Deep Reinforcement Learning for Margin Trading

Seoyong Lee
Dept. of Electrical and Computer Engineering, SNU
sylee2685@snu.ac.kr

Abstract

We utilize Deep Reinforcement Learning algorithm to devise an adaptive trading strategy for margin trading environments. Unlike ordinary stock markets, margin trading environment allows the use of leverage, which enables traders to maximize profit under risk of liquidation. We adopted Deep Deterministic Policy Gradient algorithm and used Bitcoin price data from 2017 to 2022 for training and testing the trading agent. The results showed that choosing an adequate reward function can provide the agent with a profitable and secure trading strategy.

1. Introduction

Nowadays, billions of dollars are being traded on the stock market based on quantitative trading, electronically. However, finding an optimal strategy in stock trading is an arduous task with no clear answer. One way to attack this complex problem is applying deep learning. For example, concepts such as convolutional neural network [2] and recurrent neural network [1] were utilized to tackle this problem.
Utilizing deep reinforcement learning is an alternative way to solve trading problems. Since the goal of trading is to maximize the expected profit, and the problem is easy to represent as Markov Decision Process(MDP), reinforcement learning is a suitable choice. For example, study from Xiong [4][5] applies deep reinforcement learning algorithms to trading under stock market consisting of Dow Jones stocks. Study from Zhang [3] utilizes deep reinforcement learning under the condition that having both short and long positions are possible.

In ordinary stock markets, investors can buy stocks up to their equity, the money they have. However, in margin trading environment, investors can buy stocks more than their equity. They can leverage their money so that their total market asset can be larger than their equity. One example of this margin trading environment is the futures market, which consists of numerous types – commodities futures, currency futures, interest rate futures, stock market index futures, cryptocurrency perpetual futures and much more.
However, trading algorithms that can be employed in margin trading environment are not studied yet. For this reason, we will investigate the effectiveness of deep reinforcement learning algorithm in margin trading environment. To be more specific, Deep Deterministic Policy Gradient(DDPG) algorithm with bitcoin price data(BTCUSDT) will be used. We will check whether deep reinforcement learning can be used in margin trading environment, and if so, we will illustrate the effectiveness of the algorithm and visualize the strategy it uses.

2. Preliminaries

2.1. Defining Margin trading and leverage

Margin trading is different from normal trading environment in two aspects.
First, in margin trading, it is possible to use leverage and it is possible to create a short position. Unlike normal stock trading where you can only bet on the price going up (long position), you can also bet on the price going down (short position). Also, as described in the introduction section, margin trading environment allows the use of leverage. Equity of the trader is called (initial) margin M, and total market asset, which is also known as total portfolio value, can be represented by the quantity Q multiplied by the current price of the product Pr. The leverage L is then calculated as following.

$$L = QPr/M$$

Taking short position into account, we will allow Q and L to have negative values. In short position, we first sell the product before buying, so it is natural for Q to have a negative value.
Second, in margin trading, concept of liquidation exists. When the sum of margin and unrealized profit becomes smaller than zero, the margin becomes zero and trader loses all the money, not being able to trade furthermore. For example, if the trader sets the leverage L to 10, the total market asset is ten times the margin, $QPr = LM = 10M$. When the price of the product decreases more than 10%, unrealized profit becomes smaller than $QPr \times \frac{9}{10} - QPr =$

$-\frac{1}{10}QPr = -M$. Therefore, liquidation happens, and trader loses all the money. Using high leverage allows the trader to maximize profit, but it comes with high risk. This makes margin trading environment more complex than ordinary stock markets.

More formally, liquidation price is given as

$$Pr(1-\frac{1}{L})$$

This equation holds for both long position(L>0) and short position(L<0). For long position, if the current price becomes smaller than the liquidation price, liquidation happens. For short position, if the current price exceeds the liquidation price, liquidation occurs.

It is necessary to note that leverage also affects the trading fee. The trading fee is calculated by multiplying margin, leverage, and fee rate all together.

To summarize, margin trading environment is where the trader can use leverage on both long and short positions. We will tackle the problem where the leverage is not fixed but can be controlled by the trader.

2.2. Deep reinforcement learning

The goal of reinforcement learning is to learn an optimal policy that maximizes the expected cumulative reward

$$E^{\pi}\left[\sum \gamma^t r(s_t, a_t)\right]$$

Using deep neural network as a universal function approximator, a lot of different approaches were made to solve the problem.

Deep Q-Networks(DQN) [8] is a deep reinforcement learning algorithm that parametrize Q function(Q_φ). The dataset $\{s_i, a_i, r_i, ns_i\}$ is collected, stored in replay buffer. The data is then used to generate target value

$$y_i = r_i + \gamma \max_a Q_{\varphi^-}(ns_i, a)$$

Using gradient descent, we then update φ so that

$$\sum_i (\, Q_\varphi(s_i, a_i) - y_i)^2$$

is minimized. The use of replay buffer and target network φ^- is an important concept of DQN.

On the other hand, we can also parametrize the policy(π_θ) to tackle reinforcement learning problem. The policy gradient theorem [7] tells us that $\nabla_\theta J(\theta)$, the direction θ should be updated, can be estimated as

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i\{\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t, s_t)\}\{\sum_{t=0}^{T} r(s_t, a_t)\}$$

Therefore, we can collect samples to perform gradient ascent, eventually obtaining optimal policy.

When we use policy gradient theorem in a naïve manner, the sum of rewards is estimated by Monte Carlo approach. However, we can also parametrize the value function(v_φ^π) at the same time we parametrize the policy(π_θ). This is known as actor-critic algorithm. We collect samples, update critic, v_φ^π, minimizing

$$\sum_i \left(v_\varphi^\pi(s_i) - \left(r_i + v_\varphi^\pi(ns_i)\right)\right)^2$$

Based on our critic, we calculate the sum of rewards as

$$\sum_{t=0}^{T} r(s_t, a_t) = r(s_t, a_t) + v_\varphi^\pi(s_{t+1})$$

which is then used to calculate $\nabla_\theta J(\theta)$, which is used to update our actor π_θ.

Lastly, Deep Deterministic Policy Gradient (DDPG) algorithm combines the concept of deterministic policy gradient and DQN. The critic network is updated in the direction that minimizes

$$\frac{1}{N}\sum_i (Q_\varphi(s_i, a_i) - \left(r_i + \gamma Q_{\varphi^-}(ns_i, \mu_{\theta^-}(ns_i))\right)^2$$

The actor network is updated based on deterministic policy gradient [9], which can be estimated as

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \nabla_\theta\, \mu_\theta(s_i) \nabla_a Q_\varphi(s_i, a)|_{a=\mu_\theta(s_i)}$$

DDPG algorithm uses the concept of replay buffer of DQN and uses 4 different networks $\varphi, \varphi^-, \theta, \theta^-$, the critic network, target critic network, actor network, and the target actor network.

3. Problem statement and methods

We define the market environment as MDP and state our problem. Then, we discuss through two essential components for simulation– the environment and the agent.

3.1. Markov Decision Process formulation

- $state\ s:\ \ s \in R^{14}$, consisting of 14 real values. Current leverage, average entry price, open, high, low, close(OHLC) prices, Moving Averages(window size of 5,10,20,60,120), volume, Moving Average Convergence Divergence(MACD), and Relative Strength Index(RSI)
- $action\ a:\ a \in R$, is a real value. It is the action that agent can take, which is changing the leverage (which is equivalent to buying or selling asset). We will restrict the range of this action to a certain value L_{max}, forcing
$$a \in [-L_{max}, L_{max}].$$
- $reward\ r: r(s, a)$ is calculated based on the current and past total portfolio values. In other words,
$$r = \ f(current\ portfolio\ value, past\ portfolio\ value)$$
- $policy\ \pi:\ \pi(s) = a$ is the strategy for trading and the goal of this problem

Our goal is to find an optimal policy that maximizes the expected reward, which would lead to maximizing profit.

3.2. Environment

Given (state, action) pair, the environment should calculate next state and reward. The calculation is done as following. M denotes margin, L denotes leverage, Ep denotes average entry price, Q denotes quantity, and P denotes the total portfolio value. Open, high, low, close denotes the OHLC prices. The subscript p denotes the past value, value right before the current value.

```
1. Calculate $M, Q, Ep$
   $if\ position\ is\ reversed$:
     $M \leftarrow P_p, Q \leftarrow \frac{ML}{open}, Ep \leftarrow open$
   $else$:
     $M \leftarrow M_p, Q \leftarrow \frac{ML}{open}$
     $if\ abs(L) < abs(L_p)$:
       $Ep \leftarrow Ep_p$
     $else$:
       $Ep \leftarrow abs\left(Ep_p\left(\frac{Q_p}{Q}\right)\right) + abs(\frac{open(Q - Q_p)}{Q})$

2. Calculate trading $fee$, update M, L
   $fee \leftarrow abs\left((Q - Q_p) \times open \times fee\ rate\right)$
   $M \leftarrow M - fee$
   $L \leftarrow QEp/M$

3. Calculate liquidation price
   $lp = liquidation\ price \leftarrow Ep(1 - \frac{1}{L})$

4. Determine Liquidation
   $If\ L > 0\ \&\ low < lp\ \ or\ \ L < 0\ \&\ high > lp$:
     $liquidation = True$
   $else$ :
     $liquidation = False$

5. Calculate $P$
   $if\ liquidation$ :
     $P \leftarrow 0$
   $else$ :
     $P \leftarrow M + Q(close - Ep)$

6. Calculate reward
   $reward \leftarrow f(P, P_p)$
```

The calculation above is based on two assumptions. First, we assume that we always trade on the start of the time interval, meaning that we always trade using opening price. Also, we assume that all the money is going to be used as margin, meaning that there is no left out money. The above calculation is repeated until termination condition is met. When we reach the end of our training data or when the total portfolio value becomes less than ε, a predetermined value, the process terminates, finishing one episode.

3.3. Agent

The agent should determine the optimal action given the current state. To accomplish this goal, we used DDPG algorithm to train actor and critic network. DDPG algorithm is chosen since the problem we are dealing with has continuous action space. We first build our critic and actor using neural network, consisting of fully connected layers with some dropouts. Then, we begin simulation using function Train. This function goes over the training data, making actions according to its actor network with some noise. The samples collected from this process is stored in buffer, which mitigates correlation between data and increases data efficiency. Periodically, with period T, we call function Update, which is where the network gets updated. The critic and actor networks are updated by gradient descent method, minimizing certain loss functions that is calculated from the samples of buffer. The target networks are updated at the end. The process of training is illustrated in Algorithm 1.

Algorithm 1 Training using DDPG

```
1:  function UPDATE($buffer, \phi^-, \theta^-, \phi, \theta, \tau, \gamma$)
2:      Sample minibatch from buffer

3:      $target = r + \gamma Q_{\phi^-}(ns, \mu_{\theta^-}(ns))$
4:      Critic Loss = $\frac{1}{N} \Sigma(Q_\phi(s, a) - target)^2$
5:      update critic network $\phi$ using gradient descent

6:      Actor Loss = $\frac{-1}{N} \Sigma\ Q_\phi(s, \mu_\theta(s))$
7:      update actor network $\theta$ using gradient descent

8:      $\phi^- \leftarrow \tau\phi + (1 - \tau)\phi^-$
9:      $\mu^- \leftarrow \tau\mu + (1 - \tau)\mu^-$
10: end function

11: function TRAIN($env, std, \tau, \gamma, T$)
12:     Initialize buffer
13:     Initialize target networks $\phi^-, \theta^-$
14:     Initialize critic, actor network $\phi, \theta$
15:     Freeze target networks: $\phi^-, \theta^-$
16:     $s$ =env.reset
17:     for $i = 1$ to $N$ do
18:         $a = \mu_\theta(s) + N(0, std^2)$
19:         $ns, r, done = env.step(s, a)$
20:         append $s, a, r, ns$ to buffer
21:         if $i \equiv 0(\mod T)$ then
22:             $update(buffer, \phi^-, \theta^-, \phi, \theta, \tau, \gamma)$
23:         end if
24:         s=ns
25:     end for
26: end function
```

4. Data preprocessing

We used BTC/USDT(BITCOIN / TETHER) perpetual 1 hour interval trading data. We retrieved open price, highest price, lowest price, closing price(OHLC) and volume data from Binance[6].
A total of 16103 data, from 2017 / 08 / 17 to 2022 / 06 / 04 were taken. The dataset was split into three parts. For training, 10000 data(2017-08-17-04 ~ 2021-09-22-17) were used. For validation, 1600 data(2021-09-22-18 ~ 2021-11-28-9) were used. Lastly, for testing, 4503 data (2021-11-28 10:00:00 AM~2022-06-04-12) were used. Furthermore, we split the test dataset into four smaller test datasets.
Next, we calculated the Moving Averages, Moving Average Convergence Divergence, and RSI based on the closing price data. $x(j)$ denotes the jth closing price.

- $Moving\ Average(MA)\ with\ window\ size\ w$

$$MA_w(i) = \sum_{j=i-w+1}^{j=i} x(j)$$

- $Relative\ Strength\ Index(RSI)$

$$\Delta x(i) = x(i) - x(i-1)$$
$$Ag(i) = 13Ag(i-1) + (\max(0, \Delta x(i)))/14$$
$$AL(i) = 13AL(i-1) + (\max(0, -\Delta x(i)))/14$$
$$RS(i) = \frac{Ag(i)}{AL(i)}, \qquad RSI(i) = 100 - \frac{100}{1 + RS(i)}$$

- $Moving\ Average\ Convergence\ Divergence(MACD)$

$$EMA_{12}(i) = \frac{11}{13}EMA_{12}(i-1) + \frac{2}{13}x(i)$$
$$EMA_{26}(i) = \frac{25}{27}EMA_{26}(i-1) + \frac{2}{27}x(i)$$
$$MACD(i) = EMA_{12}(i) - EMA_{26}(i)$$

Then, we normalized the data. We first normalize RSI and volume using min-max values, and normalize MACD using normal distribution.

$$RSI \leftarrow \frac{RSI - \min\{RSI(i)\}}{\max\{RSI(i)\} - \min\{RSI(i)\}}$$

$$Volume \leftarrow \frac{Volume - \min\{Volume(i)\}}{\max\{Volume(i)\} - \min\{Volume(i)\}}$$

$$MACD \leftarrow \frac{MACD - average\ of\ \{MACD(i)\}}{\text{standard deviation of}\ \{\text{MACD(i)}\}}$$

For OHLC price data and moving averages, we first find the smallest value T_{min} and the largest value $\mathrm{T_{max}}$ among them. Then we update every data X using the following equation.

$$X \leftarrow \frac{X - T_{min}}{\mathrm{T_{max}} - T_{min}}$$

This way of normalizing allows us to maintain the correlations and between price data and moving averages, which seems necessary.
We will train our agent using training dataset. The adequate hyperparameters will be determined based on the validation dataset. Lastly, the trained model will be tested using four test datasets.

5. Results

5.1. Determining reward function

For training and testing our agent, the maximum leverage value, L_{max} was set to 20. Also, the initial balance was set to 100 and the fee rate was set to 0.04%. The hyperparameters that were used in the simulation are summarized in **Table 1.**

Actor learning rate	1×10^{-5}
Critic learning rate	1×10^{-5}
Discount factor	0.99
Number of updates	10000×30
Batch size	256
Train interval	200
Buffer size	50000
Standard deviation of noise	5

Table 1 : Hyperparameters

We first set the reward function as
$$r = f(P, P_p) = (P - P_p)/P_p,$$
the rate of change of portfolio value, which is very natural and straightforward. After training our agent, its strategy was tested over the four test datasets. The results are shown in **Figure 1.**

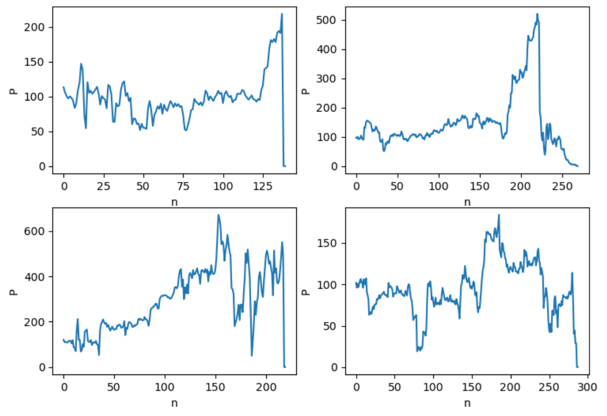

Figure 1 : Total portfolio values(test results)

For all the four cases, liquidation happened very quickly, and the agent lost all the money. This is caused because the current reward function $\left(\frac{P-P_p}{P_p}\right)$ has a lower bound of

-1, which is not small enough to express the effect of liquidation. For example, if the portfolio value doubles, the agent is rewarded +1. Therefore, if the agent doubles its portfolio value two times and gets liquidated, the total reward will still be +1, even though the agent had lost all the money. Therefore, the agent will be trained to use high leverage all the time only focusing on increasing portfolio value, not taking risk of liquidation into account.
Figure 2 shows us that this is true. The agent is extremely aggressive. It always uses the highest leverage it can use which is $L_{max} = 20$, and $-L_{max} = -20$.

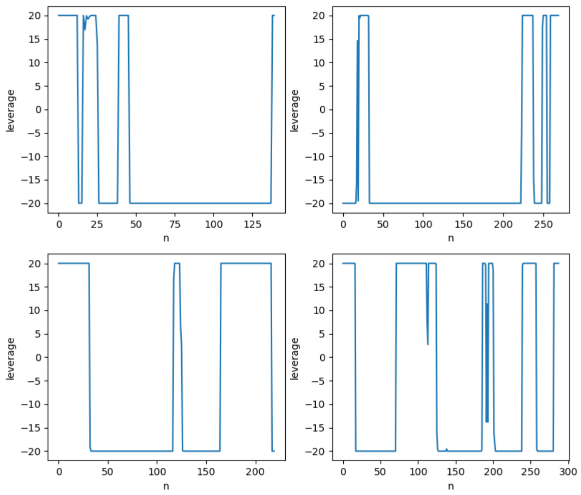

Figure 2 : Leverage used by the agent

The same phenomenon happened for the following reward function

$$r = f(P, P_p) = P - P_p$$

, the change in portfolio values, which is another intuitive and natural reward function. Because the reward function only focuses on the change of values, it could not emphasize the importance of liquidation as well. For example, the agent losing money from 200$ to 100$ and the agent losing money from 100$ to 0$, which is liquidation, returns same reward. Therefore, for the test dataset, the agent couldn't avoid being liquified.

5.2. Test results for modified agent

Therefore, we devised up the following reward function.

$$r = f(P, P_p) = \begin{cases} 0 & if\ P < \varepsilon\ and\ liquidation = False \\ \dfrac{P - P_p}{\max(\varepsilon, P)} & otherwise \end{cases}$$

Instead of dividing the change to the past value, we decided to divide it with the current portfolio value. This emphasizes the importance of risk since $r \to -\infty\ as\ P \to 0$, which is liquidation. For handling zero-division issues, we used $\max(\varepsilon, P)$ as the denominator of reward function, where ε is a constant satisfying $\varepsilon \ll 1$.

The agent was trained again with the same training data, and was evaluated using the same test data. The test results are shown in **Figure 3** and **Table 2.**

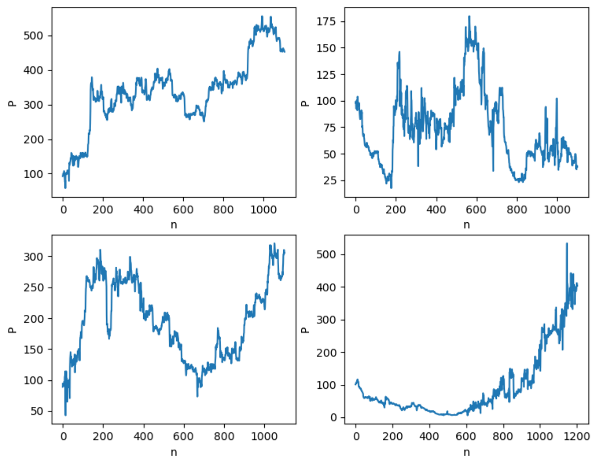

Figure 3 : Total portfolio values(test results) of modified agent

	Final portfolio value	Return
Test 1	452.2	+352.2%
Test 2	37.6	-62.4%
Test 3	305.2	+205.2%
Test 4	403.3	+303.3%

Table 2 : Final Portfolio values

As we can see, for all four test datasets, the agent was not liquidated until the end and the rate of return was 199.5% in average. The size of each test dataset was about 1200 trades, which is about 50 days since we used 1-hour interval price data. During this short amount of time, the agent could triple its asset. Also, the maximum change of the price of bitcoin was -53.85% during this period, which is far smaller than 199.5% in magnitude. This is a proof that the agent was able to exploit the leverage wisely. Lastly, **Figure 4** shows us the action(leverage value) that the agent had made. We can see that the agent doesn't use +20 or -20 all the time but uses leverage more carefully.

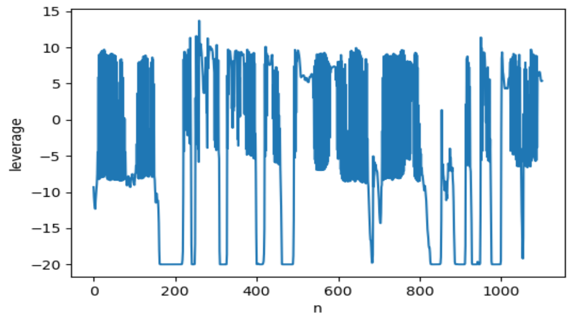

Figure 4 : Leverage used by modified agent

6. Conclusion

In this paper, we focused on margin trading environment, a unique environment where the trader can gain high profit under high risk. To examine the potential of deep reinforcement learning algorithms on this environment, DDPG algorithm was employed to train our trading agent. The results showed that choosing an appropriate reward function is vital and that it should be chosen differently from normal stock trading environments. By using a reward function that emphasizes the risk of liquidation, the agent obtained a profitable trading strategy that avoids liquidation.

References

[1] Sreelekshmy Selvin, R Vinayakumar, EA Gopalakrishnan, Vijay Krishna Menon, and KP Soman. Stock price prediction using lstm, rnn and cnnsliding window model. In 2017 international conference on advances in computing, communications and informatics (icacci), pages 1643–1647. IEEE, 2017..

[2] Yun-Cheng Tsai, Jun-Hao Chen, and Chun-Chieh Wang. Encoding candlesticks as images for patterns classification using convolutional neural networks. arXiv preprint arXiv:1901.05237, 2019.

[3] Zhang, Z., Zohren, S., & Roberts, S. (2020). Deep reinforcement learning for trading. The Journal of Financial Data Science, 2(2), 25-40.

[4] Xiong, Z., Liu, X. Y., Zhong, S., Yang, H., & Walid, A. (2018). Practical deep reinforcement learning approach for stock trading. arXiv preprint arXiv:1811.07522.

[5] Yang, H., Liu, X. Y., Zhong, S., & Walid, A. (2020, October). Deep reinforcement learning for automated stock trading: An ensemble strategy. In Proceedings of the First ACM International Conference on AI in Finance (pp. 1-8).

[6] Binance(2022), https://www.cryptodatadownload.com/data/binance/

[7] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In Neural Information Processing Systems 12, pages 1057–1063.

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[9] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014, January). Deterministic policy gradient algorithms. In International conference on machine learning (pp. 387-395). PMLR.