Trading Bot with Joint Action Learning with Deep Agent Modeling

Jaejin Cha

December 2, 2024

Abstract

We described an approach for financial trading bot with multi-agent reinforcement learning (MARL) algorithm, where the trading bot aims to maximize its profit as well as predicting the next change in the market. This approach is suitable as the trading price gets determined by the other traders, making agent modeling a reasonable approach. In this work, we delivered the implementation of a MARL algorithm, especially in Joint-Action Learning with Agent Modeling (JAL-AM), as well as a simple but applicable Markov Decision Process (MDP) based on the chart representing the market data over time. Then, we demonstrate the result from this experiment and discussed the conclusion.

1 Introduction

In the recent 5 years, there has been a huge ups and downs in the financial trading market due to various reasons. There has been a COVID-19 virus outbreak, huge price increase in cryptocurrency, high inflation in the economy under the Biden administration due to COVID-19 relief spending, the Russia-Ukraine war, and on-going huge expectation in artificial intelligence technology. Due to such dramatic changes in the financial market, by 2024, the number of people investing in stock has increased by 8% since 2014 [2].

Also, in recent years, the research in artificial intelligence has exploded in popularity, due to its profound performance and usability as well as its potential to solve various problems. And, there are rising fields of study within artificial intelligence, such as generative AI, multi-agent reinforcement learning (MARL), physical AI, etc. And as the speed of discovering new knowledge is very fast, active studies on the application of artificial intelligence, such as autonomous driving, chatbot, AI agent, etc., are in high demand.

In this work, we examine the application of MARL in finance trading to predict the change in price in the market and trade accordingly to exploit the change in the market. Then, we deliver the specific procedure to demonstrate setting the Markov Decision Process on the market and the architecture of the model, and we evaluate the performance of this approach.

2 Related Work

2.1 Trading Bot

There are a lot of on-going research in financial trading bot, so it is hard there exists a certain algorithm beating the market the best. But it is not hard to see that the high-performing trading bots are using Artificial Intelligence (AI) to analyze the market and perform corresponding actions. Especially in the field of cryptocurrency, there are heavy interest in cryptocurrency trading bots, as the cryptocurrency is well known for extreme change in the market, without well defined factors, such as dividends.

There are stock trading bot services, such as Alpaca and Interactive Brokers, requiring fees for the service, which may be a huge amount over time.

2.2 Joint Action Learning with Agent Modeling

Joint Action Learning with Agent Modeling (JAL-AM) is a MARL algorithm which trains a model of other agents and perform actions to exploit them. The approach on agent modeling can use tabular approach. But the tabular approach is limited due to its inability to generalize across states, there is another approach using deep neural network. In this work, we used deep neural network approach, training agent modeling to find the strategy of the market based on the market's change and trading bot will exploit the strategy.

3 Methodology

3.1 Market Dataset

To get a clear result from this experiment, it is necessary to use market data with minimum external factors. As a stock price is not only dependent on the information available in the chart of the company, using a stock of a certain company requires a broad knowledge as well as a very specific knowledge, ranging from predicting the market in the next decade to knowing the current cash reserve of the company.

Collecting the data to represent the external factors, by web-scraping, is extremely difficult as we need a fair amount of the data across the time, possibly creating a huge but vague information gap between each state. This problem does not disappear even when we choose an exchange-traded fund (ETF), as it also requires similar knowledge of the companies listed in the fund.

So we chose the historic dataset of Bitcoin [3]. Because Bitcoin does not have dividend, cash reserves, etc., this allows the trading bot to observe sufficient knowledge of the market with the chart. But since there are some ranges of time where there are extreme ups and downs in the chart, we removed such ranges from the training and evaluation process. As a result, we used the Bitcoin data with 1 minute gap on each timestep in 2017 and 2018 to train and 2019 to test, with attributes on open price, high price, low price, close price, volume of Bitcoin, and volume of USD.

3.2 Model Architecture

3.2.1 Neural Network

The neural network we used in the experiment is designed with fully connected architecture featuring progressive layer dimensionality reduction and dropout regularization. This network is used for both market network and trader network, which are described below. The network takes an input of variable dimension 'input_dim' and progressively narrows its layers from 512 to 32 neurons, using ReLU activation functions and 0.5 dropout at each hidden layer to prevent overfitting, and eventually the network reduces to the dimension configured with a variable 'output_dim'.

In the forward pass, the input is processed through the sequential layers, and the final output is transformed using a softmax, to consider a sampling approach in action selection, which is not used for our project as we used argmax approach.

3.2.2 Market Network

The market network will be treated as a policy network of other agents, as the market moves based on other trader's strategy. In the reference to the joint action learning with deep agent modeling [1], it refers to apply a supervised learning for modeling the other agent. But in the case of having a continuous degree of the action, we modified it to use a reward function to calculate the degree of performance based on how similar the prediction was. Then, we used nn.CrossEntropyLoss() to calculate the loss, and optim.AdamW to optimize the neural network, with a given learning rate. And, to stabilize the training process, we used Deep Q-Network to train for the market network.

3.2.3 Trader Network

The trader network takes the role of the policy network in the joint action learning with deep agent modeling [1]. This will take the output from the agent model, which is the market network, as well as observing the same observation as the market network. We also added the current status of the

Num	Action
0	buy a coin
1	sell a coin
2	no-op

Table 1: Action Space

Market Network	Trader Network
Bitcoin open price	Bitcoin open price
Bitcoin high price	Bitcoin high price
Bitcoin low price	Bitcoin low price
Bitcoin close price	Bitcoin close price
Bitcoin volume	Bitcoin volume
USD volume	USD volume
-	market network prediction on BUY
-	market network prediction on SELL
-	market network prediction on NOOP
-	current budget
-	current number of coins holding

Table 2: Observation Space

agent to enhance the understanding of its own state, which is its current budget and current number of coins it already has. By understanding such, the trader can make actions to increase efficiency in its strategy on how much to buy or sell over time. This network also used nn.CrossEntropyLoss() to calculate the loss and optim.AdamW to optimize the neural network with a given learning rate. And, to stabilize the training process, we also used Deep Q-Network to train the trader network.

3.3 Markov Decision Process

3.3.1 Action Space

For the sake of stability, which is important for observing stable performance and for practical use of trading bot, we will limit the action by the number of bitcoins an agent can buy or sell at a time by 1. And, we will disable buying bitcoin when there is not enough budget, to test its usability in a realistic situation. Also, we will disable selling bitcoin when there is no bitcoin held by the trading bot. By disabling, it means the program will change to No-op before performing the disabled action after checking the condition. We will not limit the maximum number of bitcoin it can hold.

the market agent will be assumed to have the same action space as the trader agent. We will set a threshold. And, if next_value > current_value + threshold, we will assume the market agent bought bitcoin. If next_value < current_value - threshold, we will assume the market agent sold bitcoin. If next_value $\leq$ current_value + threshold & next_value $\geq$ current_value - threshold, we will assume the market performed No-op. Intuitively, we can think of the threshold as a transaction fee in investing applications. If we can benefit less than the transaction fee, it is better to not consider it as a benefit.

3.3.2 Observation Space

The market network's policy will be based on observation of the market (open, high, low, close, Volume BTC, Volume USD) and predict the market's action. To make the trader bot to analyze the trend over time, with a given configuration of time range, the network will take the series of observations from the most recent observation of the market as long as the configured time range, which is configured to be 10 timesteps. And, the trader agent will take the same observation as above as well as the inputted agent's action, current budget, and number of bitcoins it holds. And, since the market data contains a range of the price, instead of a single price, we will assume the price the trading bot can buy or sell is the average price between open and close.

3.3.3 Reward Function

The reward function is simple. It gets a reward as much as it sells and a negative reward as much as it spends. Also, for each transaction, such as buy and sell, there will be a small amount of fee which is a scalar that subtracts from the reward. Therefore, the reward function is the amount of change in the trader's budget - transaction fee.

The fact that there is a negative reward on spending may scare the trading bot to not buy, but optimistically, it may learn that there needs a coin to sell and earn money. There could have been various approaches to creating a reward function, such as giving reward based on the change in its net worth etc., but we thought that having a simplest form will deliver more insight.

As we used the reinforcement learning to train the market network, we defined the reward function for it as well. We want the market network to predict the next move in the market, while the market network outputs the same action as policy (buy a coin, sell a coin, no-op), so we set the threshold for defining how much change in the market defines buying a coin, selling a coin, and no-op. In this experiment, we set a variable 'market_prediction_threhold'. If the market moves more ratio than the threshold, we treat this to be buying a coin, and so on.

3.3.4 Transition Function

For the transition function, we take the action of the trader network, and change the status of the trader accordingly, such as budget and number of holding coins. But before allowing such actions, we limited the action, such as if it does not have enough budget, it cannot buy a coin, as well as it cannot sell a coin if it does not have any coin. This helped the simulation to be more realistic.

3.4 Configuration

For the further details in the configured values for the experiments, we put the configured values in Table 3.

4 Result

The training process for each year took about 3 hours, using mps on M2 Max chip. And about an hour for evaluation process for each year, using mps on M2 Max chip as well. Here is the link to the repository https://github.com/jaejin0/jal-am-trading-bot

As you can see in the Figure 1, the training process achieved a moderate increase in net worth, which adds the trader's budget and the value of the coins it holds. Also, in the evaluation process achieved a greater result in increase in net worth.

5 Conclusion

The result shows a great performance on getting high net worth, as it was able to get much higher net worth within a year. But the concern is that the model did not trade actively. For example, the model bought a lot of coins at the beginning and held it until the end of the time without having lots of actions. I think this can indicate two interpretation, one is good and one is bad.

First, such strategy might indicate the agent realized the short-term trading approach has too much randomness and decided to trade with a long-term trading approach. The training data we used in the experiment had a slow but certain increase in the price, which would be reasonable for the trading bot to realize that having a long-term strategy would maximize the reward. If this is the correct reason why the model performed such way, it would indicate great success in the experiment.

But second, there is a chance where the agent was not trained correctly. The model performs to buy coins in the beginning and do not sell the coin, which gives a good result in the net worth. But if the model is not actively exploiting the market's movement, it is not the intended result of the experiment, since the experiment focused on short-term trading. And, the net worth of the model tends to follow the Bitcoin price too much, where the model is heavily depended on the market strategy, instead of exploiting the market strategy.

Variable	Value	Description
train data	BTC-2017min.csv, BTC-2018min.csv	minutely bitcoin chart data from [3]
test data	BTC-2019min.csv	minutely bitcoin chart data from [3]
market_observation_feature_dim	6	described in observation space
market_observation_time_range	10	how many timestep in the past to observe
action_dim	3	buy, sell, noop
trader_state_dim	2	current budget, current number of holding coins
budget	100000	initial budget
transaction_fee	0	set to be 0 for this experiment
buffer_size	100000	replay memory buffer size for training the model
learning_rate	0.001	learning rate for optimizer
target_update_rate	0.005	how soft we want to train the target network in DQN
discount_factor	0.99	discount factor for the next reward
batch_size	128	how much data we want to sample from the replay buffer to train
exploration_parameter	1.0	initial exploration parameter
exploration_end	0.001	minimum exploration parameter
exploration_decay	0.99999	how much we want to change the exploration parameter every timestep
market_prediction_threshold	0.001	ratio for market data to define what action it perform

Table 3: Configurations

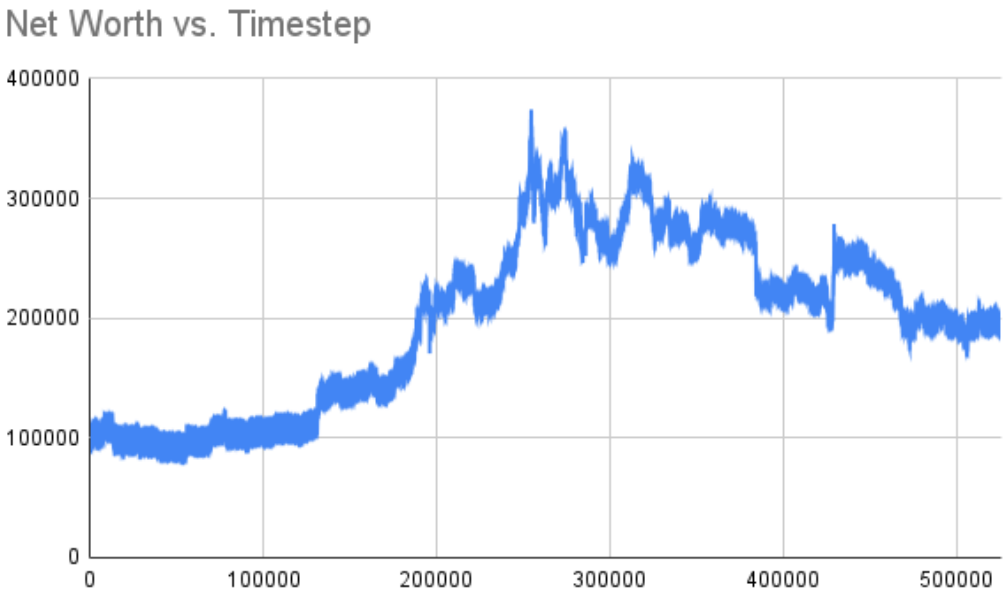

Figure 1: Net Worth vs. Timestep in training

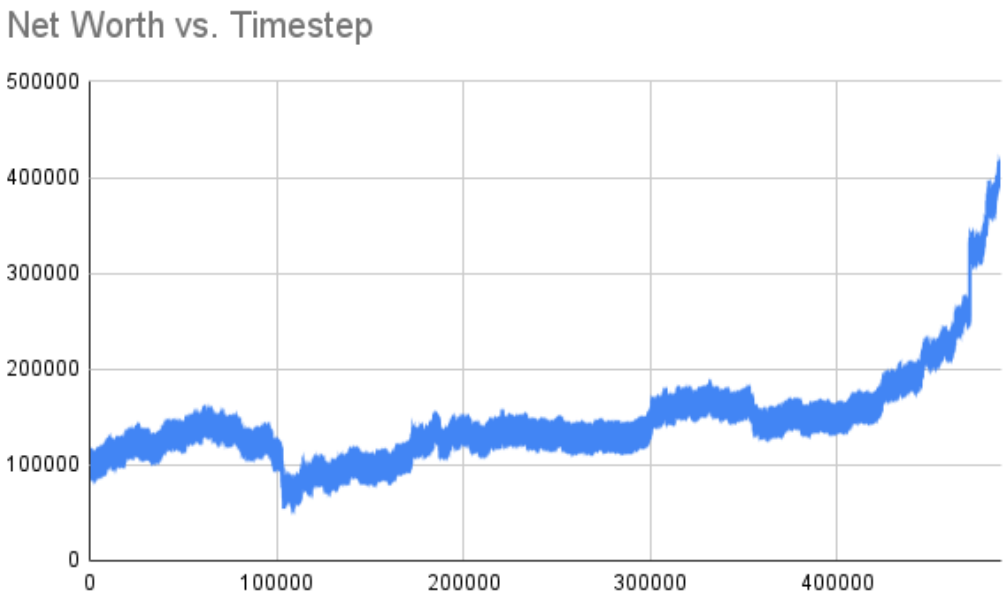

Figure 2: Net Worth vs. Timestep in evaluation

6 Future Direction

I think the conclusion interprets two things, where the model is very well trained as it gets very high net worth or where the model is not properly trained as it does not actively trade over time. I think there can be improvement on experiment design to clear such interpretation. For example, we can use various training data of other stocks, where the price has not much change over time, unlike Bitcoin. And, we can also test each neural network since there are two neural network working simultaneously, giving vague interpretation on the performance. I think if we can test each neural network's performance, we can have more clear result.

References

[1] S. V. Albrecht, F. Christianos, and L. Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024.

[2] J. Caporal. *How Many Americans Own Stock? About 162 Million – but the Wealthiest 1 The Motley Fool, 2024.*

[3] prasoon_k. Bitcoin Historical Dataset. *Kaggle, 2021.*