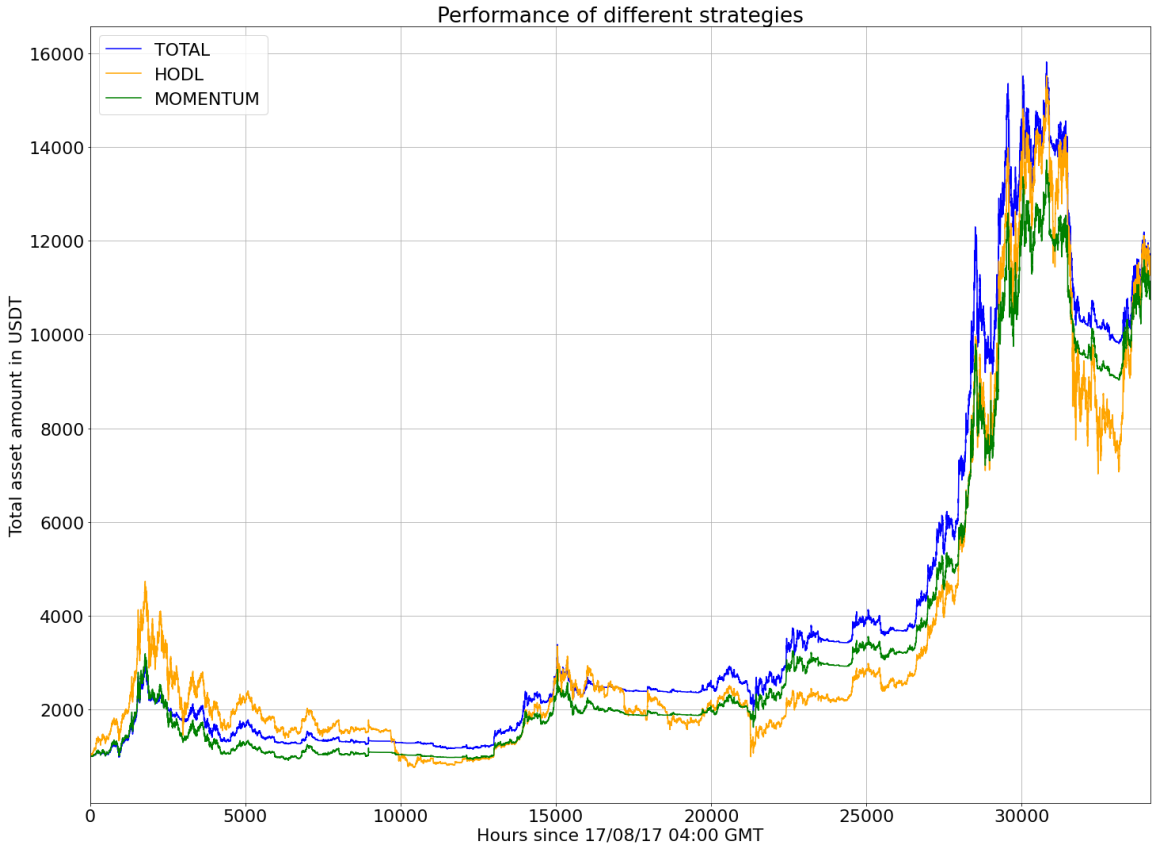

Figure 6: Results by combining program and momentum output

We can see that the hybrid approach outperforms other strategies, resulting in a slightly better result than HODL. After experimentation with the ratio between program and momentum strategy, the ratio was decided on 0.3:0.7. The performance of the modified program is as follows:

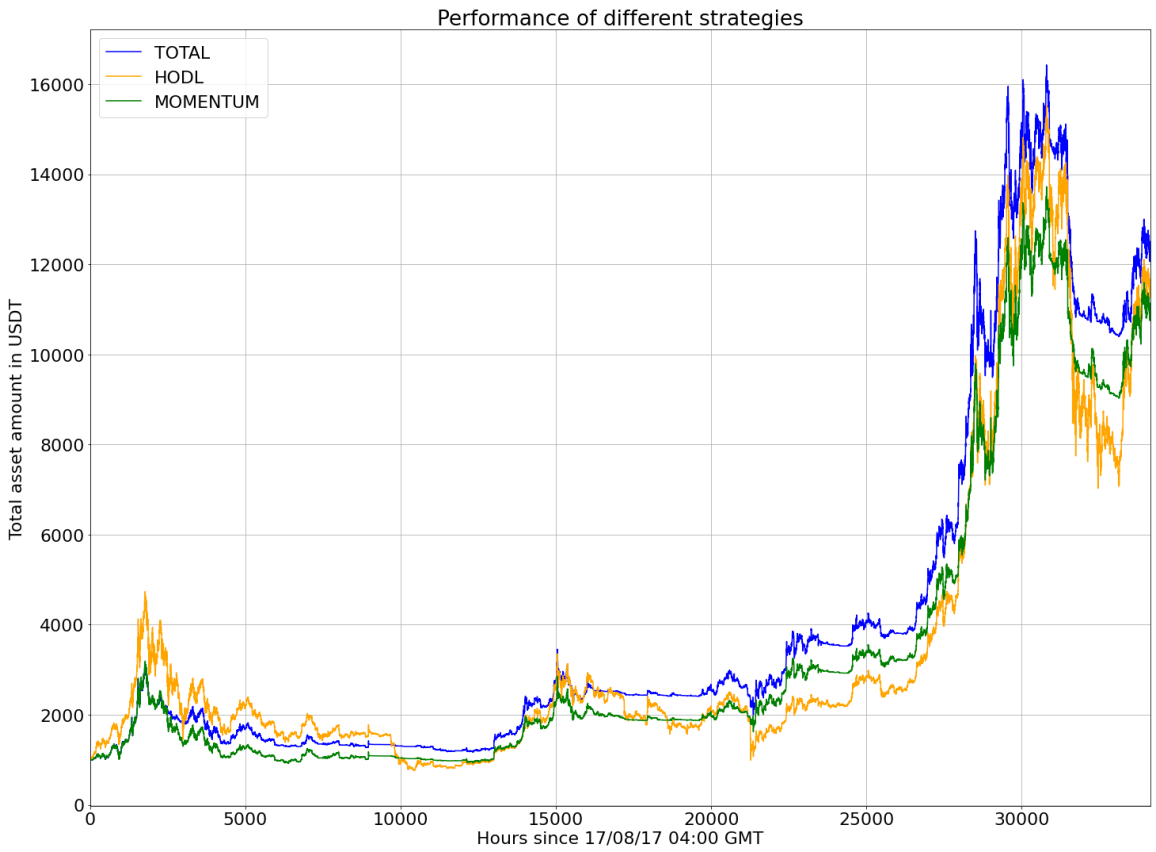

Figure 7: Results by optimizing combined output from above

When compared with the original graph, the main difference lies in the region from data points 25,000-30,000. The optimized version of the program can more accurately buy when compared with the un-optimized version, and thus produce better results. Where the original program did not have the best performance, the modified program is able to more accurately and confidently buy and sell orders.

In the final optimized version, the program ended with a total value of 12,110 USD while HODL ended with 11,319 USD and momentum ended with 10,791 USD.

Ratio

The ratio of total value from the program compared to HODL strategy is shown below. For example, a ratio of 1.2 would mean the total value from the program is 1.2 times the total value by HODL strategy at a specific hour.

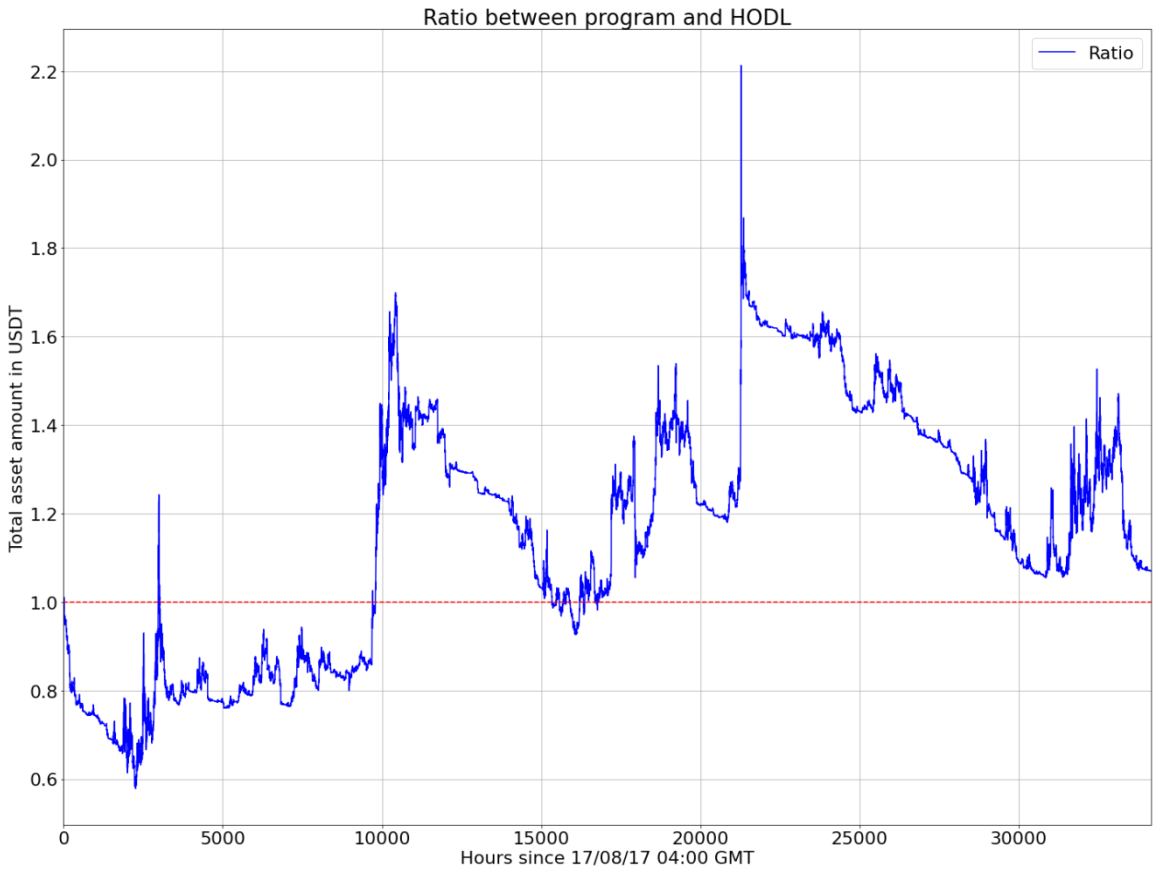

Figure 8: A graph of program results over HODL

From the graph, we can see that for the first 10,000 hours, the program does not perform as well as HODL. The program performance slowly decreases over time, but large spikes in the ratio can be observed, reaching 2.2x value of that of HODL in hour 21,277, which coincides with the sharp drop in total value of HODL in figure 7. This reveals the main advantage of the program is that it can avoid large drops in price.

On average, the program had 1.15 times the value from HODL strategy. The total value from the model is higher than that of HODL for 69.5% of the time.

Return On Investment (ROI)

ROI is a way of determining the effectiveness of trading. It is calculated as follows:

$$ROI = \frac{Final\ Amount - Initial\ Amount}{Initial\ Amount} \times 100\%$$

The ROI is shown in the graphs below:

Figure 9: Graph of ROI calculated monthly

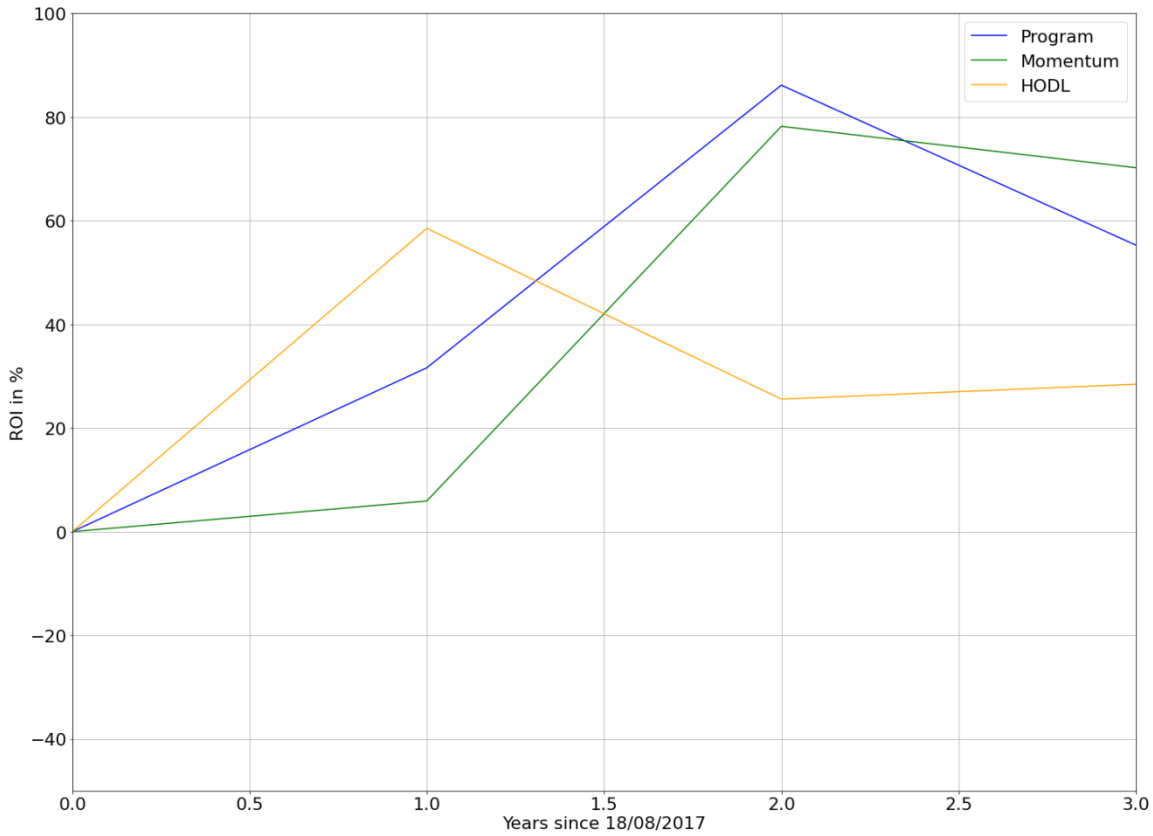

Figure 10: Graph of ROI calculated yearly

As represented in the graphs above, it is shown that both the program and momentum trading strategies perform better.

From the yearly graph, we can see that our program is performing well in the 1st year, while in the 2nd year momentum trading strategies work better.

When we look at the monthly graph, we can see that the main advantage of the program and momentum trading is avoiding large drops. For example, in months 14 and 29, the ROI of HODL strategy drops to below -25%, while the program and momentum strategy stays above -10% for those months. However, the ROI overall fluctuates greatly, demonstrating the extreme volatility of the cryptocurrency market.

Drawdown

Drawdown is a way of determining the 'fear' of losing money in trading. It is calculated as follows, where H is the highest price and L is the lowest price in a certain time interval.

$$Drawdown = \frac{L - H}{H} \times 100\%$$

The drawdown is shown in the graphs below:

Figure 11: Graph of drawdown calculated monthly

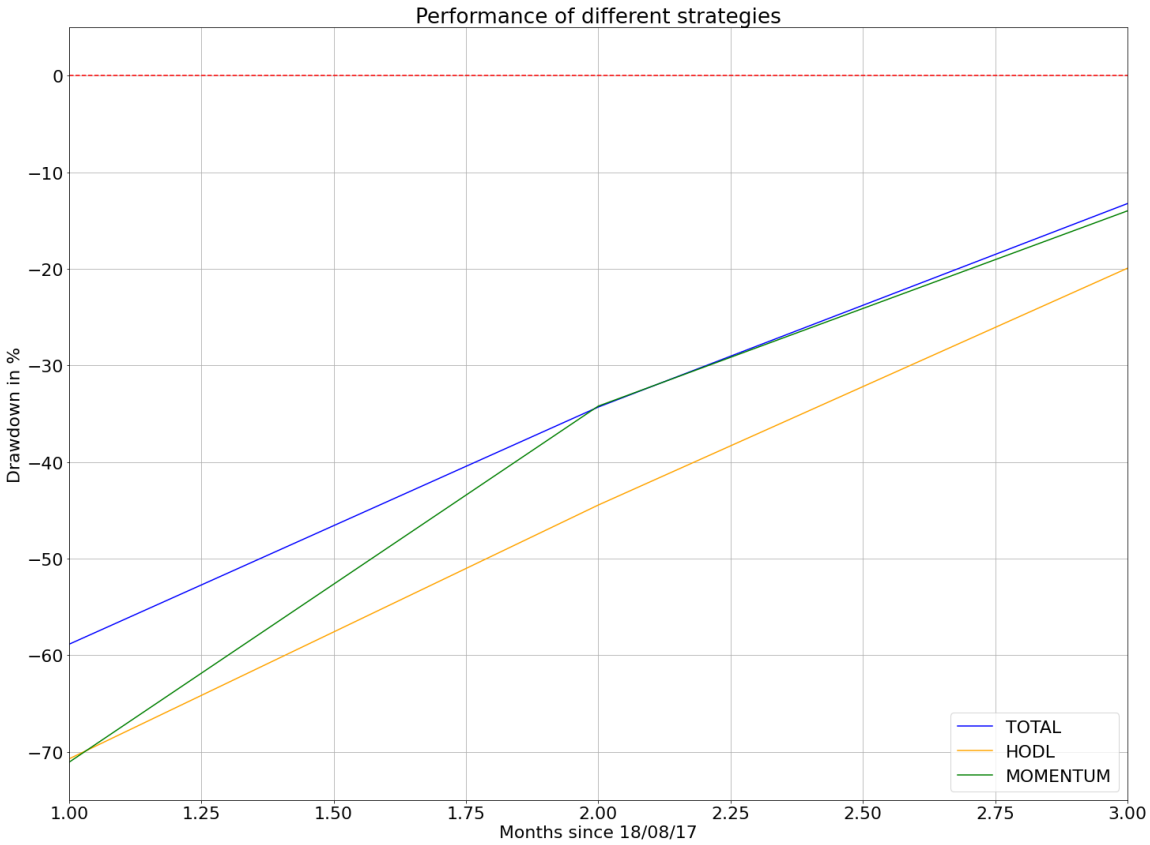

Figure 12: Graph of drawdown calculated yearly

We can see that in general, the drawdown for the program is much lower than that of HODL strategy, while momentum strategy has a similar drawdown as the program.

In the monthly graph, we can see that the drawdown of the program is always higher than -30%, while HODL strategy has dips below -50% drawdown a few times. This demonstrates the HODL strategy will cause more fear than the momentum strategy and program.

It is noted while the program may not be affected by drawdown, real life human traders may decide to sell at a loss due to lack of confidence.

Maximum Drawdown & CAGR

The maximum drawdown and CAGR for each strategy is provided in the following table.

The 'max point' and 'min point' refers to the total value of the assets the program has at each point respectively.

Strategy	Drawdown Max Point (Data Point Location)	Drawdown Min Point (Data Point Location)	Maximum Drawdown	CAGR
HODL	4,736.72 (1775)	1,385.97 (6282)	-70.7%	86.4%
Momentum	3,191.63 (1775)	922.69 (6375)	-71.1%	84.1%
Program	3,191.05 (1775)	1,101.19 (6375)	-65.5%	89.6%

Figure 13: Table of maximum drawdown and CAGR of the strategies

Conclusion

In conclusion, the program is able to train and adapt, as well as perform predictions based on the provided data. The predictions of the program can outperform the commonly used HODL strategy by using the simple 'buy low and sell high' strategy, mainly from avoiding large drops in price. While a maximum drawdown of over 50% is not ideal, the program maintained an average of 1.15 times the return of HODL strategy, with 7% improvement over total value from HODL strategy. Currently, the program can be considered a success.

In the future, the drawdown of the model can be reduced to a lower level, and the ROI of the model can be improved upon. The model may also be test run on cryptocurrency trading platform APIs such as Binance Spot API.