CP3106: A Study of Cryptocurrency Investment with Dollar Cost Averaging

Reinforcement Learning for Optimal Stopping Problem

Wang Shanmu, *Supervised by Professor He Bingsheng*

Department of Computer Science, National University of Singapore

Table of contents

1. Introduction
2. Back-testing for Dollar Cost Averaging
3. Optimal Stopping Problem and Reinforcement Learning Methods
4. Environment for Reinforcement Learning
5. Experiment Setup
6. Empirical Results
7. Conclusion

Introduction

Cryptocurrency: Overview

A CRYPTOCURRENCY is a digital or virtual currency that is secured by cryptography.

Technology

- Blockchain
- Digital Signature
- Smart Contract
- ...

Feature

- Trade Anytime
- Trade Anywhere
- Rapid Growth
- ...

Cryptocurrency: Rapid Growth

The rapid growth of the cryptocurrency market cap has attracted a lot of financial institutions as well as individual investors

Figure 1: The total cryptocurrency market cap [3]

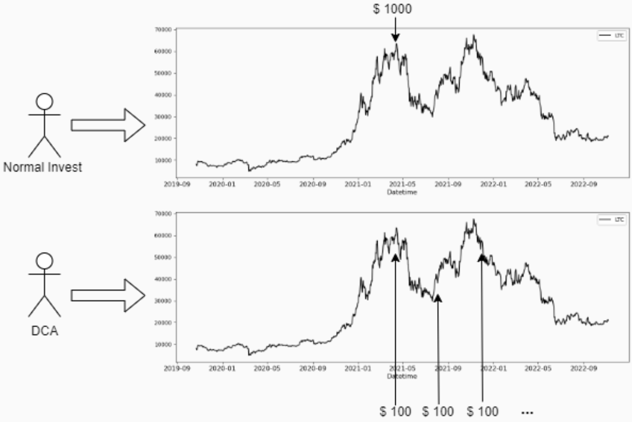
Cryptocurrencies are considered more **volatile** than stocks

Figure 2: The price change of BTC during Oct. 2022 [3]

Greater money-earning opportunities, but also more risks!

Dollar Cost Averaging

Dollar Cost Averaging (DCA) requires an investor to invest the same amount of money at regular intervals, typically weekly, monthly or quarterly, which has the potential to mitigate timing risk.



Dollar Cost Averaging: Further Improvement

- If the investor can invest on the day with the lowest price during each investment interval, they can end up with more shares at the end and are likely to make more profits

Dollar Cost Averaging: Further Improvement

- If the investor can invest on the day with the lowest price during each investment interval, they can end up with more shares at the end and are likely to make more profits
- In practice, an investor can see the current price and previous prices and decide whether to invest today

Dollar Cost Averaging: Further Improvement

- If the investor can invest on the day with the lowest price during each investment interval, they can end up with more shares at the end and are likely to make more profits
- In practice, an investor can see the current price and previous prices and decide whether to invest today
- This sequentially decision making can be formulated as an **Optimal Stopping question**

Dollar Cost Averaging: Further Improvement

- If the investor can invest on the day with the lowest price during each investment interval, they can end up with more shares at the end and are likely to make more profits
- In practice, an investor can see the current price and previous prices and decide whether to invest today
- This sequentially decision making can be formulated as an **Optimal Stopping question**
- We can apply **Reinforcement Learning** (RL) methods in such a dynamic decision-making process.

Presentation Structure

The remainder of this presentation is organized as follows:

1. show the back-testing results on cryptocurrency with DCA
2. formalize the optimal stopping problem and introduce the reinforcement learning algorithms
3. show how we build up the RL environment for experiments and elaborate the experiment setup
4. give the empirical results and conclusions

Back-testing for Dollar Cost Averaging

Back-testing for DCA: Portfolios for Cryptocurrencies

To demonstrate the benefits of Dollar Cost Averaging, we back-tested DCA method on different cryptocurrencies and investment portfolios:

- Bitcoin (BTC): Simply invests in Bitcoin
- Ethereum (ETH): Simply invests in Ethereum
- Bitwise 10 Crypto Index Fund (Index-10) [1]: tracks an index of the 10 largest crypto assets and weights the assets by market capitalization
- Max-Ratio: Similar to Index-10, but sets a limitation of 20% to the weight of each cryptocurrency.

Back-testing Toolkit

ShowEffectiveness generates the starting date list and then passes each date in this list, as well as the year and the interval, to the strategy function

AtomStrategy and *IndexStrategy* will simulate investing in an Interval from the StartDate for Year, and output the Return / Investment Ratio at the end.

Function	Desicirition
ShowEffectiveness	Generate the starting date list and call the strategy function
AtomStrategy	The strategy of buying a single cryptocurrency
IndexStrategy	The strategy of buying multiple crypto-currencies based on market capitalization

Back-testing for DCA: Violin Plots

Investments with a starting date after August 2018: The longer the investment, the higher the probability of earning more

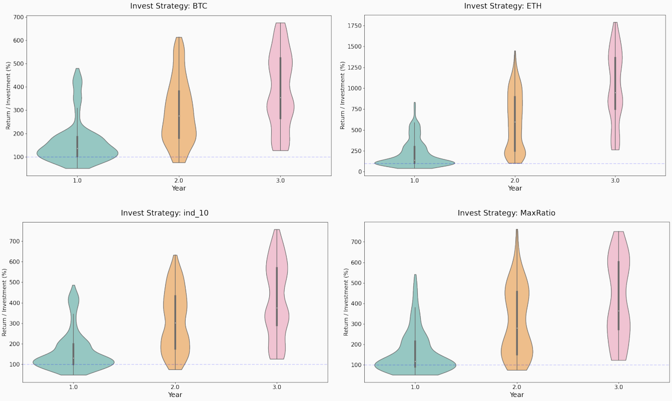


Figure 3: Distribution of the Return / Investment for different years

Back-testing for DCA: Statistics

Larger than		100%	150%	200%	250%	300%
Year 1	BTC	76.4%	41.3%	18.2%	13.1%	11.1%
	ETH	78.5%	48.3%	39.8%	32.2%	24.4%
	Index-10	75.2%	38.4%	24.5%	14.3%	11.1%
	Max-Ratio	63.5%	35.2%	26.9%	16.0%	11.4%
Year 2	BTC	93.4%	88.6%	71.5%	56.8%	44.7%
	ETH	100%	93.4%	88.5%	74.9%	70.8%
	Index-10	93.0%	84.0%	71.9%	56.6%	50.1%
	Max-Ratio	93.2%	74.7%	67.1%	52.7%	47.9%
Year 3	BTC	100%	88.5%	81.4%	77.3%	69.9%
	ETH	100%	100%	100%	100%	91.8%
	Index-10	100%	90.1%	85.2%	77.5%	73.2%
	Max-Ratio	100%	87.7%	78.1%	78.1%	68.5%

Back-testing for DCA: Summary

Our back-testing results have shown that:

- Dollar Cost Averaging strategy has the ability to mitigate timing risk, making it a suitable strategy for cryptocurrency
- To further eliminate the risk, one can also choose to start the round at different times of a year to average risk or simply extend the investment

Further Improvement:

- Recall: we want to develop an agent that is able to choose a day with a lower price during each investment cycle and purchase the crypto asset on that day

Optimal Stopping Problem and Reinforcement Learning Methods

Optimal Stopping: Definition

The optimal stopping problem consists in finding the optimal time to stop in order to maximize an expected reward. In this work, we adopt the following notations:

- $\mathcal{A}_t$: the set of possible actions at time t
- S : the set of all possible states
- T : the horizon of the problem, i.e., the investment cycle in our scenario
- s_t : the state at time t
- π : a policy map S to $\mathcal{A}$,$\pi : S \to \mathcal{A}$, i.e., $\pi(s_t) = a_t$
- $s_{0:T}$: a trajectory $[s_0, \ldots, s_T]$
- U_t : the payout received when stopping at time t after observing $s_{0:t}$

Cont'd

Specifically, $\mathcal{A}_t := \{hold, buy\}$ for $t < T$, and $\mathcal{A}_T := \{buy\}$. The stopping time with policy π is defined as:

$$\tau_\pi = min\{t \in [0, \ldots, T] \; s.t. \; \pi(s_t) = stop\}$$

An optimal policy π^* should be able to do the following decisions:

$$\pi^*(s_t) = \begin{cases} \text{buy} & \text{if } \mathbb{E}\left[U_t \mid s_{0:t}\right] \geq \mathbb{E}\left[U_{\tau_\pi^{t+1}} \mid s_{0:t}\right] \\ \text{hold} & \text{otherwise} \end{cases}$$

, where $\tau_\pi^t = min\{t' \in [t, \ldots, T, \; s.t, \; \pi(s_{t'}) = buy\}$.

In a word, if the payoff of stopping at current time is greater than the maximum payoff after current day, $\pi^*(s_t) = buy$, otherwise $\pi^*(s_t) = hold$.

Optimal Stopping: A RL Perspective

The sequential decision making problem can be seen as an analogy of a standard reinforcement learning notation:

$$Q(s,a) = \begin{cases} r(s,\ \text{buy}), & \text{if } a = \text{buy} \\ r(s,\ \text{hold}) + \gamma \mathbb{E}\left[\max\left(Q\left(s',\ \text{buy}\right), Q\left(s',\ \text{hold}\right)\right)\right], & \text{otherwise} \end{cases}$$

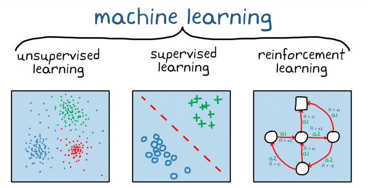
In this notation, $Q(s,a)$ refers to the Quality of action a under state s, $r(s,a)$ refers to the reward by taking action a under state s, s' is the next state, and γ is the discounted factor.

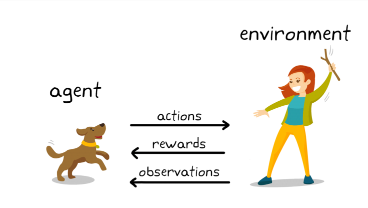
A policy for such an optimal stopping problem is to choose the action depending on which action has a higher Q-value.

Reinforcement Learning: Basic Idea

Reinforcement Learning is a branch of machine learning.

The basic idea of reinforcement learning is an agent learning to take action in the environment to maximize a reward signal by the feedback from the environment.





At each discrete time step t, the agent observe the state s_t provided by the environment, and do the action a_t based on its strategy, and then the environment provides the reward r_t for this action and next state s_{t+1}.

Reinforcement Learning: Deep Q-Learning

Deep Q-learning uses a neural network to approximate the action values for a given state s, i.e, $Q(s_t, a) = NN(s_t; \theta)$.

ϵ-greedily: At each step, the agent choose a random action with probability ϵ; otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$

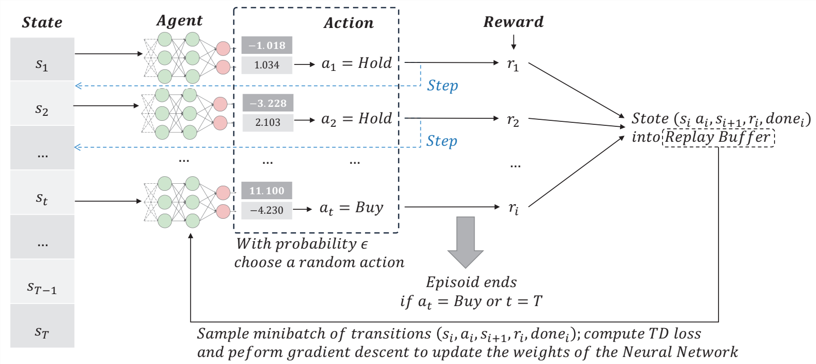
Replay Buffer: after an action, adds a transition $(s_t, a_t, r_t, s_{t+1}, done_t)$ to a replay buffer, where $done_t$ indicates whether current episode ends.

TD (temporal difference) loss: $\delta_t(\theta) = \left[(y_t - Q(s_t, a_t; \theta))^2\right]$, where

$$y_j = \begin{cases} r_t & \text{if } done_t \\ r_t + \gamma \max_{a'} \bar{Q}(s_{t+1}, a'; \bar{\theta}) & \text{otherwise.} \end{cases}$$

At each step, we sample a mini-batches uniformly from the replay buffer. The parameters θ of the neural network are optimized by using gradient descent to minimize the loss.

Workflow of the Basic DQN



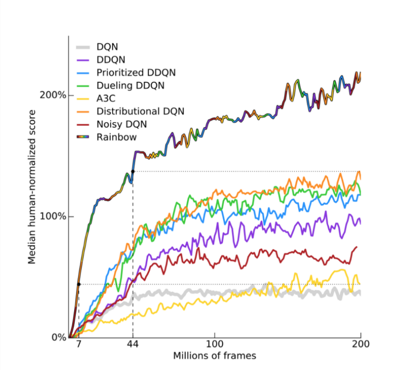
Note: The rewards from the episode can be summed up to get a score of this episode, which can be used as an important indicator of the agent's training process.

Extensions to DQN:

However, there are some problems with the basic DQN algorithm, including overestimation, long training time and so on.

Actually, many researchers have proposed a lot of improvements to the DQN algorithm:

- Double Q-learning
- Prioritized replay
- Dueling networks
- Multi-step learning
- Distributional RL
- Noisy Nets
- ...



Extensions to DQN: Double DQN

Problem with the original DQN: Overestimation.

In the basic DQN, we do the action selection and evaluation basically with the same network!

Example: consider a single state s where the true Q value for all actions equal 0, but the estimated Q values are distributed some above and below zero. Taking the maximum of these estimates (which is obviously bigger than zero) to update the Q function leads to the overestimation of Q values.

Problem with the original DQN: Overestimation.

In the basic DQN, we do the action selection and evaluation basically with the same network!

Double DQN [8] uses the second set of weights θ' to fairly evaluate the value of the action selected with the network of weights θ.

- This second set of weights can be updated symmetrically by switching the roles of θ and θ'
- It is like 2 function approximators aggregating on each other's choice of best action

Extensions to DQN: Prioritized Replay

Basic DQN samples mini-batch uniformly from the replay buffer, but some experiences are more important

Prioritized Replay [6] gives more priority to important experiences based on TD error $|\delta_i|$, so that the algorithm can sample more frequently those transitions that are much to learn

To be more specific:

- the priority of transition i: $p_i = |\delta_i| + \epsilon$
- the probability of sampling transition i as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha},$$

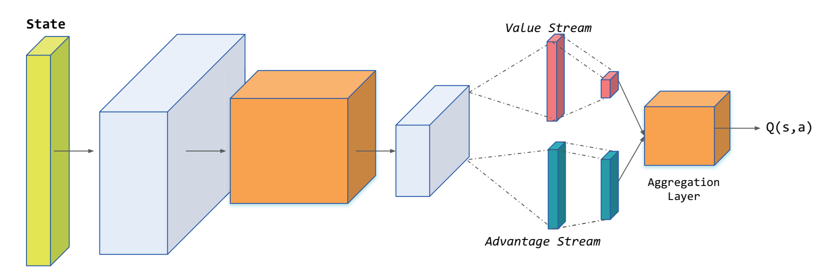
- importance-sampling (IS) weights:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$$

Extensions to DQN: Dueling networks

Dueling networks [9] explicitly separates the representation of state values and action advantages

We want to know that is the high Q value due to this action itself, or is it because any action in this state has a high Q value



Extensions to DQN: Multi-step Learning

Multi-step learning [7] proposed to use n-step return rather than using 1-step return to calculate Q values

so that the target value does not rely on just the current reward and can be more accurate.

Define the truncated n-step return from a given state s_t as

$$r_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} r_{t+k+1}.$$

A multi-step variant of DQN is then defined by minimizing the alternative loss:

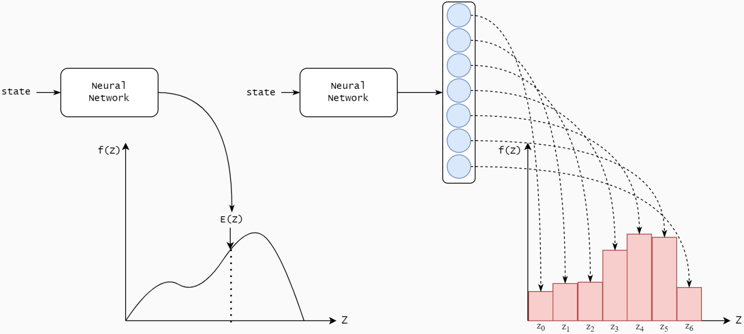
$$\left(r_t^{(n)} + \gamma_t^{(n)} \max_{a'} Q\left(s_{t+n}, a'; \bar{\theta}\right) - Q\left(s_t, a_t; \theta\right) \right)^2$$

Extensions to DQN: Distributional RL

Distributional RL [2] introduced to learn to approximate the distribution of returns instead of the expected return.

Given policy π, the return is a random variable Z. We can model the value distribution using a set of atoms:

$\{z_i = V_{\text{MIN}} + i\triangle z : 0 \leq i < N\}, \triangle z := \frac{V_{MAX} - V_{MIN}}{N-1}$

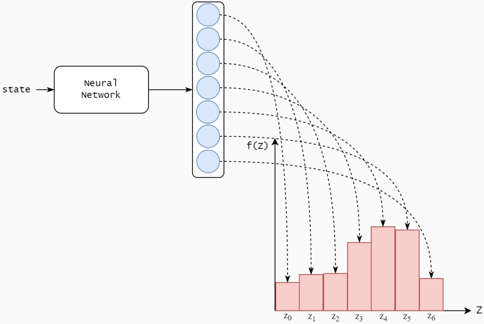


Extensions to DQN: Distributional RL

A *softmax* is applied independently for each action dimension of the output so that the output p_i is the probability of z_i.

To estimate Q-values, we can use inner product of each action's softmax distribution and the set of atoms $\{z_i\}$:

$$Q(s_t, a_t) = \sum_i z_i p_i(s_t, a_t),$$



Extensions to DQN: Noisy Net

Noisy Nets [4] were introduced to add noise to the network parameters so that it explores more.

A normal linear layer of a neural network is represented by $y = wx + b$, where $x \in \mathbb{R}^p$ is the layer input, $w \in \mathbb{R}^{q \times p}$, and $b \in \mathbb{R}$ the bias. Then the corresponding noisy linear layer is defined as:

$$y = (\mu^w + \sigma^w \odot \epsilon^w)x + \mu^b + \sigma^b \odot \epsilon^b,$$

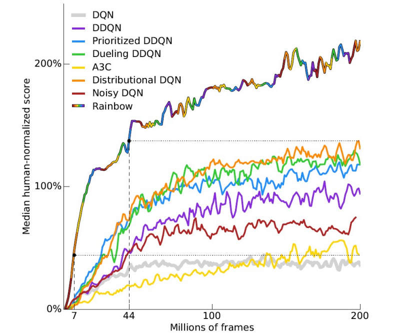
where $\mu^w + \sigma^w \odot \epsilon^w$ and $\mu^b + \sigma^b \odot \epsilon^b$ replace w and b

The parameters μ^w, μ^b, σ^w and σ^b are learnable, whereas ϵ^w and ϵ^b are noise random variables.

Over time, the network learns to ignore the noisy stream at different rates in different parts of the state space, allowing state-conditional exploration

Combination of the extensions: Rainbow DQN

The Rainbow DQN [5] makes a combination of the aforementioned extensions and provides very good performance in many missions.



In our project, we add all of these extensions to the basic DQN:

- Network Structure: Dueling + Distributional + Noisy layer
- Training: Double DQN
- Use Prioritized replay buffer
- Use multi-step learning

Environment for Reinforcement Learning

State

We adopt a price-based state with the form of:

$$s_t = [\textit{remaining time}, \textit{relative price}, \textit{window prices}]$$

On day t, we obtain the state s_t as follows:

1. **remaining time:** $(T - t)/T$, where T is the investment cycle and t indicates which day are we in this cycle.
2. **relative value:** we first select the price on the day before the start of the current investment cycle as reference price p_r, and then obtain today's price p_t. The relative value:

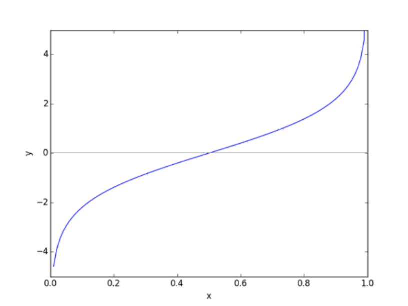
   $$\textit{relative price} = \textit{sigmoid}(p_t - p_r)$$

3. **window price:** window price is a list of price data for the previous t_{window} days from today. And a Max-Min normalization is applied so that the price data should range from 0 to 1.

Reward

We want to reward our agent if it can get a low price to invest or hold our money when faced with high price, and punish the agent if it do the opposite

Our reward scheme is based on the *logit* function $f(x) = ln\frac{x}{1-x}$, which is actually the inverse of sigmoid function.



Reward

In our scenario, the reward scheme is as follows:

1. Get the prices of the current investment cycle, and normalize the prices to 0 to 1. Let p_i be the normalized price of $i - th$ day.
2. If the agent decide to buy on the $i - th$ day or i equals to investment cycle T, we calculate the reward through:

$$r_i = -f(p_i)$$

3. If the agent decide to hold on the $i - th$ day, we calculate the reward through:

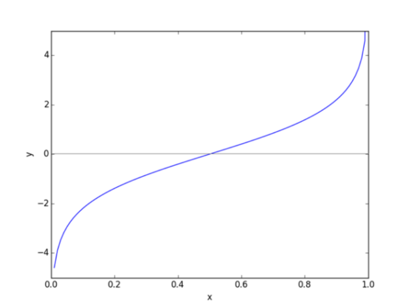
$$r_i = 0.5f(p_i)$$

, where 0.5 is to prevent dilution of the final reward

Reward

Hold: $r = 0.5f(p)$

Buy: $r = -f(p)$



Experiment Setup

Hyperparameters

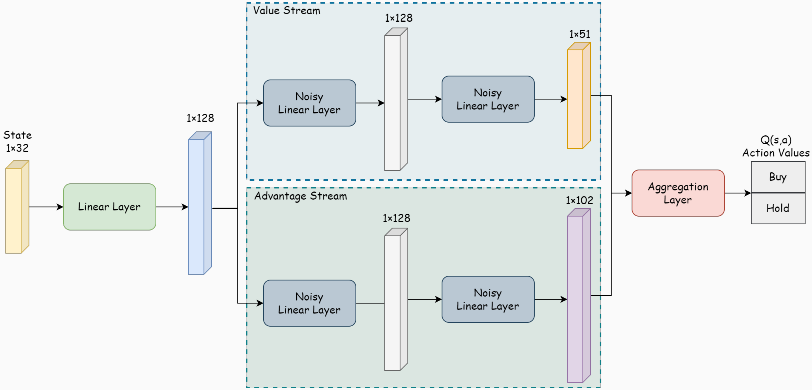
Hyperparameters		Value
Environment Parameters	t_{window}	30
	T_{cycle}	9
Basic Parameters	learning rate	5×10^{-4}
	memory size	10000
	batch size	128
	target update step U	100
	discounted factor γ	0.95
Prioritized Experience Replay	α	0.2
	β	0.6
	ϵ	1×10^{-6}
Distributional RL	V_{MIN}	0
	V_{MAX}	20
	atom size	51
N-step Learning	n step	3

Network Structure

The state input is a $(1 \times t_{windows} + 2)$, i.e, 1×32 vector

The output of the value stream is a 1×51 vector

The output of the advantage stream is a 1×102 vector



Data Splitting

Cryptocurrencies have a short history of being widely traded, so that there is a limitation to our data!

we traced back 3472 days of Bitcoin's price from 2013.05.06 to 2022.11.07 and 2649 days of Ethereum's price from 2015.08.07 to 2022.11.07.

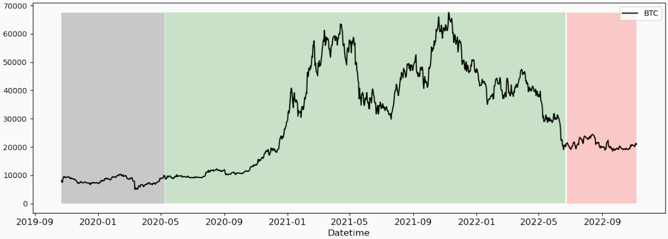
However, early price changes are too old to be a good reference and even might harm the training if involved

so those early price data are discarded. The rest of the data is split into training and test data at a ratio of 0.85 to 0.15

Data Splitting for BTC

We split the price data of Bitcoin as follows:

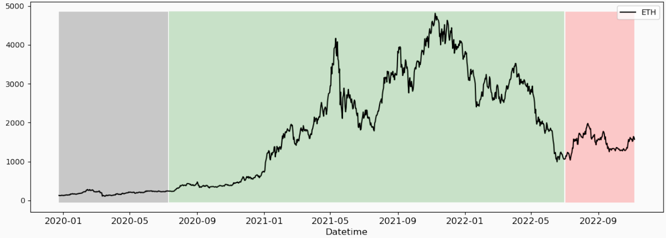
1. **Training:** from 2020.05.09 to 2022.06.23, 775 days
2. **Testing:** from 2022.06.24 to 2022.11.07, 136 days



Data Splitting for ETH

Similarly, we split the price data of Ethereum as follows:

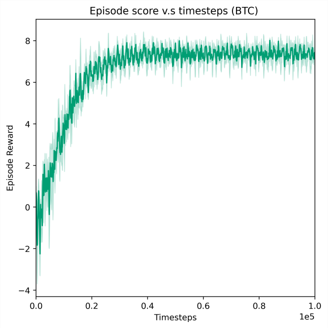
1. **Training:** from 2020.07.11 to 2022.07.02, 721 days
2. **Testing:** from 2022.07.03 to 2022.11.07, 127 days

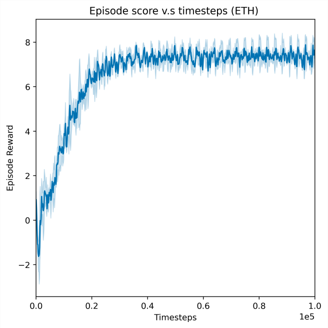


Empirical Results

Training Results

We train our Rainbow DQN agent for 100,000 steps. Curves are smoothed with a moving average over 10 points



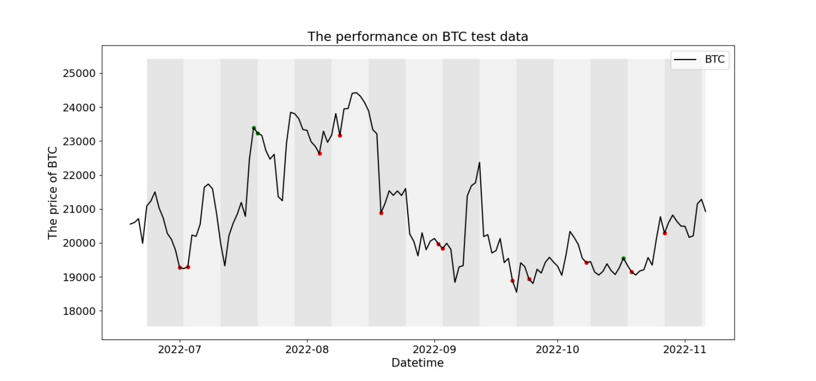


The volatility of score in BTC is larger than that in ETH environment, which could be due to the greater range of price fluctuations in BTC.

Testing Results: on BTC

Assume starting to invest on the first day of the test period.

Among 15 investments during the test period, there are 12 times we buy with a price lower than the average price.



Testing Results: on BTC

Starting on the first day and following our trained agent, the amount we end up earning after 15 cycles of investment, compared to *always investing on the first* or *last day* or *investing at the average price* (hypothetically):

Compared with	Improvement
Buy on first day	2.29%
Buy on last day	2.92%
Buy on random day	2.64%
Buy on average price	1.65%

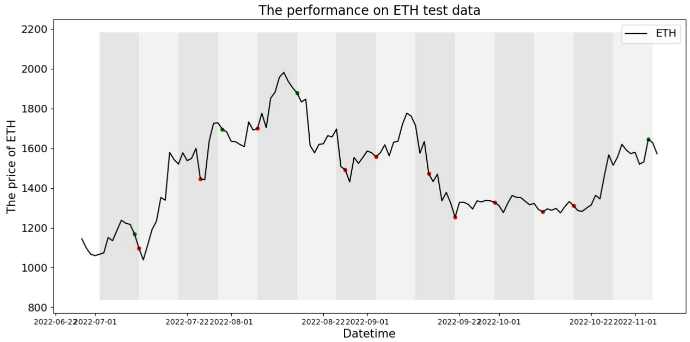
Testing Results: on BTC

We then generate all the possible episodes in test period, i.e, all 9 consecutive days in the test period.

For each episode, we compute the improvement of our agent compared with buying on first day, last day, random day, and buy with average price. The average of the improvements over all the episodes:

Compared with	Improvement
Buy on first day	2.30%
Buy on last day	2.25%
Buy on random day	1.92%
Buy on average price	2.25%

On the test data of ETH, among 14 investments, there are 10 times we buy with a price lower than the average price.



Testing Results: on ETH

Starting on the first day and following our trained agent, the amount we end up earning after 14 cycles of investment, compared with:

Compared with	Improvement
Buy on first day	1.16%
Buy on last day	6.47%
Buy on random day	2.98%
Buy on average price	2.83%

The average of the improvements over all the possible episodes in test period:

Compared with	Improvement
Buy on first day	1.22%
Buy on last day	6.71%
Buy on random day	3.89%
Buy on average price	4.71%

Testing with Different Investment Cycles

Recall: The remaining time is normalized, so it should be possible to conduct some experiments to test our trained agent's performance with different investment cycles.

Test set	Compared with	$T = 9$	$T = 5$	$T = 6$	$T = 7$	$T = 8$
BTC Test	Buying on first day	2.29%	1.09%	0.66%	1.46%	1.03%
	Buying on last day	2.92%	0.77%	0.77%	0.74%	1.85%
	Buying on random day	2.64%	0.95%	0.76%	0.74%	1.87%
	Buying on average price	1.65%	1.10%	1.26%	0.90%	1.05%
ETH Test	Buying on first day	1.16%	-0.55%	-1.08%	-0.48%	-1.06%
	Buying on last day	6.47%	1.72%	2.38%	1.76%	2.28%
	Buying on random day	2.98%	0.62%	-1.71%	-0.70%	0.70%
	Buying on average price	2.83%	0.90%	0.43%	0.41%	1.04%

Cont'd

Test set	Compared with	$T = 10$	$T = 11$	$T = 12$	$T = 13$	$T = 14$
BTC Test	Buying on first day	1.77%	0.42%	0.60%	4.47%	1.90%
	Buying on last day	1.45%	0.18%	0.64%	2.47%	1.37%
	Buying on random day	4.84%	2.50%	-0.97%	1.18%	0.20%
	Buying on average price	2.42%	2.10%	1.06%	2.40%	0.38%
ETH Test	Buying on first day	-2.33%	-0.49%	-0.34%	-2.36%	-0.58%
	Buying on last day	2.47%	3.56%	4.94%	1.36%	3.68%
	Buying on random day	2.00%	5.00%	4.13%	3.00%	3.36%
	Buying on average price	1.62%	3.67%	3.85%	0.19%	1.35%

Although the performance is not as good as the performance when $T = 9$, our agents still earn more profits compared with investing at the average price of each cycle on average.

Summary of the Testing Results

- We have shown the effectiveness of reinforcement learning methods for solving optimal stopping problems
- We demonstrate that our method is able to help investors to find a lower price to invest during each investment cycle when they are following the dollar cost averaging strategy, and thus help them to grab more profits
- Combined with our RL method, DCA can be a strategy that can not only eliminate the timing risk but also has a higher probability of earning profits

Conclusion

Summary

1. In this project, we have done back-testing experiments of cryptocurrency investment with the dollar cost averaging method, demonstrating that this method has the ability to mitigate timing risk in cryptocurrency markets
2. We formalized the problem of finding a day with a lower price to invest as an optimal stopping problem and implemented a Rainbow DQN agent to solve this problem and conduct experiments based on Bitcoin and Ethereum price data
3. We trained and tested the agent on the historical price data, and tested the performance of our trained agents under different investment cycles, each of which shows a decent performance
4. This work will be implemented into *Kreek*, a startup by our research group

Thank You for Listening!
Any Questions?

Bitw | bitwise 10 crypto index fund.

M. G. Bellemare, W. Dabney, and R. Munos.
A distributional perspective on reinforcement learning.
In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

Coinmarketcap.
Cryptocurrency market capitalizations | coinmarketcap, 2022.

M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, et al.
Noisy networks for exploration.
arXiv preprint arXiv:1706.10295, 2017.

M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver.
Rainbow: Combining improvements in deep reinforcement learning.
In *Thirty-second AAAI conference on artificial intelligence*, 2018.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver.
Prioritized experience replay.
arXiv preprint arXiv:1511.05952, 2015.

R. S. Sutton and A. G. Barto.
Reinforcement learning: An introduction.
MIT press, 2018.

H. Van Hasselt, A. Guez, and D. Silver.
Deep reinforcement learning with double q-learning.
In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas.
Dueling network architectures for deep reinforcement learning.
In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.