Cryptocurrency Trading with Deep Reinforcement Learning

Haoqi Zhao, Jinlu Ma, Xingdu Qiao, Aayush Dua

1 Introduction

Trading is a broad term and covers a multitude of financial markets, such as stock markets and foreign exchange. Cryptocurrency trading - in simple terms buying and selling cryptocurrency, adds a new dimension to currency trading regime with its dynamic force and volatility. Bitcoin as the first digital currency to be created has an extremely volatile nature. Therefore, Bitcoin trading brings out massive opportunity to capitalize on the risks and allows for maximum yield whenever possible. Furthermore, because of bitcoin's decentralized nature, its price cannot undergo revaluation by certain individuals and is less impacted by industry news, but instead as a consequence of the market as a whole, following the laws of supply and demand, bitcoin trading is extremely popular among traders and is one of the most respected, capitalized and traded cryptocurrency in the world. Because of these reasons, bitcoin trading became a frequent topic explored by machine learning researchers and industry leaders, aiming to develop generalizable and robust models.

In our project, we explored the possibility of using Deep Reinforcement Learning to train an agent to make intelligent long-short decisions in order to maximize the profit-reward function by capitalizing on the opportunity to buy bitcoin when the price is low and sell bitcoin when the price is high. The following sections first discuss relevant work done by other researchers about financial trading using deep learning and reinforcement learning, then explain our approach to implement a non deep learning model baseline, a deep learning model baseline, and an advanced deep reinforcement learning model. Lastly, we compare the performance for each of the models, and conduct analyses on why the performance differs.

2 Related Work

There are several categories of machine learning work that has been done on areas in financial trading. One category is represented by many previous attempts to predict financial instrument prices based on historical prices using machine learning or deep learning. A typical approach would be the model taking an input of a historical price matrix and trying to output a price for the next trading period, as explained in [1]. This idea represents a regression problem where the output is continuous. However, the performance of such models depends largely on how accurate the predictions are made. In real life, trading decisions are not only dependent on the prediction of future price, but also coated with additional trading strategies. Therefore, manually adding a layer of logic to abstract trading strategies is not an optimal machine-learning approach as it requires additional human intervention to produce the ultimate action[2].

Another category of previous work lies in the method of reinforcement learning. Reinforcement Learning first started being used in finance for algorithmic trading in the late 90s. Because of the

limitation of the processing power at that time, deep neural networks could not be implemented at scale. As a result, the reinforcement learning models could only represent small number of possible states and actions, which led to it only being able to address only small-scale problems[3]. Past work on financial trading using RL usually involves a single class, and without the help of deep neural networks to approximate states or values[4][5].

Lastly, there is the category of using deep reinforcement learning, which surfaced most recently. While deep neural networks clearly adds a big bonus to reinforcement learning because of its ability to approximate almost any function and thus represent a large number of states or values, there are different approaches when it comes to reinforcement learning. For example, [6] proposed a deterministic deep reinforcement learning method addressing the portfolio management problem, which directly produces the portfolio vector with raw market data, historic prices, as the input, while [7][8] outputs continuous actions by training a Q function estimator as the reward function, and a second neural network as the action function.

3 Methods

3.1 Feature engineering

Our feature matrix consists of two parts: historical prices and technical indicators. In Markov Decision Process, we assume that the current state and current input are sufficient to determine the next optimal action. However, in real life, unpredictable events and trading behaviors lead to the noisy financial market. Therefore, we are not able to directly observe the actual market states. In this case, historical prices become the part of the underlying market state that we can observe. Thus, we decided to use the minute-frequent prices of Bitcoin (BTCUSD) as the input with a look-back window of 9 to represent the current state. Specifically, the historical price is a 2-dimensional 10*5 matrix, where one axis is the time axis consisting of 10 timestamps including the past 9 minutes and the current timestamp and the other axis represents the open, high, low, close price and volume traded in the 1-minute window.

In addition to historical price, we also included technical indicators in our feature matrix. Technical indicators in technical analysis, is an analysis methodology in trading for forecasting the direction of prices through the study of past market data, primarily price and volume, are often used by traders as observations to price trends. We implemented 4 technical indicators including Simple Moving Average, Exponential Moving Average, Stochastic Oscillator and Standard Deviation to capture the trend, volatility, momentum and statistics of the closing Price. We hypothesize that including these highly used technical indicators will improve our results. These technical indicators are calculated as part of the data pre-processing and concatenated to our original dataset for each data point.

3.2 Non deep learning baseline

We implemented two machine learning models as our non deep learning baseline.

Our first model is a logistic regression. As one of the classic machine learning models for classification problem, our logistic regression model takes training data from Jan 1st, 2020 to Mar 15, 2020 with 9 features, including open, high, low, close price, volume, as well as the 4 technical indicators that we calculated previously such as the 10-minute Simple Moving Average. We tried a few different C values including 0.2, 0.1, 0.01, l1 and l2 penalties, and experimented with solver 'saga'

and 'liblinear'. At the end, we chose c=0.2, with solver='liblinear' and penalty='l2' to be the final parameters. After training the model, we tested the model with data from Mar 15th 2020 to Mar 30th 2020 and calculated the total profit at the end represented as:

$$p = X_{close}(t) * Coin(t) + Cash(t) - Cash_{initial} \tag{1}$$

where $X_{close}(t)$ is the closing price at the current timestamp t, $Coin(t)$ is the number of bitcoin held by our agent at time t, $Cash(t)$ is the cash the agent has on hand at time t, and $Cash_{initial}$ is the initial asset that is given to the agent. The discussion of the result along with comparison to other deep learning models we experimented can be found in section 4.1.

The second model that we deployed is a random forest. As a machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time, random forest is an extremely flexible and easy to use model. Random forest also ran much faster than Support Vector Machine (SVM) when we first experimented with both models. Due to the noise in financial market, we specified the minimum sample leaf to be 50 so that our model would not be so much affected by the random noise. Again, it was trained on the same data as those used in logistic regression, from Jan 1st, 2020 to Mar 15, 2020 with 9 features, and tested on data from Mar 15th 2020 to Mar 30th 2020. The discussion of the result along with comparison to other deep learning models we experimented can be found in section 4.1.

3.3 Deep learning baseline and Experimental Analysis

For our deep learning baseline, we decided to do experiment using RNN, LSTM and GRU models. The choice of those models is based on the market's time-evolution characteristic. Recurrent Networks are used for sequence prediction. In Stock Market Prediction, we predict prices just not by looking the previous timestamp data, but a sequence of previous timestamps data. The reason is because we need to capture the trend, momentum and volatility of a sequence of previous timestamps in stock market to predict Closing price at next timestamp. Hence Recurrent Network models will be quite helpful in our task.

For this task, instead of predicting increase or decrease of the market price, we predict the closing price directly and make an action based on the predicted closing price at the next timestamp and the opening price at that timestamp. Moreover, we will add the attention module to the recurrent model with best performing metric. We expect it to get better result since attention helps the model to take care of some important timestamps like timestamps showing sudden jump up or sudden drop down.

3.3.1 Data

For this task we perform the following operations on data:

1. We implemented Technical Indicators such as Simple Moving Average, Average True Rate, and Relative Strength Index over timeperiod=10 min to capture trend, volatility and momentum. We implemented these technical indicators on Opening Price, Low Price, High Price and Close Price. We tried adding other indicators however after some research we found out that some indicators perform the same task of capturing trend, momentum and volatility. Hence to avoid

redundancy we eliminated the other technical indicators for this task. Including the technical indicators and the original features we got 14 features.

2. We split this dataframe into train(80%) and dev(20%).
3. We then split the train and dev into sequences of 60 minutes.
4. Our target variable will be the closing price after every 60 minutes.
5. Finally we make a train and dev dataloader with batch size=16 and Shuffle=False since we dont want to shuffle the sequence of training and development data because its a time series and hence we want to mantain the same sequence.
6. The final shape of data that is fed to the Recurrent networks is: (16,60,14) where 16 is the batch size, 60 is the sequence length and 14 is the number of features.

3.3.2 Goal

Given the sequence of 60 minutes of data, predict the closing price at 61^{st} minute.

3.3.3 Recurrent Neural Network

The following hyperparameters were chosen for this network:
1. Learning Rate: 0.0001
2. Loss: Torch MSE Loss (Predicted Closing Price at every 61^{st} minute, Original Closing Price at every 61^{st} minute)
3.Optimizer: Adam
4. Hidden Layers:3
5. Number of Epochs:25
6. The hidden Size was varied for hyperparameter tuning:

Hidden Sizes	Train RMSE	Validation RMSE
(128,64,32)	18.224	12.47
(256,128,64)	16.323	11.38
(512,256,128)	17.543	11.47

Observing from the table we see that Hidden Layers with size(256,128,64) gave the best validation results.

3.3.4 Gated Recurrent Network

The following hyperparameters were chosen for this network:
1. Learning Rate: 0.0001
2. Loss: Torch MSE Loss (Predicted Closing Price at every 61^{st} minute, Original Closing Price at every 61^{st} minute)
3.Optimizer: Adam
4. Hidden Layers:3
5. Number of Epochs:30
6. The hidden Size was varied for hyperparameter tuning:

Hidden Sizes	Train RMSE	Validation RMSE
(128,64,32)	16.743	10.572
(256,128,64)	13.798	7.334
(512,256,128)	15.475	9.953

Observing from the table we see that Hidden Layers with size(256,128,64) gave the best validation results.

3.3.5 Long Short Term Memory

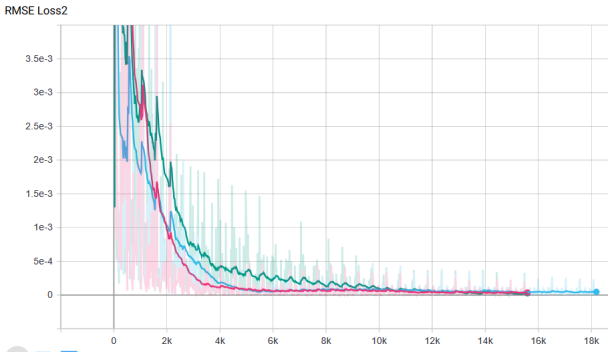
The following hyperparameters were chosen for this network:
1. Learning Rate: 0.0001
2. Loss: Torch MSE Loss (Predicted Closing Price at every 61^{st} minute, Original Closing Price at every 61^{st} minute)
3.Optimizer: Adam
4. Hidden Layers:3
5. Number of Epochs:35
6. The hidden Size was varied for hyperparameter tuning:

Hidden Sizes	Train RMSE	Validation RMSE
(128,64,32)	14.96	9.598
(256,128,64)	14.27	8.119
(512,256,128)	18.576	12.422

Observing from the table we see that Hidden Layers with size(256,128,64) gave the best validation results.

The following is the Training Loss Plots of all the models:

Figure 1: Green:RNN, Pink:GRU, Blue:LSTM



3.3.6 Profit Function

Since this was a regression task, we had to tune the Profit function to suit this task.
The following is the algorithm:

1. Given at the start of each minute, we get the Opening Price of the minute. We already have the predicted closing price of this minute from feeding the model the information about previous 60 minutes.
2. We check if Predicted Closing Price > Opening Price. That means if there is an predicted increase in the value in stock. If there is predicted increase then we buy the stock at the beginning of the minute and sell the stock the end of minute.
3. If there is predicted decrease in the value of stock then we keep our stocks.
4. If we predicted increase and the Real Closing Price at the end of minute is also greater than opening price, then we get profit otherwise loss.
5. If we predicted decrease in value of stock, we did nothing and hence we dont expect any profit from that minute.

The final results of these three models are:

Model	Val RMSE	Initial Cash	Profit Earned
RNN	11.38	$1000000	$ 271121.34
GRU	7.334	$1000000	$ 271332.52
LSTM	8.119	$1000000	$ 272412.74

As we can observe we have used two metrics to compare our model. This first metric RMSE tells us how close was our model in predicting the closing price. The Profit Earned metric was self made to evaluate that given the predicted closing price and the opening price of that particular moment, what is the profit we get.

From the table we can observe the following:

1. RNN has not performed as well as GRU and LSTM in both the metrics. The main reason is:
   - GRU and LSTM performed better than RNN is because GRU and LSTM retains past information whereas in RNN it creates a bottleneck due to which the most recent features dominate alot whereas alot of past information from the sequence is lost in RNN as it progresses in the sequence.
2. LSTM has shown better performance in Profit Metric and GRU has shown better performance in Val RMSE metric. The potential reason could be the following:
   - In Stock Market Prediction, we very well know that although early timestamp features are important however most important information is in the most recent timestamps. LSTM has a long term information cell which might not have been useful in this regression problem since this prediction may be more inclined towards the information in the most recent timestamps than the previous timestamps. Hence GRU got a advantage over LSTM in RMSE metric.
   - In the Profit Metric LSTM gave better performance than GRU which could be because we are only concerned in increase or decrease in value of stock rather than price of stock. It could be possible that Long term cell in the LSTM captured the trend, volatility and

momentum needed to tell the direction of price which RNN couldnt because it doesnt have long term cell.

The Following are the regression plots for the three models:

Figure 2: RNN Regression Plot

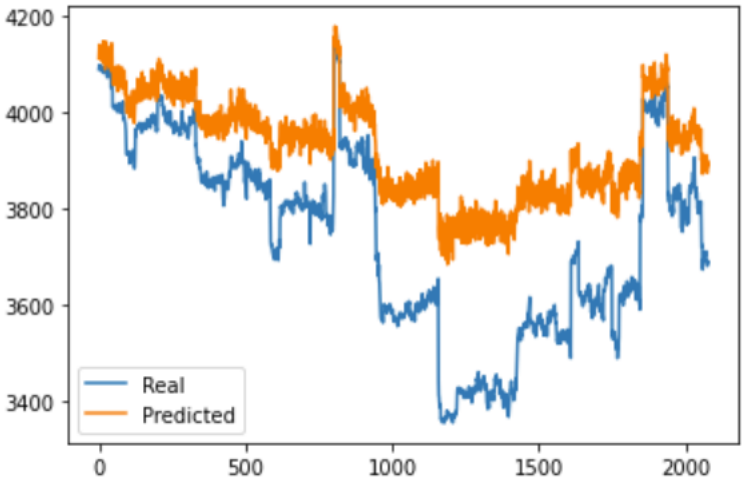


Figure 3: GRU Regression Plot

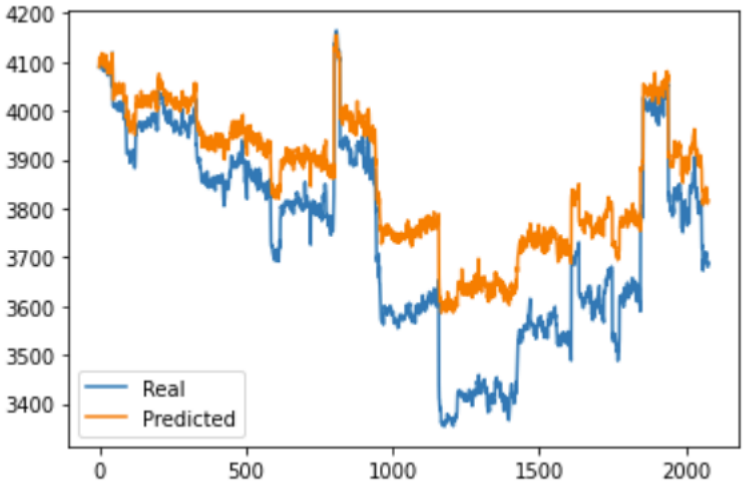
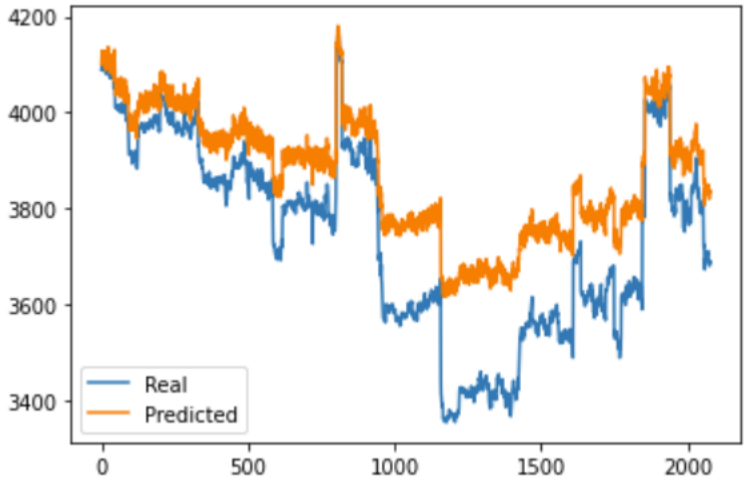
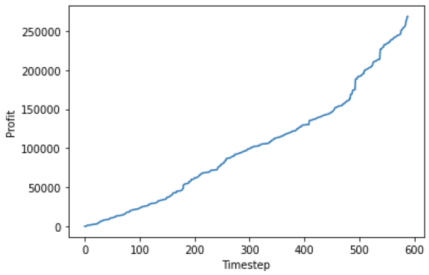


Figure 4: LSTM Regression Plot

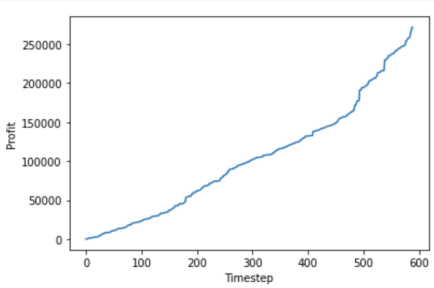


As we hypothesized in our analysis above, RNN was able to capture trend but failed to capture the volatility. GRU and LSTM did good job at capturing the trend and a better job than RNN in capturing the volatility.

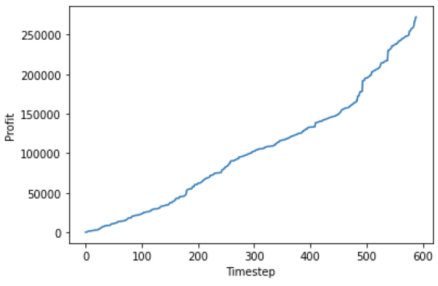
The following are the Profit Plots for the three models



(a) RNN Profit Graph



(b) GRU Profit Graph



(c) LSTM Profit Graph

All the profit graphs look almost same. The most probable reason as we mentioned earlier is because because all the models were successful in fitting the trend. And as we mentioned we are

concerned with predicted increase or decrease in stock prices while calculating the profit. Since all our models were successful in predicting that hence, the rewards graph also follows the same trend even though the final rewards are different.

3.4 Deep reinforcement learning

We implemented two Deep-Q-Networks as our advanced deep learning models where we combine deep neural nets and reinforcement learning to enable our agent to make investment decisions including long, short and keep to maximize the profit-reward function. The training and decision algorithm is the standard Deep-Q-Learning algorithm:

Algorithm 0: Deep Q learning with experience replay

```
Initialize the replay memory M to capacity N;
Initialize the action-value function with random weights;
for episode = 1,E do
    Initialize the environment, the observation s_1;
    while done == False do
        With probability ϵ select a random action a_t;
        Otherwise a_t = max_a Q(s_t, a; θ);
        Execute action a_t in emulator and observe reward r_t and s_{t+1};
        Store transition {s_t, a_t, r_t, s_{t+1}} in M;
        Sample minibatch from M;
        Set y_j = r_j + γ max_a Q(s_t, a; θ) for non-terminal s_{t+1} and set y_j = r_j for terminal
          s_{t+1};
        Perform a gradient descent on (y_j − Q(s_j, a_j; θ))^2
    end
end
```

In the following, we will discuss the environment, the agent, the reward and the Q neural network, respectively.

3.4.1 The environment and the agent

Our goal is to build a trading agent that can buy, sell, keep the stock/asset with the maximum reward. We use Reinforcement Learning agent to perform actions on the Bitcoin Price Series. For each trading period, the environment gives an observation matrix to the agent. The agent generates an action including long, short and keep based on the observation and then tell the environment. The environment changes the state and goes to the next trading period.
The observation $s(t)$ consists of the internal observation $w(t)$ and external observation $X(t)$.

$$s(t) = \{X(t), w(t)\} \tag{2}$$

The internal observation $w(t)$ is the current state describing the current cash and current holding bitcoin. This state will affect the reward and the action and thus should be included. The external environment $X(t)$ consists of the data of 9 timestamps, including the current timestamp and 8 immediate past timestamps. Each timestamp has 10 data: Open, High, Low, Close and other 6 other technical indicators.

3.4.2 Reward function

The trading price is set as the close price of the current period $X_{close}(t)$. The profit get by action $a(t)$ is given by the subtraction between the current total asset and the total asset of time $t+1$:

$$p(t) = Coin(t+1)X_{close}(t+1) + Cash(t+1) - Coin(t)X_{close}(t) + Cash(t) \tag{3}$$

and thus the total reward:

$$p_{total} = \sum_t p_t \tag{4}$$

3.4.3 Q neural network

The Q neural network act as a information extraction processor and is the core of the Deep-Q-Learning. For each action step, the network receives the observation and generate the Q value for "buy", "hold" and "sell". The structure construction should be based on the internal characteristic of the market. Here we use two kinds of structures: CNN and RNN with attention and we will explain them in detail.

It is widely believed that some data patterns like candlestick pattern can reflect the confidence and the condition of the market and can predict the price in the future to some extent. Conventionally, technical analysis uses candlestick consisting of the open price, high price, low price and close price to predict the performance of the market. Here, we have much more complicated patterns consisting of 10 kinds of data. On the other hand, CNN is good at complex pattern recognition and thus we use CNN as our first Q neural network model and the structure is shown in Fig. 3

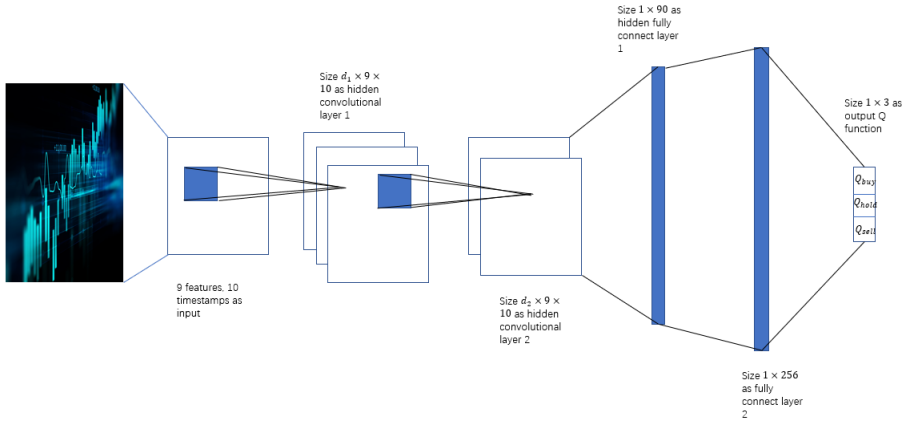


Figure 6: CNN Q Network sketch map

While tuning the hyperparameters, we mainly tune the discount γ in Q learning algorithm, the learning rate, the number of episode and the maximum steps of each episode. We found that γ here is very important. The reason we think is that: If γ is too large, the agent will be too concern about the long-time return. Since there are high risks in the market, the agent has more probability to lose money. On the other hand. if γ is too small, the agent will be too concern about the short-time return and hard to get profit. We tried 1, 0.9, 0.88, 0.85 and finally choose 0.9 as our training hyperparameter. Other hyperparameter:

- Channel 1: 1
- Channel 2: 1
- Learning rate: 0.001
- Optimizer: Adam
- Number of episodes: 10
- maximum steps of each episode: 80000

Another characteristic of the stock market is its change with time. Naturally, this kind of problem can be solved with RNN. Like the natural language, while processing the data, the network should focus on the specific part of the past data. For example, a huge increase (decrease) in the past 9 timestamps stand for the market is optimistic (Pessimistic) and the agent should buy (sell) accordingly. We use attention mechanism in the model to better capture such kind of signal. The structure of RNN with attention model is shown in Fig. 4.

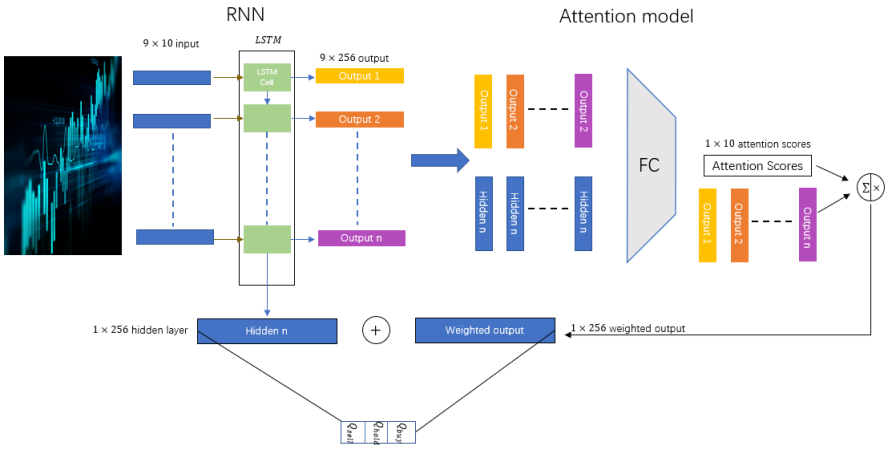


Figure 7: RNN Q Network sketch map

3.5 Advanced deep learning

From our Deep Learning model analysis, we found that GRU and LSTM were able to give very good RMSE and Profit Metric Value. And hence we plan to implement attention module on GRU and LSTM as an advanced Deep Learning Module. We expect it to get better result since attention helps the model to take care of some important timestamps like timestamps showing sudden jump up or sudden drop down.

3.5.1 GRU with Attention

The following hyperparameters were chosen for this network:
1. Learning Rate: 0.0001
2. Loss: Torch MSE Loss (Predicted Closing Price at every 61^{st} minute, Original Closing Price at every 61^{st} minute)

3.Optimizer: Adam
4. Hidden Layers:3
5. Number of Epochs:35
6. The hidden Size was varied for hyperparameter tuning:

Hidden Sizes	Train RMSE	Validation RMSE
(256,128,64)	15.23	14.119
(512,256,128)	11.73	11.369

Hence we will chose hidden size= (512,256,128) 7. The Attention Module hidden layer sizes chosen after tuning were:(512,256)

3.5.2 LSTM with Attention

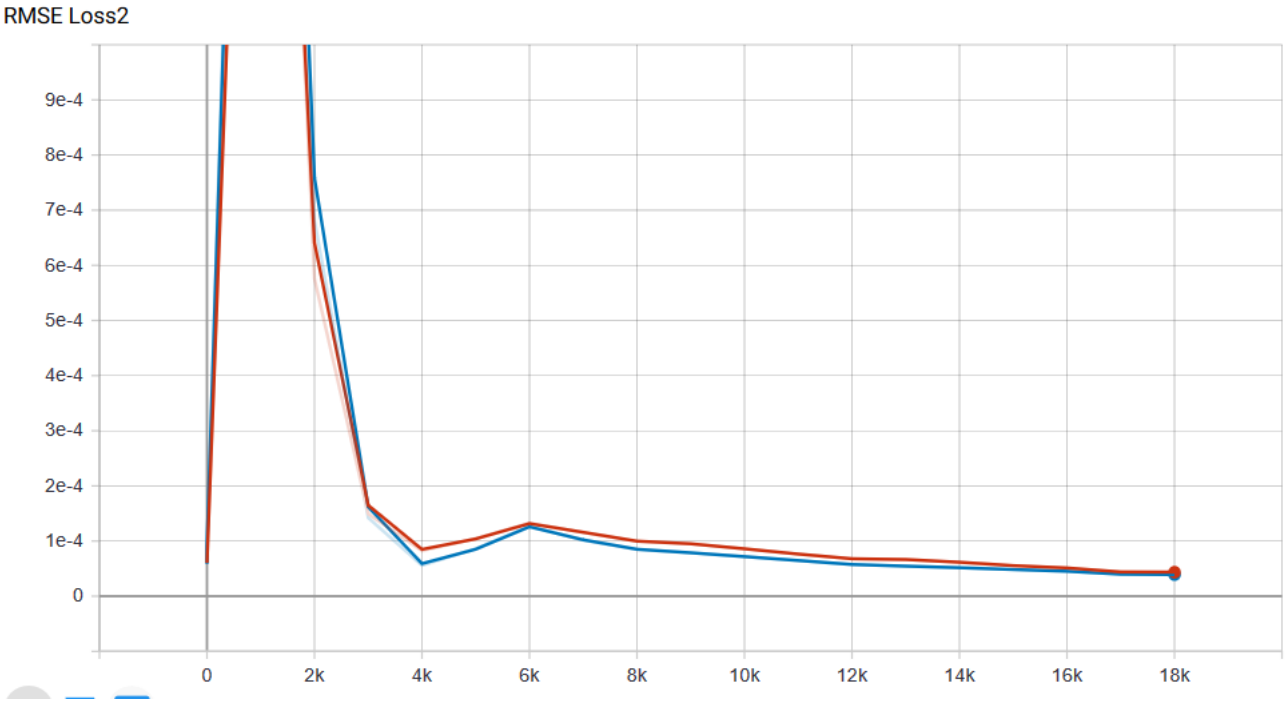
The following hyperparameters were chosen for this network:
1. Learning Rate: 0.0001
2. Loss: Torch MSE Loss (Predicted Closing Price at every 61^{st} minute, Original Closing Price at every 61^{st} minute)
3.Optimizer: Adam
4. Hidden Layers:3
5. Number of Epochs:35
6. The hidden Size was varied for hyperparameter tuning:

Hidden Sizes	Train RMSE	Validation RMSE
(256,128,64)	18.224	12.329
(512,256,128)	15.304	8.713

Hence we will chose hidden size= (512,256,128) 7. The Attention Module hidden layer sizes chosen after tuning were:(512,256)

The following is the RMSE Loss Plot for both the models:

Figure 8: Red:GRU with Attention, Blue: LSTM with Attention



From this plot we can observe that both GRU with Attention and LSTM with Attention have converged really well on the training data.

The following are the Profit Metric when $1000000 were invested and RMSE Metric results for the two models:

Models	Val RMSE	Profit Earned
GRU with Attention	11.369	$268606
(512,256,128)	8.713	$270885

We can observe that LSTM with Attention was able to perform better than GRU with attention on both VAL RMSE and Profit Metric.
The following could be the reasons why LSTM with Attention performed better than GRU with Attention:

1. It has a Long Memory known as cell state and short memory known as hidden state whereas in GRU both are combined in a single hidden state thus losing information. Although GRU is computationally less expensive than LSTM due to less gates and memory however LSTM proved to be better.

2. Secondly we discussed in the experimental analysis of Deep Learning models that in the Profit Metric LSTM gave better performance than GRU which could be because we are only concerned in increase or decrease in value of stock rather than price of stock. It could be possible that Long term cell in the LSTM captured the trend, volatility and momentum needed to tell the direction of price which RNN couldnt because it doesnt have long term cell.

When compared with Deep Learning models without attention, our deep learning models with attention performed poorer. We were expecting attention module to improve the results because it will focus on more important timestamps however it is the opposite. The most probable reason for this is because Deep Learning models with Attention can be computationally very expensive. Even though we increased the number of epochs, the models due to its complexity is very vulnerable to overfitting the data and hence we got these results.

4 Experiments and Analysis

The data used in the experiments is the historical bitcoin data coming from GEMINI. The training time period is from Jan. 1st 2020 to Mar. 15th 2020, including 100,000 timestamps. The testing time period is from Mar. 15th 2020 to Mar. 30th 2020, including 20,000 timestamps. We set the initial total asset to be 1 million and compare the final rewards we can get after testing.

4.1 Performance comparison

First, Figure 9 and Figure 10 show the results from our non deep learning baselines. Figure 9 shows the resulting profit graph from our logistic regression, where the blue line is our profit and the orange line is bitcoin baseline (price trend). As shown in the graph, logistic regression earned a 306% profit of 3,060,358, when given 1,000,000 dollars to start with.

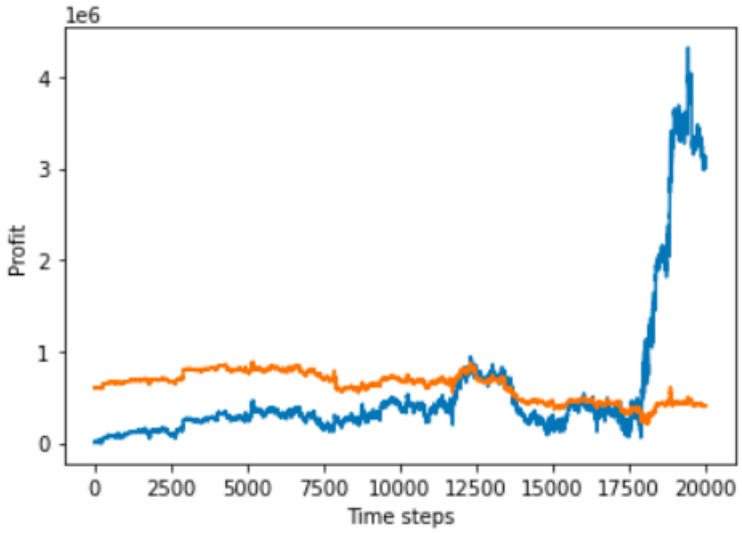


Figure 9: Profit trend using Logistic Regression

Figure 10 shows the resulting profit from our random forest model. As shown in the graph, random forest achieved a 444.8% profit of 4,447,971, when given 1,000,000 dollars to start with.
Figure 11 shows the test result of our CNN based deep Q learning. As shown in the figure, the CNN-based deep Q agent achieved a 265.7% profit of 2,656,966, when given 1,000,000 dollars to start with. Figure 12 shows the test result of our RNN based deep Q learning. As shown in the

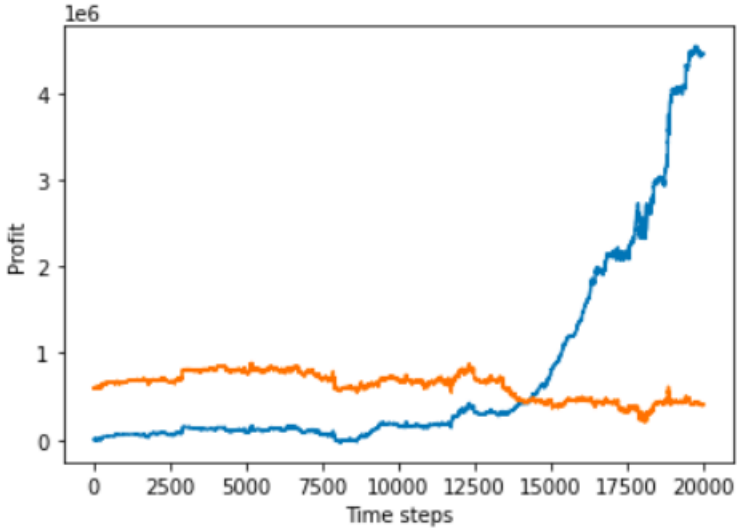


Figure 10: Profit trend using Random Forest

figure, the RNN-based deep Q agent achieved a 39.7% profit of 396,582, when given 1,000,000 dollars to start with. We have fully tuned the parameter but the result is still not as good as other methods. We will try to explain the results in the next section.

4.2 Reason analysis

In this section we mainly compare the 2 deep reinforcement learning results. The CNN-based agent performs much better than RNN+attention based agent. We think that's because in the market trading problem, what really matters is the market trend shown in the historical price pattern rather than some specific price. Therefore, professional market trader need to be good at price pattern recognition. For the same reason, CNN is good at recognize the pattern and therefore have much better performance. On the contrary, RNN, although can memory the previous price, is not good at pattern recognition and thus the preformance is bad.

5 Discussion

In this report we have tried out different machine learning and deep learning models for financial trading. First we built two machine learning models as the baselines, one based on logistic regression and the other random forest. Then another baseline based on recurrent neural network was built as well. After building these three baselines, a deep reinforcement learning network was built. We trained and tested these four different models separately and the results are demonstrated and compared in the sections above. Overall, our deep reinforcement network showed the best performance. The agent trained shows the ability to triple the total asset in the test.

Our reinforcement models can be further improved if we were given more time. First, the element of transaction fee should be taken under consideration. In the current work, there is no transaction fee

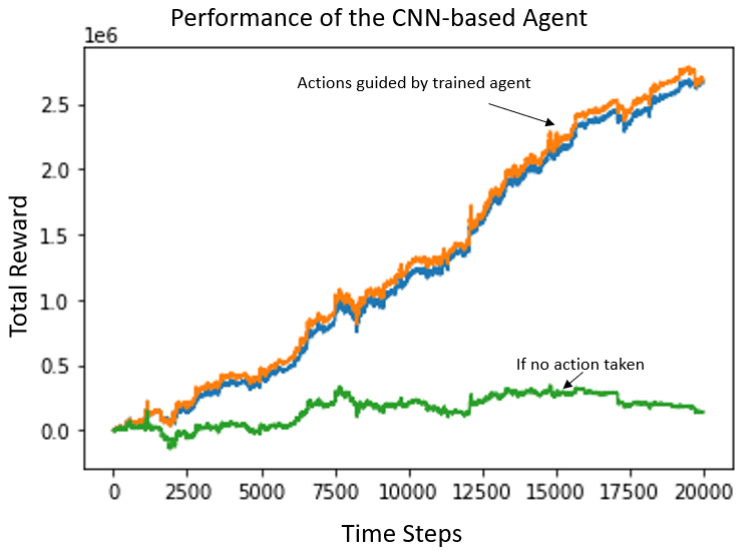


Figure 11: Profit trend using CNN-based agent

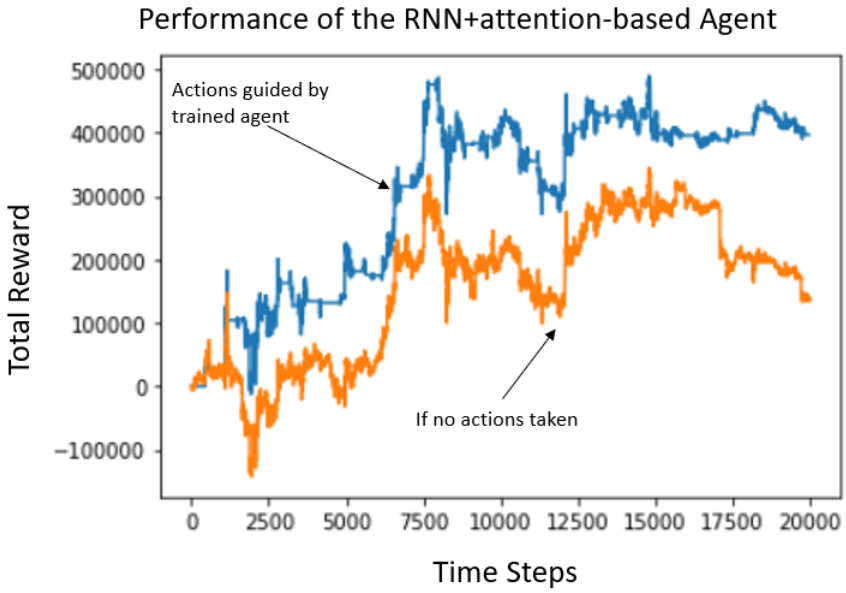


Figure 12: Profit trend using RNN-based agent

considered. However, in practice this can greatly affect the trading strategy. Therefore, we would explore the influence brought by transaction fee if we were given more time. Second, in the current experiment, we found that the model tends to show bad performances once there is a relatively sharp decrease of the bit-coin prices in the training dataset. We would explore the mechanism to smooth the curve either by doing this during the data pre-processing or by burying the curve-smoothing mechanism inside our machine learning or deep learning model.

6 References

[1] J. B. Heaton, N. G. Polson, and Jan Hendrik Witte. Deep learning for finance: deep portfolios, 2018.
[2] Fabio D Freitas, Alberto F De Souza, and Ailson R de Almeida. Prediction-based portfolio optimization model using neural networks, 2009.
[3] John Moody, Matthew Saffell. Learning to Trade via Direct Reinforcement, 1997.
[4] James Cumming. An investigation into the use of reinforcement learning techniques within the algorithmic trading domain, 2015.
[5] M.A.H. Dempster and V. Leemans. An automated FX trading system using adaptive reinforcement learning, 2006.
[6] Zhengyao Jiang, Jinjun Liang. Cryptocurrency Portfolio Management with Deep Reinforcement Learning, 2017.
[7] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic Policy Gradient Algorithms, 2014.
[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous Control with Deep Reinforcement Learning, 2016.
[9] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading, 2016.
[10] Zhengyao Jiang, Dixing Xu, Jinjun Liang. A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem, 2017.
[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning, 2015.
[12] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution, 2006.