# Commission Fee is not Enough: A Hierarchical Reinforced Framework for Portfolio Management

**Rundong Wang**[1*], **Hongxin Wei**[1*], **Bo An**[1], **Zhouyan Feng**[2], **Jun Yao**[2]

[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[2] WeBank Co. Ltd., China
{rundong001, hongxin001}@e.ntu.edu.sg, boan@ntu.edu.sg, {yanisfeng, junyao}@webank.com

## Abstract

Portfolio management via reinforcement learning is at the forefront of fintech research, which explores how to optimally reallocate a fund into different financial assets over the long term by trial-and-error. Existing methods are impractical since they usually assume each reallocation can be finished immediately and thus ignoring the price slippage as part of the trading cost. To address these issues, we propose a hierarchical reinforced stock trading system for portfolio management (*HRPM*). Concretely, we decompose the trading process into a hierarchy of portfolio management over trade execution and train the corresponding policies. The high-level policy gives portfolio weights at a lower frequency to maximize the long term profit and invokes the low-level policy to sell or buy the corresponding shares within a short time window at a higher frequency to minimize the trading cost. We train two levels of policies via pre-training scheme and iterative training scheme for data efficiency. Extensive experimental results in the U.S. market and the China market demonstrate that *HRPM* achieves significant improvement against many state-of-the-art approaches.

## Introduction

The problem of portfolio management is widely studied in the area of algorithmic trading. It aims to maximize the expected returns of multiple risky assets. Recently, reinforcement learning (RL) models are gaining popularity in the fintech community (Mosavi et al. 2020), with its great performance in different fields, including playing games (Silver et al. 2016; Mnih et al. 2013), controlling robots (Lillicrap et al. 2015), and the Internet of things (Zhang and Tao 2020).

Due to the dynamic nature of markets and noisy financial data, RL has been proposed as a suitable candidate by its own nature (Almahdi and Yang 2017; Jiang and Liang 2017). That is, RL can directly output trade positions, bypassing the explicit forecasting step by the trial-and-error interaction with the financial environment. Many existing RL methods get promising results by focusing on various technologies to extract richer representation, e.g., by model-based learning (Tang 2018; Yu et al. 2019), by adversarial learning (Liang et al. 2018), or by state augmentation (Ye

et al. 2020). However, these RL algorithms assume that portfolio weights can change immediately at the last price once an order is placed. This assumption leads to a non-negligible trading cost, that is, *the price slippage* – the relative deviation of the expected price of a trade and the price actually achieved. In a more realistic trading environment, the trading cost should be taken into consideration. Moreover, due to the need of balancing the long-term profit maximization and short-term trade execution, it is challenge for a single/flat RL algorithm to operate on different levels of temporal tasks.

As our motivation, we observe that there is a hierarchy of portfolio managers and traders in real-world trading. Portfolio managers assign a percentage weighting to every stock in the portfolio periodically for a long-term profit, while traders care about the best execution at the favorable price to minimize the trading cost. This paper gives the first attempt to leverage a similar idea to algorithmic trading. Concretely, we develop a **H**ierarchical **R**einforced trading system for **P**ortfolio **M**anagement (*HRPM*). Our system consists of a hierarchy of two decision processes. The high-level policy changes portfolio weights at a lower frequency, while the low-level policy decides at which price and what quantity to place the bid or ask orders within a short time window to fulfill goals from the high-level policy.

Our contributions are four-fold. First, we formulate the portfolio management problem with trading cost as a hierarchical MDP, and combine portfolio management and trade execution by utilizing the hierarchical RL (HRL) framework to address the non-zero slippage. Second, as traditional HRL performs poorly in this problem, we propose the entropy bonus in the high-level policy to avoid risk, and use action branching to handle multi-dimensional action space. Third, to make full use of the stock data, we pre-train multiple low-level policies with an iterative scheme for different stocks in the limit order book environment, and then train a high-level policy on the top of the low-level policies. Fourth, we compare *HRPM* with baselines based on the historical price data and limit order data from the U.S. market and the China market. Extensive experimental results demonstrate that *HRPM* achieves significant improvement against state-of-the-art approaches. Furthermore, the ablation studies demonstrate the effectiveness of entropy bonuses and the trading cost is a non-negligible part in portfolio management.

---

*Equal contribution

## Problem Formulation

In this section, we introduce some definitions and the objective. Portfolio management is a fundamental financial task, where investors hold a number of financial assets, e.g., stocks, bonds, as well as cash, and reallocate them periodically to maximize future profit. We split the time into two types of periods: holding period and trading period as Fig. 1. During the holding period, the agent holds the pre-selected assets without making any purchase or selling. With the fluctuations of the market, the assets' prices would change during the holding period. At the end of the holding period, the agent will decide the new portfolio weights of the next holding period. In the trading period, the agent buys or sells some shares of assets to achieve the new portfolio weights. The lengths of the holding period and trading period are based on specific settings and can change over time.

**Definition 1.** (Portfolio) A portfolio can be represented as:

$$\boldsymbol{w}_t = [w_{0,t}, w_{1,t}, w_{2,t}, \ldots, w_{M,t}]^T \in \mathbb{R}^{M+1} \text{ and } \sum_{i=0}^{M} w_{i,t} = 1 \quad (1)$$

where $M + 1$ is the number of portfolio's constituents, including one risk-free asset, i.e., cash, and $M$ risky assets, i.e., stock[1]. $w_{i,t}$ represents the ratio of the total portfolio value (money) invested at the beginning of the holding period $t$ on asset $i$. Specifically, $w_{0,t}$ represents the cash in hand. Also, we define $\boldsymbol{w'_t}$ as the portfolio weights at the end of the holding period $t$.

**Definition 2.** (Asset Price) The opening prices $\boldsymbol{p}_t$ and the closing prices $\boldsymbol{p'}_t$ of all assets for the holding period $t$ are defined respectively as $\boldsymbol{p}_t = [p_{0,t}, p_{1,t}, p_{2,t}, \ldots, p_{M,t}]^T$ and $\boldsymbol{p'}_t = [p'_{0,t}, p'_{1,t}, p'_{2,t}, \ldots, p'_{M,t}]^T$. Note that the price of cash is constant, i.e., $p_{0,t} = p'_{0,t} = p_{0,t+1}$ for $t \geq 0$.

**Definition 3.** (Portfolio Value) We define $v_t$ and $v'_t$ as portfolio value at the beginning and end of the holding period $t$. So we can get the change of portfolio value during the holding period $t$ and the change of portfolio weights:

$$v'_t = v_t \sum_{i=0}^{M} \frac{w_{i,t} p'_{i,t}}{p_{i,t}} \text{ and } w'_{i,t} = \frac{\frac{w_{i,t} p'_{i,t}}{p_{i,t}}}{\sum_{i=0}^{M} \frac{w_{i,t} p'_{i,t}}{p_{i,t}}} \text{ for } i \in [0, M] \quad (2)$$

**Definition 4.** (Market order) A market order refers to an attempt to buy or sell a stock at the current market price, expressing the desire to buy (or sell) at the best available price.

**Definition 5.** (Limit Order) A limit order is an order placed to buy or sell a number of shares at a specified price during a specified time frame. It can be modeled as a tuple $(p_{target}, \pm q_{target})$, where $q_{target}$ represents the submitted target price, $q_{target}$ represents the submitted target quantity, and $\pm$ represents trading direction (buy/sell).

**Definition 6.** (Limit Order Book) A limit order book (LOB) is a list containing all the information about the current limit orders.

---

[1]In this paper, we focus on the stocks for ease of explanation. This framework is applicable to other kinds of assets.
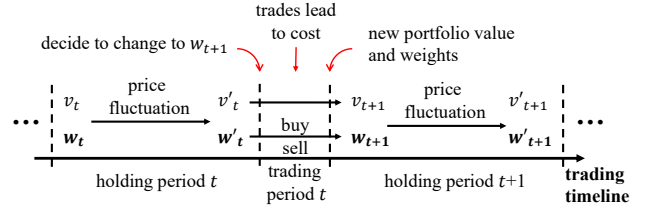


Figure 1: Illustration of the portfolio management process.

We observe that a strategy pursuing a better selling price is always associated with a longer (expected) time-to-execution. However, the long time-to-execution could be very costly, since a strategy might have to trade at a bad price at the end of the trading period, especially when the market price moves downward.

**Definition 7.** (Trading Cost) Trading cost consists of commission fee and slippage: $c_{trade} = c_{com} + c_{slippage}$. Commissions are fees that brokers charge to implement trades and computed as $c_{com} = \lambda \sum_{i=1}^{M} (q_{i,target} \times p_{i,avg})$, where $\lambda$ is the constant commission rate for both selling and buying. Slippage is the difference between the expected price of a trade and the price actually achieved. Here we define the slippage as the average execution price $p_{avg}$ achieved by the strategy relative to the price at the end of the trading period $t$: $c_{slippage} = (p_{avg} - p_{t+1}) \times (\pm q_{target})$.

Considering the trading cost, the change of portfolio value during a holding period and a trading period $t$ satisfies:

$$v_{t+1} = \underbrace{v_t w_{0,t} - \sum_{i=1}^{M} [\pm q_{i,target} \times p_{i,avg} + \lambda q_{i,target} \times p_{i,avg}]}_{\text{Cash}}$$

$$+ \underbrace{\sum_{i=1}^{M} \left[ (\frac{v_t w_{i,t}}{p_{i,t}} \pm q_{i,target}) \times p_{i,t+1} \right]}_{\text{Assets Value}}$$

$$= \sum_{i=0}^{M} \frac{v_t w_{i,t} p_{i,t+1}}{p_{i,t}} - c_{trade,t} \quad (3)$$

Our objective is to maximize the final portfolio value given a long time horizon by taking into account the trading cost.

## Hierarchical MDP Framework

In this section, we propose a hierarchical MDP framework to handle the real-world trading scenarios with slippage. We start by introducing the Markov decision process (MDP), which is defined by the tuple: $\text{MDP} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, T)$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is a transition probability function, $r : \mathcal{S} \to \mathbb{R}$ is a bounded reward function, $\gamma \in (0, 1]$ is a discount factor and $T$ is a time horizon. In an MDP, an agent at state $s_t \in \mathcal{S}$ performs an action $a_t \in \mathcal{A}$. The agent's actions are often defined by a policy $\pi_\theta : S \to \mathcal{A}$ parameterized by $\theta$. A Q-value
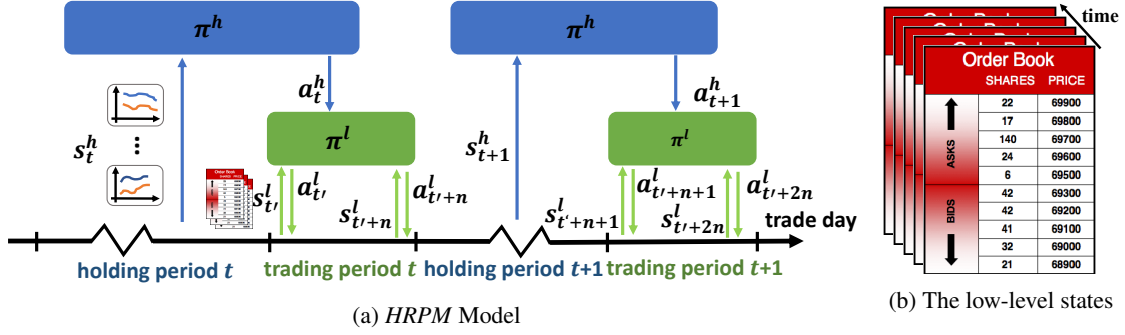
(a) HRPM Model      (b) The low-level states

Figure 2: Overview of *HRPM*

function gives expected accumulated reward when executing action $a_t$ in state $s_t$ and following policy $\pi$ in the future, which is $Q^\pi(s_t, a_t) = \mathbb{E}_{(s_{t+1}, \cdots \sim \pi)} \left[ \sum_{i=t}^{T} \gamma^i r(s_i, a_i) \right]$. The objective of the agent is to learn an optimal policy: $\pi_{\theta^*} = \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta} \left[ \sum_{i=0}^{T} \gamma^i r_{t+i} | s_t = s \right]$.

A key challenge to the real-world portfolio management is to balance the multifaceted and sometimes conflicting objectives of different decision processes. Portfolio managers' main concerns are about longer-term profit. They typically think of execution as a chore to fulfill the clients' profit requirements and risk preferences. While traders care about best execution at the most favorable price to minimize the trading cost, but might not always be aware of managers' sources of profit and urgency of trade. Moreover, flat RL cannot handle complex tasks with long time horizons. Portfolio managers take many hours (sometimes days) while agents need to make decisions every few seconds or faster. The sampling frequency is limited by this time horizon issue so that all available information about market dynamics is not fully utilized. Consequently, electronic trading algorithms are supposed to operate on multiple levels of granularity.

Formally, we model the portfolio management with trading cost as two hierarchical MDPs, respectively for portfolio management and trade execution:

$$\text{MDP}^h = (\mathcal{S}^h, \mathcal{A}^h, \mathcal{P}^h, r^h, \gamma, T^h)$$

$$\text{MDP}^l = (\mathcal{S}^l, \mathcal{A}^l, \mathcal{P}^l, r^l, \gamma, T^l)$$

where the detailed definitions are given in the following subsections. The high-level policy and the low-level policy are executed in different timescales. Concretely, the time horizon for the high-level MDP is the total holding time of the portfolio, and the timestep of the high-level MDP is the holding period, which might consist of several trading days. On the other hand, the time horizon for the low-level MDP is a small trading window, e.g., a trading day. The timestep of the low-level MDP could be several minutes for the low-level policy to take actions. We use $t$ as the high-level timestep, and $t'$ as the low-level timestep.

## The High-level MDP for Portfolio Management

The high-level MDP is modeled for the portfolio management problem. That is, the trading agent gives new portfolio weights for reallocating the fund into a number of financial assets at the beginning of each holding period.

**State.** We describe the state $s_t^h = \{ \boldsymbol{X}_t, \boldsymbol{w_t} \} \in \mathcal{S}^h$ with historical stock basic features $X_t$, and the current portfolio weights $\boldsymbol{w_t}$. For the $M$ non-cash assets, the features $\boldsymbol{X}_{i,t}$ for asset $i$ are built by a time window $k$: $\boldsymbol{X}_{i,t} = \{ \boldsymbol{x}_{i,t-k+1}, \boldsymbol{x}_{i,t-k+2}, \cdots, \boldsymbol{x}_{i,t} \}$, where $\boldsymbol{x}_{i,t}$ presents the basic information of asset $i$ at trading day $t$, including opening, closing, highest, lowest prices and volume. Note that the state can be enriched by adding more factors for better performance. We only use basic daily information to describe high-level state in this work for simplification and fair comparison.

**Action.** At the end of the holding period $t$, the agent will give a high-level action $a_t^h$ based on the state $s_t^h$ to redistribute the wealth among the assets as a subtask. The subtask is determined by the difference between portfolio weights $\boldsymbol{w_t'}$ and $\boldsymbol{w_{t+1}}$. Since $\boldsymbol{w_t'}$ has already been determined in the holding period $t$ according to Eq. (2), the action of the agent in holding period $t$ can be represented solely by the portfolio vector $\boldsymbol{w_{t+1}}$. Consequently, we define the action as $a_t^h = \boldsymbol{w_{t+1}}$ in the continuous action space, and the action $a_t$ holds the properties in Eq. (1).

**Reward.** For each state-action pair, i.e., $(s_t^h, a_t^h)$ at holding period $t$, according to Eq. (3), $r_t = v_{t+1} - v_t$.

## The Low-level MDP for Trade Execution

The high-level policy assigns subtasks for the low-level policy. The subtasks are represented by an allowed time window with length of $T_{window}$ and the target trading quantity $q_{target}$. $T_{window}$ is based on manual configurations, and $q_{target,t}$ of trading period $t$ can be computed by $q_{target,t} = v_t' | w_{t+1} - w_t' | \odot \frac{1}{p_t'}$, with assumption of a continuous market. The subtasks can be considered as the trade execution, where the trading agent places many small-sized limit orders at different times at corresponding desired prices. As a result, we model the subtask as the low-level MDP.

**State.** We maintain a state $s_{t'}^l$ at each low-level time step $t'$, considering the private states and the market states. The private states of trade execution are remaining trading time

$T_{window} - T_{elapsed}$ and the remaining quantity $q_{target,t} - q_{finished}$, where $T_{elapsed}$ is the used time and $q_{finished}$ is the finished quantity. The market states consist of the historical limit order books and multiple technical factors as shown in Fig. 2b. Specifically, we collect historical LOBs in a time window $k'$: $\boldsymbol{LOB}_{t'} = \{\boldsymbol{lob}_{t'-k}, \boldsymbol{lob}_{t'-k+1}, \cdots, \boldsymbol{lob}_{t'-1}\}$, where $\boldsymbol{lob}_{t'}$ is the limit order book at time step $t'$, including price and volume.

**Action.** Each low-level action corresponds to a limit order decision with the target price $p_{target}$ and the target quantity $\pm q_{target}$. Note that a zero quantity indicates that we skip the current trading time step with no order placement, and if an action $a_{t'}$ fails to be executed in the low-level time step $t'$ due to an inappropriate price, the action will expire at the next low-level time step $(t' + 1)$ without actual trading. Any quantity remaining at the end of the trading period must be cleaned up by using a market order, walking through the lower prices on the bid side of the order book until all remaining volumes are sold.

**Reward.** Once receiving a limit order decision, the environment will match this order and feedback an executed price $p_{paid}$. We define the low-level reward as: $r^l_{t'} = -[\lambda(\pm q_{target} \times p_{paid}) + (p_{paid} - p_{t+1}) \times (\pm q_{target})]$. So the cumulative trading cost in a low-level episode will be reported to the high-level policy to compute the high-level reward $c_{trade,t} = -\sum_{t'}^{T^l} r^l_{t'}$.

## Optimizing via Deep Hierarchical Reinforcement Learning

In this section, we first present our solution for learning hierarchical policies, then propose our solution algorithms respectively for the two levels of MDPs. Finally, we introduce two schemes: pre-training and iterative training for better financial data usage.

### Hierarchy of Two Policies

Our environment has two dynamics: the price dynamic and the limit order book dynamic. We extend the standard RL setup to a hierarchical two-layer structure, with a high-level policy $\pi^h$ and a low-level policy $\pi^l$. Different from the existing HRL algorithms, our high-level policy and the low-level policy run in different timescales. As shown in Fig. 2a, each high-level time step $t$ consists of three steps:

1. The higher-level policy observes the high-level state $s^h_t$ and produces a high-level action $a^h_t$, which corresponds to the new portfolio for the next step.

2. The high-level action $a^h_t$ would generate subtasks for the low-level policy according to the gap between the current portfolio and the new one.

3. The low-level policy produces the low-level action $a^l_{t'}$ based on the low-level states $s^l_{t'}$. These actions would be applied to the limit order dynamic for updating the current portfolio.

### High-level RL with Entropy Bonus

We consider the high-level problem as a continuous control problem, since the high-level policy is supposed to gener-

ate a continuous vector according to Eq. (1). Previous works utilize DDPG (Lillicrap et al. 2015) to generate portfolio weights. Unfortunately, DDPG faces challenges in solving our problem. First, DDPG highly relies on a well-trained critic. However, the critic also needs lots of data, which is limited in the financial area (Liang et al. 2018). Additionally, the exploration in DDPG is dependant on the Gaussian noise when applying actions into the environment. This exploration might fail due to choosing specific sequences of portfolio weights and excluding other possible sequences of portfolio weights. Also, we find that the portfolio weights will converge to a few assets. That is, the agent only holds few assets.

Here we utilized the REINFORCE algorithm (Sutton et al. 2000). One approach to address these issues is to introduce an entropy bonus. In order to encourage the high-level policy not to "put all the eggs in one basket", we aim to find a high-level policy that maximizes the maximum entropy objective:

$$
\begin{aligned}
\pi^{h*} &= \underset{\pi^h}{\arg\max} \sum_{t=0}^{T^h} E_{s^h_0 \sim \beta_0, s^h_{t+1} \sim p^h, a^h_t \sim \pi^h} [\gamma^h \tilde{r}^h(s^h_t, a^h_t)] \\
&= \underset{\pi^h}{\arg\max} \sum_{t=0}^{T^h} E_{s^h_0, s^h_{t+1}, a^h_t} \left[\gamma^h \left(r^h(s^h_t, a^h_t) + \eta \mathcal{H}(a^h_t)\right)\right]
\end{aligned}
\tag{4}
$$

where $\beta_0$ is the intial high-level state distribution, $s^h_{t+1} \sim p^h$ is an abbreviation of $s^h_{t+1} \sim p^h(\cdot|s^h_t, a^h_t)$, $a^h_t \sim \pi^h$ is an abbreviation of $a^h_t \sim \pi^h(\cdot|s^h_t)$, $\eta$ determines the relative importance of the entropy term versus the reward, $\mathcal{H}(a^h_t)$ is the entropy of the portfolio weights, which is calculated as $\mathcal{H}(a^h_t) = \mathcal{H}(\boldsymbol{w^h_{t+1}}) = -\sum_i w^h_{i,t+1} \log w^h_{i,t+1}$, and $\tilde{r}^h$ is an abbreviation of the summation of the high-level reward and entropy bonus.

We use the softmax policy as the high-level policy $\pi^h(s^h, a^h; \theta) = \frac{e^{\theta^T \cdot \phi(s^h, a^h)}}{\sum_b e^{\theta^T \cdot \phi(s^h, b)}}$ for all $s^h \in \mathcal{S}^h, a^h \in \mathcal{A}^h$. Let $J(\pi^h_\theta) = \mathbb{E}_{\pi^h} \left[\sum_{t=0}^{T} \gamma^t t_t\right]$ denote the expected finite-horizon discounted return of the policy. According to the policy gradient theorem (Sutton et al. 2000), we compute the gradient $\nabla_\theta J\left(\pi^h_\theta\right) = \underset{\tau \sim \pi_\theta}{\mathbb{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi^h_\theta\left(a_t|s_t\right) G_t\right]$, where $G_t = \sum_{j=0}^{T-t} \gamma^j \tilde{r}^h$. Thus we update the high-level policy parameters by $\theta \leftarrow \theta + \alpha \gamma^t \nabla_\theta \log \pi^h_\theta\left(a_t|s_t\right) G_t$.

### Low-level RL with Action Branching

The low-level problem is considered as a discrete control problem with two action dimensions: price and quantity. We utilize the Branching Dueling Q-Network (Tavakoli, Pardo, and Kormushev 2018). Formally, we have two action dimensions with $|p^l| = n_p$ discrete relative price levels and $|q^l| = n_q$ discrete quantity proportions. The action value $Q^l_d$ at state $s^l \in \mathcal{S}^l$ and the action $a^l_d \in \mathcal{A}^l_d$ are expressed in terms of the common state value $V^l(s)$ and the corresponding (state-dependent) action advantage $Adv^l_d\left(s, a^l_d\right)$ for $d \in \{p, q\}$:

$$Q_d^l(s^l, a_d^l) = V(s^l) + (Adv_d(s^l, a_d^l) - \frac{1}{n} \sum_{a_d^{l'} \in \mathcal{A}_d} Adv_d(s^l, a_d^{l'}))$$

We train our Q-value function based on the one-step temporal-difference learning:

$$y_d = r + \gamma Q_d^-(s^{l'}, \arg\max_{a_d^{l'} \in \mathcal{A}_d} Q_d(s^{l'}, a_d^{l'})), d \in \{p, q\} \quad (5)$$

$$L = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}}[\frac{1}{N} \sum_{d \in \{P,I\}} (y_d - Q_d(s^l, a_d^l))^2] \quad (6)$$

where $\mathcal{D}$ denotes a (prioritized) experience replay buffer and $a$ denotes the joint-action tuple $(p^l, q^l)$.

## Training Scheme

The financial data, including price information, multiple factors, and limit order book, is complex and high-dimensional. In order to use the available data more efficiently, we utilize the pre-training scheme and the iterative training.

**Pre-training.** In this work, we pre-train the low-level policy in the limit order book environment. Specifically, we first pre-select several assets, i.e., stocks, as our pre-trained data sources. By individually training the corresponding buying/selling policy of each asset, we derive multiple low-level policy parameters for these assets. In the general setting of hierarchical reinforcement learning, the high-level policy and the low-level policy are only trained together in a single environment. However, this training scheme suffers from data insufficiency, since the agent's interactions with the environment are severely restricted due to the joint training, especially for the low-level policy training. By introducing the pre-training scheme, the low-level policy would be trained with more and diverse interactions, thereby having better generalization and robustness.

**Iterative training.** Following the iterative training scheme in (Nevmyvaka, Feng, and Kearns 2006), we augment the private state repeatedly in the low-level pre-training. Specifically, we traverse the target quantity from $(0, 0)$ to a max target quantity $q_{max}$ and the remaining time from 0 to a max trading time window $T'_{max}$. In this way, the low-level policy is trained with the augmented private states, thus can generalize different subtasks assigned by the high-level policy.

## Experiment

In this section, we introduce our data processing, baselines, and metrics. Then we evaluate our algorithm with baselines in different markets. Lastly, we analyze the impact of entropy in reward and the trading cost by ablation study.

## Dataset Setting and Preprocessing

Our experiments are conducted on stock data from the U.S. stock market and China stock market. The stock data are collected from wind[2]. We evaluate our algorithm on different stock markets to demonstrate robustness and practicality. We choose stocks with large volumes so that our trading

[2]https://www.wind.com.cn/

|  | the U.S. Market | the China Market |
|---|---|---|
| Training | 2000/01/03-2014/01/03 | 2007/02/08-2015/07/03 |
| Evaluate | 2014/01/06-2015/12/31 | 2015/07/06-2017/07/03 |
| Test | 2016/01/04-2018/05/22 | 2017/07/04-2018/07/04 |

Table 1: Period of stock data used in the experiments.

action would not affect the market price. Specifically, we choose 23 stocks from *Dow Jones Industrial Average Index (DJIA)* and 23 stocks from *SSE 50 Index* for the U.S. stock market and China stock market respectively. In addition, we introduce 1 cash asset as a risk-free choice for both of the two markets. Moreover, the period of stock data used in the experiments is shown in Table 1.

For the high-level setting, we set the holding period to 5 days and set the trading period to one day. We set the time window of the high-level state as 10 days. We normalize the stock data to derive a general agent for different stocks in data pre-processing. Specifically, we divide the opening price, closing price, high price, and low price by the close price on the first day of the period to avoid the leakage of future information. Similarly, we use the volume on the first day to normalize the volume. For missing data which occurs during weekends and holidays, we fill the empty price data with the close price on the previous day and set zero volume to maintain the consistency of time series . For the constant commission rate, we set it to $0.2\%$ on both two markets.

For the low-level setting, we set the time interval to 30 seconds. We set the time window of the low-level state as opening hours of one day. Similarly, we also normalize the prices in limit order books by dividing the first price at the beginning of the period, as well as the volume. For missing data which occurs during weekends and holidays, we fill the empty price data with the previous price and set zero volume to maintain the consistency of time series.

## Baseline and Metrics

**Baselines.** We compare our methods with the baselines:

- *Uniform Constant Rebalanced Portfolios (UCRP)* (Cover 2011) keeps the same distribution of wealth among a set of assets from day to day.

- *On-Line Moving Average Reversion (OLMAR)* (Li and Hoi 2012) exploits moving average reversion to overcome the limitation of single period mean reversion assumption.

- *Weighted Moving Average Mean Reversion (WMAMR)* (Gao and Zhang 2013) exploits historical price information by using equal weighted moving averages and then learns portfolios by online learning techniques.

- *Follow the Winner (Winner)* (Gaivoronski and Stella 2000) always focus on the outperforming asset and transfers all portfolio weights to it.

- *Follow the Loser (Loser)* (Borodin, El-Yaniv, and Gogan 2004) moves portfolio weights from the outperforming assets to the underperforming assets.

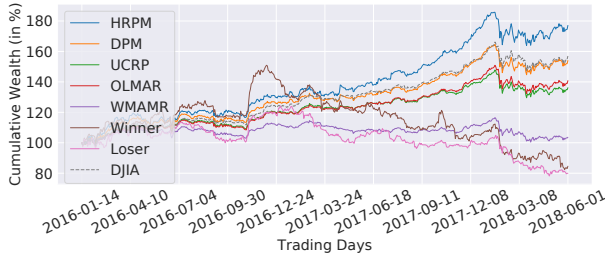- *Deep Portfolio Management(DPM)* (Jiang, Xu, and Liang 2017) is based on Ensemble of Identical Independent

Figure 3: The portfolio value in the U.S. market.

|  | ARR(%) | ASR | MDD | DDR |
|---|---|---|---|---|
| UCRP | 13.827 | 1.24 | 0.124 | 1.348 |
| Winner | -6.773 | -0.474 | 0.454 | -0.455 |
| Loser | -8.961 | -0.911 | 0.371 | -0.723 |
| OLMAR | 15.428 | 1.252 | 0.123 | 1.503 |
| WMAMR | 1.503 | 0.468 | 0.138 | 0.143 |
| DPM | 19.864 | **1.263** | 0.121 | 1.815 |
| **HRPM** | **27.089** | 1.246 | **0.117** | **2.403** |
| *DJIA* | *20.823* | *1.206* | *0.116* | *2.547* |

Table 2: Performance comparison in the U.S. market



Figure 4: The portfolio value in the China market.

|  | ARR(%) | ASR | MDD | DDR |
|---|---|---|---|---|
| UCRP | 23.202 | 2.917 | 0.136 | 1.863 |
| Winner | -13.6 | -1.201 | 0.368 | -0.705 |
| Loser | -17.645 | -2.1 | 0.293 | -1.045 |
| OLMAR | 24.947 | 3.252 | 0.122 | 1.961 |
| WMAMR | 15.873 | 2.485 | 0.137 | 2.775 |
| DPM | 35.81 | 3.126 | 0.153 | 2.404 |
| **HRPM** | **48.742** | **3.813** | **0.131** | **3.325** |
| *SSE50* | *6.081* | *1.1* | *0.172* | *0.641* |

Table 3: Performance comparison in the China market

Evaluators (EIIE) topology. *DPM* uses asset prices as state and trains an agent with a Deep Neural Network (DNN) approximated policy function to evaluate each asset's potential growth in the immediate future and it is the state-of-the-art RL algorithm for PM.

In addition, *SSE 50 Index* and *Dow Jones Industrial Average Index (DJIA)* are introduced as baselines in the comparison of performance to demonstrate whether these portfolio strategies are able to outperform the market.

**Metrics.** we use the following performance metrics for evaluations: *Annual Rate of Return (ARR)*, *Annualized Sharpe Ratio (ASR)*, *Maximum DrawDown (MDD)* and *Downside Deviation Ratio (DDR)*.

- *Annual Rate of Return (ARR)* is an annualized average of return rate. It is defined as $ARR = \frac{V_f - V_i}{V_i} \times \frac{T_{year}}{T_{all}}$, where $V_f$ is final portfolio value, $V_i$ is initial portfolio value, $T_{all}$ is the total number of trading days, $T_{year}$ is the number of trading days in one year.

- *Annualized Sharpe Ratio (ASR)* is defined as *Annual Rate of Return* divided by the standard deviation of the investment. It represents the additional amount of return that an investor receives per unit of increase in risk.

- *Maximum DrawDown (MDD)* is the maximum observed loss from a peak to a bottom of a portfolio, before a new peak is attained. It is an indicator of downside risk over a specified time period.

- *Downside Deviation Ratio (DDR)* measures the downside risk of a strategy as the average of returns when it falls below a *minimum acceptable return (MAR)*. It is the risk-adjusted ARR based on Downside Deviation. In our experiment, the *MAR* is set to zero.
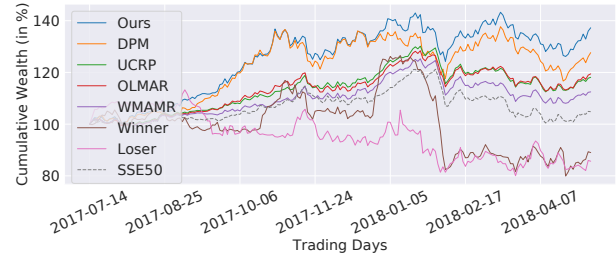
## Comparison with Baselines

**Backtest Results in the U.S. Market.** Figure 3 shows the cumulative wealth vs. trading days in the U.S. market. From the plots of *DJIA* index, we can see this period is in a bull market generally, although the market edges down several times. While simple methods like *Following the Winner* and *Following the Loser* fail, the portfolio values conducted by most of the strategies keep climbing but still underperform the market. Particularly, our *HRPM* is the only strategy that outperforms the *DJIA* index, when *DPM* keeps almost the same as the trend of the market.

We can compare the metrics of different strategies in detail in Table 4. On *ARR*, *HRPM* performs much better than the other methods and outperforms the market with 6.2%. On *ASR*, most of methods get higher scores than *DJIA* index and *HRPM* is the best. That is, our strategy could gain more profit under the same risk. When it goes to *MDD* and *DDR*, the results show that *HRPM* bears the least risk, even lower than *UCRP*. The risk of our method is acceptable, although a little higher than the *DJIA* index.

**Backtest Results in the China Market.** Figure 4 shows cumulative wealth vs. trading days in the China market. As we can see, *SSE50* index first rises slowly, but from then on it falls back to the starting point. During the rising stage, *HRPM* keeps the best performance and the longest time. When the portfolio values of other methods decline, our strategy still hovers at the peak. This phenomenon demonstrates that *HRPM* is not only superior to all the baselines, it is also robust under different market conditions relatively.

Table 3 compares all metrics of different strategies in detail. On all the four metrics, our *HRPM* achieves the best among all the strategies. Specifically, *HRPM* outperforms *DPM* by 13% and outperforms market by 42% on *ARR*.
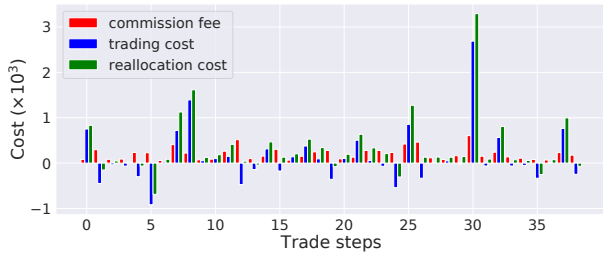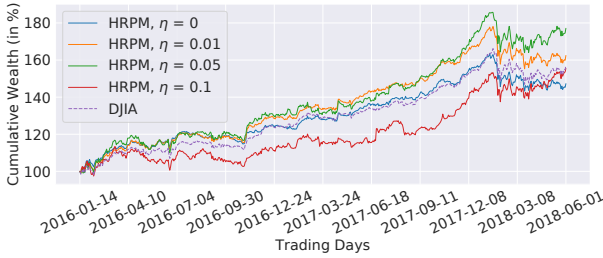
Figure 5: Trading cost during the trading periods.



Figure 6: The effect of the entropy bonus in the U.S. Market.

| | ARR(%) | ASR | MDD | DDR |
|---|---|---|---|---|
| HRPM, $\eta = 0$ | 17.092 | 1.207 | 0.191 | 1.631 |
| HRPM, $\eta = 0.01$ | 25.172 | 1.201 | 0.183 | 2.275 |
| HRPM, $\eta = 0.05$ | 27.089 | 1.246 | 0.117 | 2.403 |
| HRPM, $\eta = 0.1$ | 20.33 | 1.346 | 0.124 | 1.754 |
| *DJIA* | *20.823* | *1.206* | *0.116* | *2.547* |

Table 4: Ablation on the effect of entropy in the U.S. market

Even considering return with risk by the *ASR* metric, our strategies work better than others. Different from the performance in the U.S. market, *HRPM* not only reaches the least risk among these strategies but also performs better than the market on *MDD* and *DDR*. It means that our strategy is able to adapt to volatile markets with low risk.

### Ablation Study

In our ablation study, we mainly answer two questions: (1) why the commission fee is not enough to account for the actual trading cost? (2) Does the entropy bonus work?
**Commission Fee is Not Enough.** In Fig. 5, we show the trading cost at each trading period of testing in the China market. The x-axis is the index of each trading period in testing, and the y-axis is the cost. Note that a negative trading cost means gaining profit in the trading period. A negative trading cost happens when the submitted orders are finished at a better price than the target price. We can see that the commission fee is only a small part of the total trading cost in most transactions, especially in large transactions.
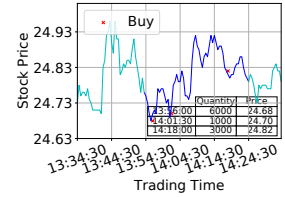**Entropy Bonus.** We show the effect of the entropy bonus in
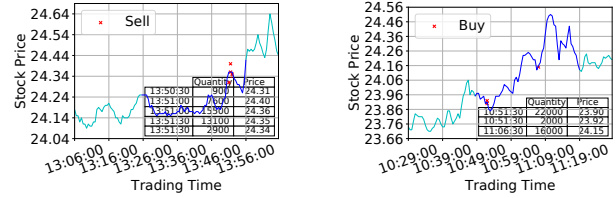


(a) Selling 40,000 shares



(b) Buying 40,000 shares
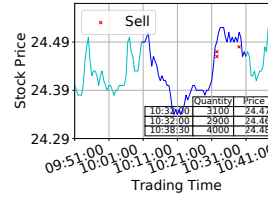


(c) Selling 10,000 shares



(d) Buying 10,000 shares

Figure 7: The interpretation of the low-level policy. The line is the mid-price of the market. The dark blue part is the trading window, and the light blue part shows the price before and after. The table inside the figure shows the finished trades.

Fig. 6. As $\eta$ in the entropy bonus term in reward gets higher, the agent would tend to give portfolios with more diversity. While $\eta$ is small, the agent might be more likely to hold only a few stocks. Consequently, we may get a balance between the risk-control and profit by controlling $\eta$. As Table 4 shows, *HRPM* with $\eta = 0.05$ reaches the maximum ARR. Note that *HRPM* with $\eta = 0.1$ has the lowest MDD than others. By increasing the effect of the entropy, we can gain more profit relative to the risk.

### Trading Strategy Interpretation

Here, we try to interpret the underlying investment strategies of the low-level policy. We show the trades that happened in the trading period in Fig. 7. Fig. 7(a)(c) shows that the low-level policy tries to sell orders at a relatively high position. It first sells part of the quantity at the beginning of the trading period. Then it sells the rest at the following local maximum. In this way, the low-level policy can avoid missing potential growth and decline. We can get the same conclusion for buying in Fig. 7(b)(d).

## Conclusion

In this paper, we focus on the problem of portfolio management with trading cost via deep reinforcement learning. We propose a hierarchical reinforced stock trading system (*HRPM*). Concretely, we build a hierarchy of portfolio management over trade execution and train the corresponding policies. The high-level policy gives portfolio weights and invokes the low-level policy to sell or buy the corresponding shares within a short time window. Extensive experimental results in the U.S. market and China market demonstrate that *HRPM* achieves significant improvement against many state-of-the-art approaches.

# Acknowledgements

# References

Almahdi, S.; and Yang, S. Y. 2017. An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. *Expert Systems with Applications* 87: 267–279.

Borodin, A.; El-Yaniv, R.; and Gogan, V. 2004. Can we learn to beat the best stock. In *Advances in Neural Information Processing Systems*, 345–352.

Cover, T. M. 2011. Universal portfolios. In *The Kelly Capital Growth Investment Criterion: Theory and Practice*, 181–209. World Scientific.

Gaivoronski, A. A.; and Stella, F. 2000. Stochastic nonstationary optimization for finding universal portfolios. *Annals of Operations Research* 100(1-4): 165–188.

Gao, L.; and Zhang, W. 2013. Weighted moving average passive aggressive algorithm for online portfolio selection. In *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, volume 1, 327–330. IEEE.

Jiang, Z.; and Liang, J. 2017. Cryptocurrency portfolio management with deep reinforcement learning. In *2017 Intelligent Systems Conference (IntelliSys)*, 905–913. IEEE.

Jiang, Z.; Xu, D.; and Liang, J. 2017. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059* .

Li, B.; and Hoi, S. C. 2012. On-line portfolio selection with moving average reversion. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, 563–570.

Liang, Z.; Chen, H.; Zhu, J.; Jiang, K.; and Li, Y. 2018. Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940* .

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* .

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* .

Mosavi, A.; Ghamisi, P.; Faghan, Y.; and Duan, P. 2020. Comprehensive Review of Deep Reinforcement Learning Methods and Applications in Economics. *arXiv preprint arXiv:2004.01509* .

Nevmyvaka, Y.; Feng, Y.; and Kearns, M. 2006. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd International Conference on Machine Learning*, 673–680.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587): 484.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 1057–1063.

Tang, L. 2018. An actor-critic-based portfolio investment method inspired by benefit-risk optimization. *Journal of Algorithms & Computational Technology* 12(4): 351–360.

Tavakoli, A.; Pardo, F.; and Kormushev, P. 2018. Action branching architectures for deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 4131–4138.

Ye, Y.; Pei, H.; Wang, B.; Chen, P.-Y.; Zhu, Y.; Xiao, J.; and Li, B. 2020. Reinforcement-learning based portfolio management with augmented asset movement prediction states. *arXiv preprint arXiv:2002.05780* .

Yu, P.; Lee, J. S.; Kulyatin, I.; Shi, Z.; and Dasgupta, S. 2019. Model-based deep reinforcement learning for dynamic portfolio optimization. *arXiv preprint arXiv:1901.08740* .

Zhang, J.; and Tao, D. 2020. Empowering Things with Intelligence: A Survey of the Progress, Challenges, and Opportunities in Artificial Intelligence of Things. *IEEE Internet of Things Journal* doi:10.1109/JIOT.2020.3039359.