Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Hierarchical Reinforcement Learning for Algorithmic Trading

Semester Thesis

Pierre Motard

pmotard@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Zhao Meng
Prof. Dr. Roger Wattenhofer

March 4, 2022

Acknowledgements

I want to thank Yunpu Ma and Zhao Meng for their active involvement in challenging my ideas and proposing interesting approaches as well as Professor Wattenhofer for making this project possible in the Distributed Computing department. I would also like to thank my family for the constant support in all my undertakings.

Abstract

The cryptocurrency market is growing at an unprecedented rate, characterized by much more emotionally driven and non rational behaviors with 'hodlers' as well as high volatility exposing investors to high risks. Different statistical and machine learning techniques have been applied for automated trading, and more recently reinforcement learning. This work implements a hierarchical reinforcement learning agent that decomposes the order execution into different levels allowing for a more structured exploration of each sub-action as well as reducing the state space. The model demonstrates that it is capable of making profits in both simplified and low dimensional trading environment as well as in a realistic environment involving bid-ask spread and fees.

Contents

Chapter 1

Introduction

1.1 Motivation for reinforcement learning

The use of automatic trading strategies has intensified since more data have been made available to the wider public. Existing statistical and machine learning approaches [1] are based on identifying and extracting significant signals, followed by building a trading strategy based on these signals to maximize profit. Common trading strategies rely on a recurring pattern in the data, called an "edge", which acts as an indicator for taking a position in the market. These edges are increasingly difficult to find and never last long. Traditional machine learning approaches are therefore limited in their interactivity with the environment. Models are learned and parameterized after cross-validation over different periods, then backtested to measure performance. Reinforcement learning allows to directly interact with a stochastic environment [2] to learn an optimal policy for the agent in order to automate decision making and maximize a long term objective. The main advantage of reinforcement learning is the feedback of a reward signal that can be customized to the desired strategy.

1.2 Motivation for cryptocurrency trading strategy

The trading strategy used in this work is called trend following, it is also used by humans making it easily understandable. It consists of identifying a trend in the market using different market characteristics, as well as indicators, and taking a position by following this trend. These trends most often come from emotional drivers. Because the cryptocurrency market is much less controlled by high-frequency trading systems and open to the general public, it is more likely to incorporate predictable trends and opportunities. The cryptocurrency market remains less efficient overall, meaning that historical data can potentially lead to informative predictions about future prices, although its efficiency is increasing rapidly over time [3].

In addition, since the cryptocurrency market is prone to high volatility, it is

possible to achieve larger gains and presents itself as an opportunity to rely on agents learning safer strategies.

1.3 Contribution

In this project, I explore trend following trading strategies in light of a reinforcement learning framework. Using two environments, from a simpler low-dimensional environment using basic financial data to a more complex and realistic environment using limit order book data from [4]. Both environments are presented in section 3. I implement a hierarchical reinforcement learning (HRL) agent presented in section 4: the TradeR agent [5]. It decomposes its order execution by first deciding on an amount and then on the position to take (buy, sell or hold), thus reducing the dimensionality of each action.

Chapter 2

Background

In this section, I present the types of financial data that will compose the environments. There are currently no standards in the financial data to be used but rather different levels of complexity and granularity.

2.1 OHLCV data

The first simple way to capture the overall state of a financial asset at a given time is the OHLCV data point. Open, High, Low, Close, Volume (OHLCV) data is a compact way to represent the information flow of the financial market information. "Open" is the price of the asset at the beginning of that period and "Close" at the end. "High" is the highest price during that period and "Low" is the lowest. "Volume" quantifies the volume traded during this period. An example of an OHLCV data point is shown in figure 2.1.

2.2 Limit order book data

Another much finer source of market trading data is the limit order book, which contains the status of bids and offers being offered in the market at a given time. It can be seen as two stacks, one for each side, representing the bids, orders from traders offering to buy an asset, and asks, orders from traders offering to sell their asset. When the best (higher) bid is lower than the best (lower) ask offer, an arbitrage opportunity exists. The difference between these two prices is called the bid-ask spread [6]. Figure 2.2 from [7] illustrates a limit order book of depth 10 because it contains 10 price and quantity levels. Both sides are shown here in red for asks and blue for bids.

The bottom red line represents the best ask with a price of 8711.93, so the lowest price at which someone is willing to sell. That is, the lowest price at which you can buy BTC. However, the quantity is only 0.01 BTC, which means that if you buy more, you consume this first line and move directly to the next line,

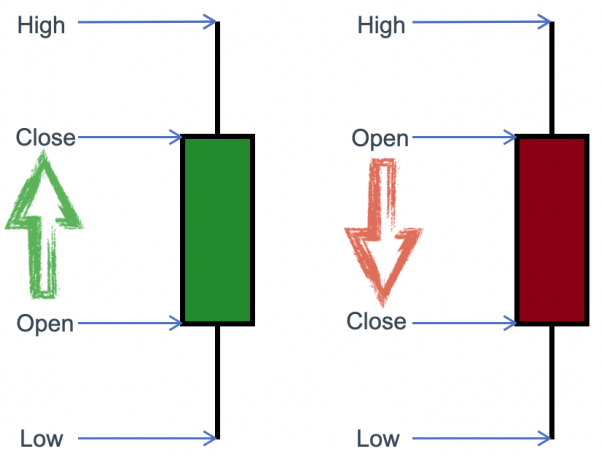

Figure 2.1: Diagram illustrating the OHLCV data.

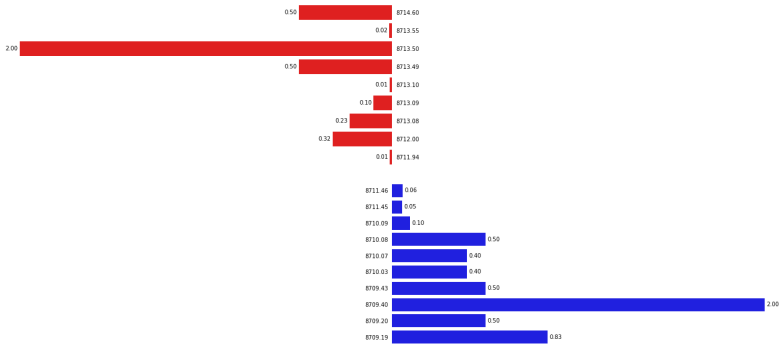

Figure 2.2: Diagram illustrating a sample depth 10 limit order book. [7]

now buying at the price of 8712.00 which has a higher quantity. This example demonstrates the need for high liquidity for traders trading large volumes, otherwise they quickly consume several levels and end up buying at unexpected high price.

The symmetrical behavior occurs for the top blue line representing the best bid at a price of 8711.46. Similarly, traders who want to sell large volumes will consume levels and end up selling at a much lower price than expected.

Note that the best ask price, 8711.93, is higher than the best bid price of 8711.46, and the bid-ask spread is the difference, 0.47. The midpoint, generally used as the asset price, is the average between the best bid and best ask, thus in this case 8711.695.

2.3 Indicators

From the raw data, indicators can be computed to give a specific feature and extract information that could be correlated to future prices.

Some examples used in the environment [4] are the following:

Change in midpoint
Logarithmic difference between the current and the previous midpoints.

$$\delta m_t = logm_t - logm_{t-1}$$

where m_t is the midpoint price at time t.

Spread
Difference between the best bid and the best ask

$$spread_t = bestbid_t - bestask_t$$

Trade Flow Imbalance (TFI)
Indicator measuring the imbalance between buys initiated b and sells s initiated transactions over a time window w.

$$TFI_t = \frac{UP_t - DWN_t}{UP_t + DWN_t}$$

where $UP_t = \sum_{i=0}^{w} b_i$ and $DWN_t = \sum_{i=0}^{w} s_i$.

Custom Relative Strength Index (CRSI)
Normalized measure of the magnitude of price changes over a time window.

$$CRSI_t = \frac{gain_t - |loss_t|}{gain_t + |loss_t|}$$

where $gain_t = \sum_{i=0}^{w} \Delta m_i$ if $\Delta m_i > 0$ else 0 and $loss_t = \sum_{i=0}^{w} \Delta m_i$ if $\Delta m_i < 0$ else 0 and $\Delta m_t = \frac{m_t}{m_{t-1}} - 1$.

Chapter 3

Environment

In this section, I present the two environments used to train the reinforcement learning agents as well as the reward functions.

3.1 Simple environment

The first environment implemented simulates a simplified financial market in which the agent owns a portfolio and manages the balance between its cash and a cryptocurrency asset. It incurs small fees to the agent for each transaction similar to a real cryptocurrency exchange. In order to make informed decisions, it observes historical and current market data as well as the state of its own portfolio. Table 3.1 summarizes the characteristics of this environment.

TABLE 3.1: Summary of the main modeling choices for the simple environment.

Term	Description
Data	1. Train: BTC, 7 months; 2. Test: ETH, 7 months
State Space	1. OHLCV data; 2. Portfolio status
Action Space	1. Discrete ($\in 0, 1, 2$); 2. Continuous ($\in$ [0,1])
Reward	1. $networth_t$ 2. $networth_t - networth_{t-1}$ 3. $balance_t$
Starting balance	1 000 000
Metric	1. Final net worth; 2. Episode return

3.1.1 State space

The state space consists of public and private data. Public data is represented by the Open, High, Low, Close, and Volume information of an asset with a window lag to include historical data, set to 100. Private data consists of the current state of the portfolio. It is composed of the balance representing the current liquidity of the portfolio, the shares held, i.e. the quantity of an asset held by the agent, and the net worth, i.e. the total value of our portfolio by

adding the value of the liquidity and the value of assets. For public data, the training phase uses Bitcoin OHLCV data while the testing phase uses Ethereum data. Both data frames last 7 months with one data point every hour. The size of the state space is therefore

$$\text{State space size} = 5 \times (w + 1) + 3$$

where w is the number of past data points being concatenated to the current observation. Three observations are added for the portfolio status.

3.1.2 Action space

The action space spans all possible actions that the agent can perform. It is a combination of discrete and continuous actions. The order side o_t is the discrete component that can take three values 0,1,2 corresponding to buy, sell or hold orders. The amount x_t is the continuous component ranging from 0 to 1 representing the percentage of the agent's ability to buy or sell. In the case of a buy order, the ability to buy is given by the balance, the available liquidity, and an amount equal to 1 means that the agent is investing all its cash in the asset. In the case of a sell order, the ability to sell is given by the shares of asset held. An amount equal to 1 means that the agent sells all its assets, removing all exposure to changes in the asset price. The action at time t is represented as follows:

$$a_t = (x_t \in [0, 1], o_t \in 0, 1, 2)$$

3.2 Limit order book environment

The environment considered above is great for its simplicity making it understandable by every human being that can assess the trading behavior. However, it lacks a large amount of information included in the markets that can be useful for the agent to have a more accurate view of the markets. It also lacks a realistic approach to trading in a real financial market when buyers fill *ask* orders and sellers fill *bid* orders, for market orders. Because of the spread between both, buyers buy at a higher price than sellers sell at. The following work [4] proposes a framework that allows to train an agent in such an environment. It explains the complexities of the environment in detail, therefore I will only summarize the main features.

3.2.1 State space

Data collection and feature engineering

Data is recorded from cryptocurrency market exchanges, here Bitmex, and reconstructs the limit order book at a frequency of one second. Every second, it records the price and quantity at depth 20 of the limit order book. Similar to schema 2.2 but records 20 bars in the bid side in blue and 20 bars in the ask side in red, each bar consisting of the price and the quantity, thus $2 \times 20 \times 2 = 80$ values in total. The notation used is $p_{t,i}^{side}$ for the price at time t, LOB level i and side $\in \{bid, ask\}$ as well as $q_{t,i}^{side}$ for the associated quantity. The dollar value of each level can then be computed as follows:

$$\mathcal{X}_{t,i}^{side} = \sum_{i=0}^{I-1} p_{t,i}^{side} \times q_{t,i}^{side}$$

where I ranges from 1 to 20.

The dollar value computed above allows to measure the imbalances in the LOB between the ask and bid sides. At each level, the imbalance $\iota_{t,i}$ can be computed:

$$\iota_{t,i} = \frac{\mathcal{X}_{t,i}^{ask} - \mathcal{X}_{t,i}^{bid}}{\mathcal{X}_{t,i}^{ask} + \mathcal{X}_{t,i}^{bid}}$$

Recording the live limit order book of exchanges allows to get informations about the orders, notably whether they are cancel C, limit L or market M orders. Dollar values for each are added to the features, at each LOB level i and for each side.

$$\begin{aligned} C_{t,i}^{side} &= p_{t,i}^{side} \times q_{t,i}^{side} \\ L_{t,i}^{side} &= p_{t,i}^{side} \times q_{t,i}^{side} \\ M_{t,i}^{side} &= p_{t,i}^{side} \times q_{t,i}^{side} \end{aligned}$$

Finally, indicators detailed in 2.3 are also part of the observation space. They are computed over multiple windows: 5, 10 and 15 minutes for RSI and 5, 15 and 30 minutes for TNS.

Data preprocessing

The normalization method is described in [8]. It deals with the non-stationarity of features in the limit order book by transforming the non-stationary prices into a percentage difference to the current midpoint, making the time series stationary. Then it uses a z-score normalization using the mean and standard deviation of previous days' data.

Position features

The position features contain features related to the agent's portfolio status including the realized PNL remaining constant if the agent holds and unrealized PNL sensitive to the variations of the best ask and best bid. The starting balance is also 1000000.

3.2.2 Action space

The action space is the same as in the simple environment presented above 3.1.2, therefore each action is denoted as $a_t = (x_t \in [0, 1], o_t \in 0, 1, 2)$.

3.3 Reward functions

The reward function drives the learning of the reinforcement learning agent. It is very useful in financial applications as we can keep the same goal of making profits, but through different strategies and different risk acceptances. Several rewards functions were tried out in this project.

3.3.1 Net worth

This first simple reward functions allows the agent to understand that the key is to always keep the net worth high. It also gives a continuous reward since the net worth is always positive, unless the agent loses all the money in which case the episode is stopped due to lack of funds.

$$r_t = networth_t$$

3.3.2 Difference in net worth

The net worth difference with the previous timestep provides a way to assess the impact of the current action, but it is highly exposed to asset price movements, even if the agent acts positively.

$$r_t = networth_t - networth_{t-1}$$

3.3.3 Realized PnL

Another reward is the realized PnL, which provides the agent with a constant feedback signal and an incentive to increase the liquidity throughout the episode.

$$r_t = balance_t$$

3.3.4 Differential sharpe ratio

Used in [4], the differential sharpe ratio provides the agent with a continous signal feedback, representing the risk-adjusted return to which the agent has exposed itself. The formula is as follows:

$$DSR_t = \frac{B_{t-1}\delta A_t - \frac{1}{2}A_{t-1}\delta B_t}{(B_{t-1} - A_{t-1}^2)^{\frac{3}{2}}}$$

where

$$\begin{aligned} A_t &= A_{t-1} + \eta(R_t - A_{t-1}) \\ B_t &= B_{t-1} + \eta(R_t^2 - B_{t-1}) \\ \delta A &= R - A_{t-1} \\ \delta B &= R^2 - B_{t-1} \\ R &= Inventory_t \cdot m_t \end{aligned}$$

m_t being the midpoint price at time t, $Inventory_t$ the number of assets held at time t.

3.3.5 Trade completion

Also used in [4], trade completion provides the agent with a goal-based feedback signal, with a reward $r_t \in [-1, 1]$ depending on the success of a profit objective defined by a threshold.

Chapter 4

Models

4.1 Hierarchical reinforcement learning

In this section, I present a new promising way of modeling policies called hierarchical reinforcement learning.

4.1.1 Theory

Definition

The essence of this work is to implement and experiment with a recent method for solving Markov decision processes (MDP) in reinforcement learning, namely hierarchical reinforcement learning. The former abstracts the idea of a “macro-operator” [9] [10] sometimes called "meta" or simply “macro” which is a sequence of operators or actions that a higher-level policy can invoke as a primitive action. The concept of hierarchy comes from the fact that these higher-level policies can take advantage of the output of lower-level policies as a black box, which may themselves have lower-level policies. More concretely, as described in TradeR [5], the global policy $\pi(a_t|s_t)$ can be decomposed into many components, specifically many policies $\pi_i(a_t|s_t)$ each responsible for solving a sub-MDP. This hierarchical abstraction of policy into its components allows temporal coherency in state and value predictions. These components can then be coupled together in varied forms such as a master-slave structure or a sequence, in which cases the sub-MDPs interact simultaneously or sequentially, respectively.

Benefits

In many reinforcement learning applications, the hierarchy offers multiple advantages [11] [10] over classical RL. First, it allows for structured exploration, meaning that the agent can explore through sub-policies rather than trying multiple combinations of primitive actions. By construction, each task is properly trained independently of the others. Therefore, each level of the hierarchy can

embody different knowledge allowing for transfer learning. Moreover, as each agent focuses on an easier task, often with a smaller state space or action space, this helps tackle the curse of dimensionality in problems with too large spaces. HRL offers a solution for many applications suffering from a scaling problem. Finally, by deconstructing a task into several simple policies, the reinforcement learning algorithms are more sample efficient because they require less interaction with the environment and thus fewer data to achieve good performance.

4.1.2 Hierarchical Reinforcement Learning with TradeR

TradeR [5] aims to address two practical challenges, namely *catastrophy* and *surprise minimization* by formulating trading as a real-world hierarchical reinforcement learning problem. Solving catastrophy minimization relies on the hierarchical structure, allowing for an improved sample efficiency and thus tolerate trial and error learning for a shorter amount of time. Surprise minimization is addressed be estimating the deviation of the state, embedded in the learning of the agent in order to make it robust to abrupt changes.

The implementation of TradeR is based on PPO [12] which consists of an Actor-Critic framework as shown in Figure 4.1. The actor is the decision-maker and where lies the hierarchical architecture. It consists of order and bid networks as two distinct policies. The critic is used to evaluate the current actor policy and to update the network parameters. The main feature of PPO is that it updates the policy in such a way that it does not incur performance collapse relative to the previous step policy, hence the name proximal policy. This goal can be achieved by clipping the updated policy so that it does not deviate too much from the old policy.

PPO is an on-policy algorithm and TradeR as well with a rollout buffer. Each transition (a_t, s_t, r_t) is appended to a buffer that is periodically rolled out in reversed order to compute the discounted rewards and advantages estimation. The loss consists of the PPO loss, thus a combination of a mean-square error component that measures the difference between the discounted rewards and the state values evaluated by the critic network, and a surrogate loss function to ensure that the policies remain proximal and to avoid abrupt drops in performance. The PPO loss also includes an entropy term to reward high-entropy action distributions and encourage exploration. The parameters of the order and bid policies, as well as the critic policy, are updated jointly.

As discusses, the hierarchy lies in the actor part. I detail the structure of the actor using two sequential policies, each addressing a sub-problem.

The goal of the actor is to make a decision i.e. an action in the action space consisting of a tuple. Each component of the tuple has its own sub-policy.

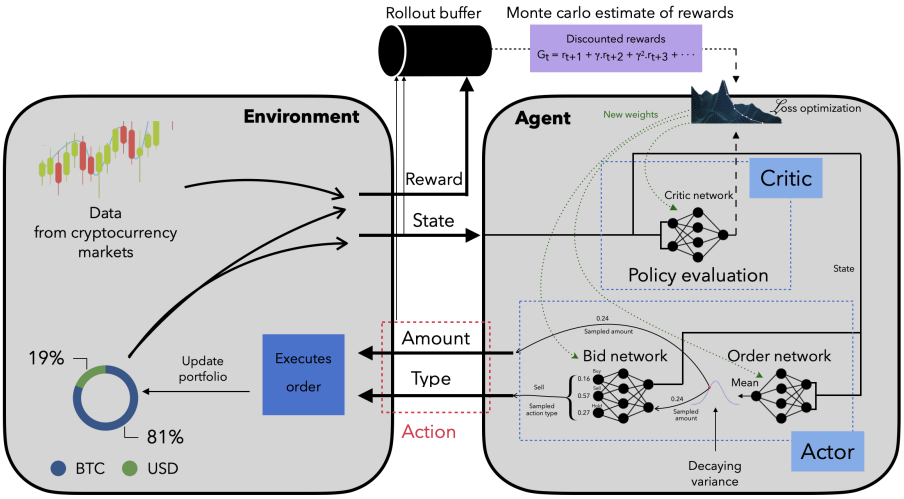

Figure 4.1: Diagram illustrating the components of TradeR.

Order determination: The first sub-problem is the choice of a quantity to invest in assets or to retrieve in cash. At a given timestep t, the order network estimates this quantity $\bar{x}_t$ by observing the state s_t following the sub-policy $\pi_{ord}(\bar{x}|s_t)$. The output of the network is the mean quantity which, with a standard deviation hyperparameter σ_t, defines a normal distribution from which we can sample to use the sampled quantity $x_t \sim \mathcal{N}(\bar{x}_t, \sigma_t)$ in our future order. This allows us to explore different amounts in the distribution and tune the standard deviation towards more exploitation or exploration. This hyperparameter decreases during learning to account for the agent's improvement over time.

Bid execution: The bid policy is built on top of the order policy because it observes the same state s_t but also the output of the former policy, x_t. The bid network, using the sub-policy $\pi_{bid}(\bar{o}_t|s_t, x_t)$, returns three values, summing up to 1, that form a categorical distribution between buy, sell, and hold actions. Sampling this softmax distribution yields the decided order type $o_t \in \{0, 1, 2\}$.

From the quantity amount sampled from the first policy x_t and the discrete action to be taken, sampled from the second policy o_t, we can construct the final action $a_t = (o_t \in \{0, 1, 2\}, x_t \in [0, 1])$ as a tuple that determines the order to be executed by the environment.

4.1.3 Surprise reward function

The implementation of a hierarchical reinforcement learning agent from TradeR [5] includes an energy-based intrinsic motivation aiming at making the agent robust to abrupt changes in the observation space. The idea is derived from contraction theory and is applied by transforming a surprise signal from the

agent with a contraction operator τ that will be added to the reward to utilize surprise minimization as an intrinsic motivation objective.

The contraction operator τ is the log-sum-exp operator

$$f(w) = log \sum_{w} exp(f(w))$$

and the updated reward signal is

$$\hat{r}(s_t, a_t, \sigma_i) = r(s_t, a_t) + log \sum_{w} exp(V_{surp}(\sigma_i))$$

where σ_i is the standard deviation across all values of the batch for feature i and V_{surp} is a neural network with a single layer and a ReLU non-linear activation function.

Chapter 5

Experiments

5.1 Experiments

The following table 5.1 summarizes the main results for the first, simple, environment considered.

TABLE 5.1: Final results for the simple environment.

Model	Reward function	Final networth	Return
Baseline model			
- PPO	Net worth	1 049 040	4.90 %
- PPO	Difference Net worth	982 610	-1.74 %
- PPO	Realized PnL	960 332	-3.97 %
- A2C	Net worth	993 902	-0.61 %
- A2C	Difference Net worth	1 000 584	0.06 %
- A2C	Realized PnL	1 039 690	3.97 %
TradeR			
- without surprise term	Net worth	948 628	-5.14 %
- with surprise term	Net worth	1 034 960	3.5 %
- without surprise term	Difference Net worth	981 257	-1.87%
- with surprise term	Difference Net worth	986 143	-1.39%

and table 5.1 summarizes the results for the second environment.

Figure 5.1 shows the trading behavior of a PPO agent and figure 5.2 of an A2C agent. We observe that PPO trades in a sparser fashion while A2C has compulsive buy periods followed by compulsive sell periods.

Figure 5.3 shows the trading behavior from TradeR.

It is worth noticing that these experiments results are not comparable to the results observed in [4] as not only they are not trained and tested on the same periods but [4] uses a market making trading strategy which is fundamentally different to the trend following strategy. The market maker supplies liquidity on

Figure 5.1: Plot of the executed trades by a PPO agent.

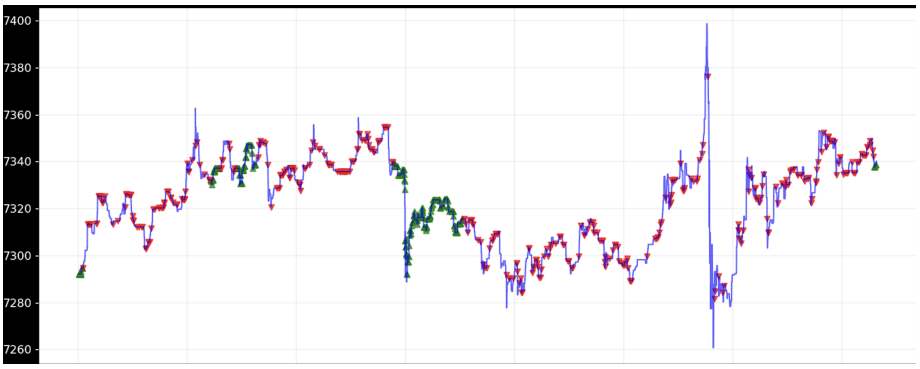

Figure 5.2: Plot of the executed trades by an A2C agent.

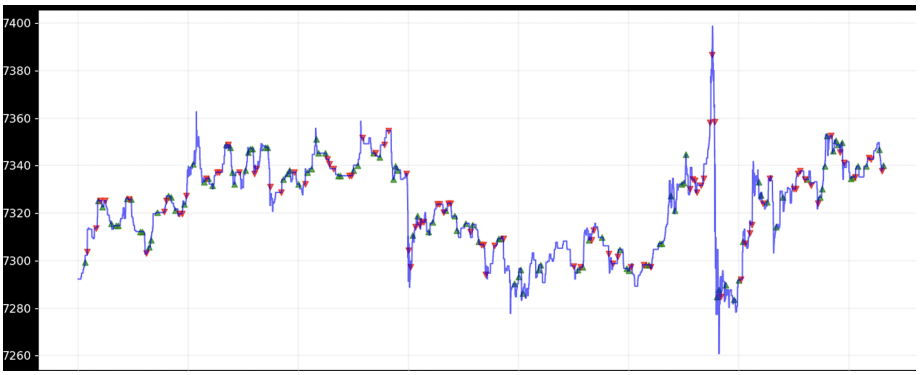

Figure 5.3: Plot of the executed trades by a TradeR agent.

TABLE 5.2: Final results for the second environment.

Model	Reward function	Final networth	Return
Baseline model			
- PPO	Net worth	1 005 645	0.56 %
- PPO	Difference Net worth	1 005 524	0.55 %
- PPO	Realized PnL	1 006 754	0.68 %
- PPO	Differential Sharpe Ratio	1 001 877	0 %
- PPO	Trade Completion	1 000 926	0.09 %
- A2C	Net worth	1 005 686	0.57 %
- A2C	Difference Net worth	1 000 570	0.06 %
- A2C	Realized PnL	1 006 203	0.62 %
- A2C	Differential Sharpe Ratio	1 002 629	0.26 %
- A2C	Trade Completion	1 003 178	0.32 %
TradeR			
- without surprise term	Net worth	1 001 852	0.19 %
- with surprise term	Net worth	999 796	-0.02 %
- without surprise term	Difference Net worth	1 003 945	0.39%
- with surprise term	Difference Net worth	1 001 911	0.19%
- without surprise term	Realized PnL	995 290	-0.47%
- with surprise term	Realized PnL	997 140	-0.29%
- without surprise term	Differential Sharpe Ratio	1 009 222	0.92%
- with surprise term	Differential Sharpe Ratio	994 640	-0.54%
- without surprise term	Trade Completion	998 291	-0.17%
- with surprise term	Trade Completion	1 007 115	0.71%

both sides by providing bids and asks, making profit from the bid-ask spread as well as getting fees rather than paying fees.

From these experiments, I do not observe much improvement from the surprise term added to the reward but that the net worth, included in the observation space, deviates less from the initial value. Reasons for not improvement much positive performance could include the lack of training time, lack of hyperparameter tuning but would overall need to dig more into the learning of each policy to understand the strategies in use.

Chapter 6

Conclusion

6.1 Further work

There is a lot of future work to be done in improving hierarchical structures to tackle the curse of dimensionality with financial data entering the big data era. By deconstructing decisions into smaller dimensional ones, an example of interesting work could be to develop explainable models to justify the choices of the agent at each level and evaluate accountability when a large loss is incurred or identify potential edges when recurring good decisions appear. The high volatility of cryptocurrency market is not yet well handled by machine learning and reinforcement learning techniques, more robust methods of reducing surprise states could be developed. Finally, with a drastic change in the environment, similar techniques could be applied to Automated Market Makers (AMM) [13] to explore the opportunities in decentralized finance.

6.2 Conclusion

In this work, I exploit the concept of hierarchical reinforcement learning to encourage an agent to optimize each sub-part of its decisions. The results show a positive outcome on average since in most experiments the agent makes profits even in a realistic setting including transaction fees. However, these experiments are also characterized by an instability in the behaviors and further work should be done in the explainability of decision drivers.

Bibliography

[1] M. L. de Prado, *Advances in Financial Machine Learning*, 2018. [Online]. Available: https://ssrn.com/abstract=3104847

[2] S. Jansen, *Machine Learning for Algorithmic Trading*, 2020. [Online]. Available: https://ssrn.com/abstract=3104847

[3] C. López-Martín, S. Benito Muela, and R. Arguedas, "Efficiency in cryptocurrency markets: new evidence," *Eurasian Economic Review*, vol. 11, no. 3, pp. 403–431, Sep 2021. [Online]. Available: https://doi.org/10.1007/s40822-021-00182-5

[4] J. Sadighian, "Extending deep reinforcement learning frameworks in cryptocurrency market making," 2020.

[5] K. Suri, X. Q. Shi, K. Plataniotis, and Y. Lawryshyn, "Trader: Practical deep hierarchical reinforcement learning for trade execution," 2021.

[6] J. Fernandez-Tapia, "Modeling, optimization and estimation for the on-line control of trading algorithms in limit-order markets," Ph.D. dissertation, 09 2015.

[7] "Understanding the limit order book," *Tradient blog*, 2020.

[8] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Using deep learning for price prediction by exploiting stationary limit order book features," *CoRR*, vol. abs/1810.09965, 2018. [Online]. Available: http://arxiv.org/abs/1810.09965

[9] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," vol. 13, p. 2003, 2003.

[10] E. S. Pierre Motard, Shuaijun Gao, "Hierarchical rl for cryptocurrency trading," *Course project from Foundations of Reinforcement Learning*, 2022.

[11] Y. Flet-Berliac, "The promise of hierarchical reinforcement learning," *The Gradient*, 2019.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[13] J. Xu, K. Paruch, S. Cousaert, and Y. Feng, "Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols," 2022.