# Applying Different DRL Algorithms in Cryptocurrency Trading Environment

Chenggong Zhang
UID: 205837871
chenggong61@g.ucla.edu

Haocheng Zhang
UID:706071484
haochen235@g.ucla.edu

## Abstract

*In this project, we aim to create and test different Deep Reinforcement Learning Algorithms. We will test the multiple different DRL algorithms used to create the decision-make agents by applying them to Ethereum(ETH) Trading data. The ETH trading data include the opening and closing prices of Ethereum from 2018/5/15 to 2023/2/5, the highest and lowest prices reached, trading volume, and the timestamp of each trade. We investigate different types of Deep Reinforcement Learning Algorithms as well as neural network architectures to determine the performance of our various agents. Given the very difficult process of training agents, we were only able to make agents evaluate positive scores after 100 episodes, which means that trading-decision agents could make profits after 100 iterations.*

## 1. Introduction

The application of deep reinforcement learning algorithms(DRL) to automated cryptocurrency trading is a highly active area of research. Our objective as novice researchers are to gain a comprehensive understanding of the implementation of policy gradient methods and the modeling of the trading environment. Through the process of re-implementing existing modern RL algorithms, we aim to develop a methodology that will enable us to evaluate the performance of these algorithms and assess their potential for enhancing cryptocurrency trading strategies. Our ultimate goal is to contribute to the advancement of knowledge in this field and to gain insights that can be applied to real-world trading scenarios.

## 2. Methodology

### 2.1. Modelling Trading Environment

This project investigates application of RL in cryptocurrency trading on a centralized exchange, e.g., Binance. For simplicity, slippage and trading fee are neglected. We formulate the trading problem as a dynamic programming with continuous state space and continuous action space. We set $S_t = [\mathbf{f}_t, c_t, e_t] \in \mathbb{R}^{29}$, where $\mathbf{f}_t \in \mathbb{R}^{27}$, $c_t$ and $e_t \in \mathbb{R}$. $\mathbf{f}_t$ is features of market information at time step $t$. $c_t$ is the amount of US dollar the agent has at time step $t$, whereas $e_t$ is the amount of Ethereum the agent has at time step $t$. We incorporate basic market information as well as technical indicators in $\mathbf{f}_t$, as shown in Table 1. Action at time step $t$, $a_t \in [0, 1]$ denotes the proportion of Ethereum in portfolio. Based on the frictionless trading assumption, given $S_t$ and $a_t$, the transition is defined by 1 and 2 . Reward at time step $t$ is defined as $r_t = \alpha \log\left(\frac{c_{t+1}+p_{t+1}e_{t+1}}{c_t+p_te_t}\right)$. $\alpha$ is for scaling and it is set to be 100 herein.

$$e_{t+1} = a_t \frac{c_t}{p_t} + a_t e_t \tag{1}$$

$$c_{t+1} = (1 - a_t)(c_t + p_t e_t) \tag{2}$$

### 2.2. Reinforcement Learning Algorithms

There are multiple algorithms in deep reinforcement learning, In this project, we re-implement and applied DDPG [4], TD3 [2], SAC [3] in our ETH trading environment. There (A) is the architecture for our algorithms. Additionally, most of our codes are adopted and modified from the GitHub open source [1].

### 2.3. Experiment Setting

In this project, we adopt hourly data of Ethereum from Bitfinex from 2018/5/15 to 2022/08/16. Data during the period between 2018/5/15 and 2022/02/24 is used to train RL agents, and data during the period between 2018/02/24 and 2022/08/16 is used to evaluate the performance of RL agents. Three RL algorithms, i.e., DDPG, TD3, and SAC, are investigated in this project, and random policy is taken as the baseline. Random policy means action at each time step is taken randomly from the range between 0 and 1. To be rigorous, the random policy is run 100 times, and each RL agent is run 5 times repeatedly due to the limited computation resources we have. All RL agents are trained for 200 episodes.

Table 1. Features utilized to construct state

| Feature Name | Number |
|---|---|
| $\log(close/open)$ | 1 |
| $\log(high/open)$ | 1 |
| $\log(low/open)$ | 1 |
| hourly trading volume | 1 |
| Double Exp Moving Avg Price | 1 |
| Parabolic SAR/hourly open price | 1 |
| Average Directional Movement Index | 1 |
| Absolute Price Oscillator | 1 |
| Aroon Oscillator | 1 |
| Balance Of Power | 1 |
| Commodity Channel Index | 2 |
| Chande Momentum Oscillator | 1 |
| Directional Movement Index | 1 |
| Minus Directional Movement | 1 |
| Momentum | 1 |
| Plus Directional Movement | 1 |
| TRIX | 1 |
| Ultimate Oscillator | 1 |
| Stochastic Momentum Indicator | 3 |
| Stochastic Fast Momentum Indicator | 3 |
| Normalized Average True Range | 1 |
| True Range | 1 |
| Hilbert Transform - Dominant Cycle Period | 1 |
| Hilbert Transform - Dominant Cycle Phase | 1 |

## 3. Results

Fig 1 shows that all RL agents are able to beat the baseline, and SAC outperforms DDPG and TD3. It is noteworthy that the difference between DDPG and TD3 in their average score is not significant, though TD3 was proposed as an improved version of DDPG. However, the variance of TD3 is smaller than that of DDPG, indicating that TD3 has better training stability.

| | random policy | DDPG | TD3 | SAC |
|---|---|---|---|---|
| mean | -14.66 | 5.88 | 4.70 | 55.43 |
| std | 17.52 | 22.19 | 15.96 | 26.47 |

Table 2. mean and standard deviation of DDPG, TD3, SAC, and random policy.

## 4. Discussion

### 4.1. Agent performance

According to the experiment results, we find that RL agents are capable of earning money in a frictionless marketplace. Positive average scores of DDPG, TD3, and SAC indicate the predictability of the market trend of Ethereum. However, it is noteworthy that the frictionless trading environment is far from the real trading environment that trading agents face in practice. This means that the RL agents are very likely to fail in a real trading environment, which is not desired. Besides, it is also observed that standard deviations of scores obtained by all RL agents are relatively large. Though we only train each RL agent by 5 times, and the estimates of standard deviations are probably not accurate enough, the result can still support the conclusion that the training process of RL agents on this problem is rather unstable.

Besides, we also see that curves of average scores do not converge. This is probably because of the covariance shift or even the dynamic difference between the training set and validation set. Since our data is from the centralized exchange, which is efficient enough, we can hardly find an oracle to directly provide future market information. Therefore, we believe that algorithmic Ethereum trading is more challenging than we thought.

### 4.2. Future Work

This project adopts a rather strong assumption of frictionless trading. Under this assumption, it is easier to learn a reasonable trading policy, whereas, in a real marketplace, agents need to consider slippage and trading fee. The second direction that we can extend our work is to investigate trading policy on decentralized exchange rather than on traditional order books. Besides, we can extend this trading policy problem into a portfolio management problem by considering trading multiple risky assets. From the perspective of learning, we can add more features that can reflect further market tendencies.

## References


[1] Petros Christodoulou. Deep reinforcement learning algorithms with pytorch. https://github.com/p-christ/Deep-Reinforcement-Learning-Algorithms-with-PyTorch, 2021. 1

[2] Stephen Dankwa and Wenfeng Zheng. Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd international conference on vision, image and signal processing*, pages 1–5, 2019. 1

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018. 1

[4] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 1

## A. Architecture and Hyperparameter

These are three pseudocode algorithms for PPDG, TD3, and SAC under ETH Trading Environment. Basically, describe the initial step and the agent training process algorithm.

**Algorithm 1** Deep Deterministic Policy Gradient (DDPG) for ETH Trading

**Initialize:**
- Environment (ETH trading market)
- DDPG hyperparameters (actor_learning_rate, critic_learning_rate, gamma, tau, replay_buffer_size, batch_size, etc.)
- Actor network (policy) with parameters $\theta$
- Critic network (Q-value) with parameters $\phi$
- Target actor network with parameters $\theta'$
- Target critic network with parameters $\phi'$
- Experience replay buffer with capacity N
- Step counter $t = 0$

**procedure** TRAIN_DDPG_AGENT
  Observe initial state $s$
  **while** $t <$ max_steps **do**
    Get action $a$ from the actor network with exploration noise
    Execute action $a$ in the environment, observe next state $s'$, reward $r$, and done flag $d$
    Store transition $(s, a, r, s', d)$ in the replay buffer
    **if** replay buffer is large enough for a batch **then**
      Sample a batch of transitions from the replay buffer
      Compute target Q-values $y_i$ for each transition in the batch using target networks:
      $y_i = r_i + \gamma * (1 - d_i) * Q'(s'_i, \mu'(s'_i; \theta'); \phi')$
      Update the critic network by minimizing the loss:
      Critic loss = MSE$(Q(s_i, a_i; \phi), y_i)$
      Update the actor network using the policy gradient:
      Actor loss = $-\frac{1}{N} \sum_i Q(s_i, \mu(s_i; \theta); \phi)$
      Soft update target networks:
      $\theta' = \tau * \theta + (1 - \tau) * \theta'$
      $\phi' = \tau * \phi + (1 - \tau) * \phi'$
    **end if**
    Set state $s = s'$
    Increment step counter $t$
  **end while**
**end procedure**

**Algorithm 2** Twin Delayed Deep Deterministic (TD3) for ETH Trading

**Initialize:**
- Environment (ETH trading market)
- TD3 hyperparameters (actor_learning_rate, critic_learning_rate, gamma, tau, replay_buffer_size, batch_size, policy_noise, target_policy_noise, target_policy_noise_clip, etc.)
- Actor network (policy) with parameters $\theta$
- Two Critic networks (Q-value) with parameters $\phi_1$ and $\phi_2$
- Two target Critic networks with parameters $\phi'_1$ and $\phi'_2$
- Target actor network with parameters $\theta'$
- Experience replay buffer with capacity N
- Step counter $t = 0$

**procedure** TRAIN_TD3_AGENT
  Observe initial state $s$
  **while** $t <$ max_steps **do**
    Get action $a$ from the actor network with policy noise
    Execute action $a$ in the environment, observe next state $s'$, reward $r$, and done flag $d$
    Store transition $(s, a, r, s', d)$ in the replay buffer
    **if** replay buffer is large enough for a batch **then**
      Sample a batch of transitions from the replay buffer
      Compute target Q-values $y_i$ for each transition in the batch using target networks and clipping:
      $y_i = r_i + \gamma * (1 - d_i) * \min(Q'_1(s'_i, \mu'(s'_i; \theta'); \phi'_1), Q'_2(s'_i, \mu'(s'_i; \theta'); \phi'_2)) +$ policy_noise $* \epsilon_i$
      $\epsilon_i \sim$ clip$(\mathcal{N}(0, 1), -$target_policy_noise_clip, target_policy_n
      Update the critic networks by minimizing the loss:
      Critic loss = MSE$(Q_1(s_i, a_i; \phi_1), y_i) +$ MSE$(Q_2(s_i, a_i; \phi_2), y_i)$
      **if** $t$ mod 2 == 0 **then**
        Compute actor loss = $-\frac{1}{N} \sum_i Q_1(s_i, \mu(s_i; \theta); \phi_1)$
      **end if**
      Soft update target networks:
      $Q1' = \tau * Q1 + (1 - \tau) * Q1'$
      $Q2' = \tau * Q2 + (1 - \tau) * Q2'$
      $\theta' = \tau * \theta + (1 - \tau) * \theta'$
    **end if**
    Set state $s = s'$
    Increment step counter $t$
  **end while**
**end procedure**

**Algorithm 3** SAC Agent for ETH Trading

**Initialize:**
- Environment (ETH trading market)
- SAC hyperparameters (alpha, gamma, tau, actor_learning_rate, critic_learning_rate, batch_size, etc.)
- Actor network (policy) with parameters $\theta$
- Critic network (Q-value) with parameters $\phi$
- Target critic network with parameters $\phi'$
- Experience replay buffer with capacity N
- Step counter $t = 0$

**procedure** TRAIN_SAC_AGENT
  Observe initial state $s$
  **while** $t <$ max_steps **do**
    Get action $a$ by sampling from the actor network with temperature $\alpha$
    Execute action $a$ in the environment, observe next state $s'$, reward $r$, and done flag $d$
    Store transition $(s, a, r, s', d)$ in the replay buffer
    **if** replay buffer is large enough for a batch **then**
      Sample a batch of transitions from the replay buffer
      Compute target Q-values $y_i$ for each transition in the batch using target critic network:
      $y_i = r_i + \gamma * (1 - d_i) * (Q'(s'_i, \mu'(s'_i; \theta'); \phi') - \alpha * \log(\mu(s'_i; \theta')))$
      Update the critic network by minimizing the loss:
      Critic loss = MSE$(Q(s_i, a_i; \phi), y_i)$
      Update the actor network by minimizing the policy loss:
      Actor loss $= \frac{1}{N} \sum_i \alpha * \log(\mu(s_i; \theta)) - Q(s_i, \mu(s_i; \theta); \phi)$
      Soft update target networks:
      $\phi' = \tau * \phi + (1 - \tau) * \phi'$
    **end if**
    Set state $s = s'$
    Increment step counter $t$
  **end while**
**end procedure**

| Parameters | DDPG | TD3 | SAC |
|---|---|---|---|
| optimizer | Adam | Adam | Adam |
| learning rate | 0.0003 | 0.0003 | 0.0003 |
| hidden units (A, C) | $[64, 64]$ | $[20, 20]$ | $[64, 64]$ |
| internal activation (A, C) | ReLU | ReLU | ReLU |
| output activation (A) | Sigmoid | Sigmoid | Sigmoid |
| output activation (C) | None | None | None |
| $\tau$ for target network | 0.05 | 0.05 | 0.05 |
| gradient clip norm | 0.5 | 0.5 | 0.5 |
| buffer size | $10^6$ | $10^6$ | $10^6$ |
| batch size | 256 | 256 | 256 |
| discount factor | 0.99 | 0.99 | 0.99 |
| action noise | $\mathcal{N}(0, 0.1^2)$ | - | $\mathcal{N}(0, 0.1^2)$ |
| action noise range | $[-0.2, 0.2]$ | $[-0.2, 0.2]$ | $[-0.2, 0.2]$ |
| steps before learn | 1000 | 1000 | 1000 |

Table 3. Hyperparameters of DDPG, TD3, and SAC

## B. Results and Performance

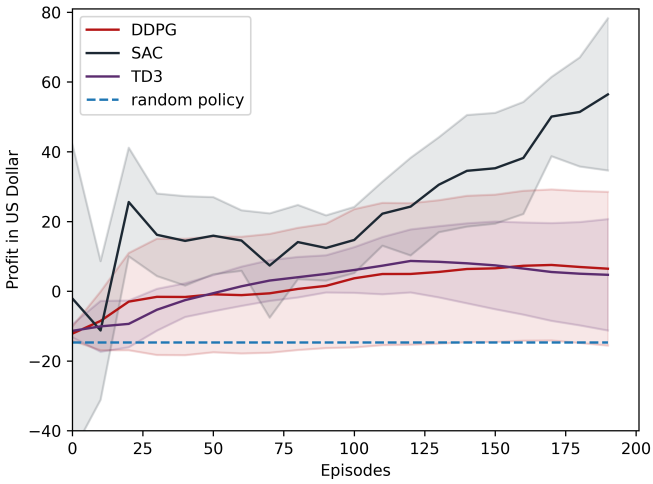


Figure 1. Moving average results of total reward in one episode obtained by DDPG, TD3, SAC, and the baseline