上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS

论文题目：TRADING STRATEGIES BASED ON REINFORCEMENT LEARNING IN A-SHARE MARKET

学生姓名：何静海
学生学号：518020910068
专　　业：金融学
指导教师：花成
学院（系）：安泰经济与管理学院

教务处制表

上海交通大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文《Trading Strategies Based on Reinforcement Learning in A-share Market》，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：何静海

日期： 2022 年 5 月

上海交通大学
学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□，在___年解密后适用本授权书。

本学位论文属于

不保密□。

（请在以上方框内打“√”）

学位论文作者签名：何静海　　指导教师签名：花成

日期：2022　年 5 月　7 日　　日期：2022　年 5 月 7　日

基于强化学习的 A 股市场交易研究

摘要

设计交易策略是投资研究的核心问题之一。在现代机器学习算法和日益增进的计算能力的帮助下，基于机器学习的算法交易和投资组合管理逐渐获得了行业与学术界的关注与重视。然而，仅仅凭借基于回归和聚类的传统静态机器学习算法，在复杂和动态的股票市场中设计一种普遍盈利的策略仍然是极具挑战的。

强化学习 (RL) 是一种旨在通过与提供奖励的环境交互进而找到最佳策略的算法。它强调不断通过与未知环境的交互，习得在不同情境下的最优反应策略。这一过程与交易者在股票市场中在不同的行情下通过做出多空决策以最优化收益的行为不谋而合。

在本文中，我们将强化学习算法应用于A股市场的股票交易，通过不断与环境进行交互从而学习交易策略，以实现收益最大化。我们首先将股票市场的交易过程建模为马尔可夫决策过程 (MDP)，并使用六种成熟的强化学习算法分别训练：TD3, DDPG, PPO, DDQN, A2C与SAC。此外，考虑到不同的算法分别具有乐观与悲观的特性，为了获得更稳健的交易算法，我们探索了将上述算法进行基于最优策略集成和基于模型组合的集成（BMC）交易算法。

我们分别通过在上证 50 (SEE.50)、沪深 300(CSI.300)和中证 500(CSI.500)成分股的训练与回测交易中评估和比较我们的策略。通过与基准收益 ETF（SEE.50, CSI.300 与 CSI.500 指数）以及市场中的可交易指数型基金比较，我们发现大多数强化学习算法在收益上都可以得到大幅度超过基准的回报与较高的夏普比率。此外，我们引入了加权集成的算法以提升投资回报的稳健型。我们发现相比其他机器学习算法，集成算法可以得到更低的最大回撤与波动率。这为 A 股市场量化交易与投资组合研究提供了额外思路与补充，并拓展了多决策算法集成在中国市场中的应用。

关键词：算法交易，强化学习，马尔可夫决策，集成学习，A 股市场

TRADING STRATEGIES BASED ON REINFORCEMENT LEARNING IN A-SHARE MARKET

ABSTRACT

Designing trading strategies is one of the core issues in investment research. With the help of modern machine learning algorithms and exponentially growing computing capacity, algorithmic trading and portfolio management based on statistical learning are gradually gaining popularity and empirical success. However, it remains thought-provoking to design a universally profitable strategy in complex and dynamic stock markets using traditional machine learning algorithms built primarily on regression and clustering.

Reinforcement Learning (RL) is a set of algorithms aiming to find the optimal strategies for providing rewards from interacting with the environment. It focuses on learning and exploring an unknown environment with feedback on agents' actions and exploiting the best strategy. This process is compatible with a trader making long-short decisions in the stock market and gaining profit from his/her actions.

In this paper, we apply reinforcement learning algorithms to the A-share market and generate trading strategies to maximize total returns. We model the trading process in the stock market as a Markov Decision Process and train RL agents separately using six algorithms: Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Dual Deep Q-Network (DDQN), Advantage Actor-Critic (A2C), Asynchronous Advantage Actor-Critic(A3C) and Soft Actor-Critic (SAC). Besides, in order to obtain more robust trading algorithms, we ensemble these algorithms based on Bayesian optimal strategy and Bayesian model combination (BMC).

We evaluate and compare the returns of our strategies with the baseline ETFs (SEE.50, CSI.300, and CSI.500) and index-based funds in the market. We find that most of our RL agents outperform the baselines in returns, and the ensembling methods are more robust in terms of Sharp Ratio and maximum drawdown. Our work may contribute to the field of algorithmic trading and ensembling-based portfolio management in the A-share market.

Key words: Algorithmic Trading, Reinforcement Learning, Markov Decision Process, Model Ensembling, A-Share Market

Content

Chapter One Introduction

Stock trading can be simplified as longing and shorting companies' shares in the financial market to maximize the return on the investment. When deciding how many shares to buy or sell a piece of stock, one of the main challenges is to model and predict the complex and dynamic price series, in which stiff trading strategies from so-called experts always fail to make stable profits. From traditional financial modeling, designing and testing asset pricing models are the main focus for trading strategies. Some well-known ones include the capital asset pricing model (CAPM) [1] and Fama & French factor model [2]. On the other hand, in computer science and statistics, applying data-driven algorithms, especially machine learning techniques, to analyze financial data is the mainstream [3]. Moreover, deep learning methods that implement neural networks in finance have recently become appealing due to their attractive ability to dig out meaningful representations and accurate predictions [4].

Reinforcement learning (RL) is an emerging branch of statistical learning and machine learning algorithms. Unlike supervised and unsupervised learning, RL algorithms are designed to generate the optimal strategy that maximizes the expected reward in a dynamic and stochastic environment [5]. The last decade has witnessed many significantly successful RL approaches in various domains such as recommendation systems [6], games [7], and robotics [8]. One of the most famous milestones might be AlphaGo [9], who beats the most talented human players in the game of Go.

Ensembling methods can apply multiple learning algorithms to obtain better and more robust performance than could be obtained from any of the constituents alone [10]. This methodology can derive from collective wisdom, which indicates that making a

decision based on different opinions can achieve a better result. Successful approaches like Bootstrap Aggregating (Boosting) [11], Adaptive Bootstrap (Adaboost) [12], and Random Forest [13] all outperform their component algorithms.

There are mainly four reasons why RL can help for better trading. Firstly, the aim of designing optimal strategies under a non-deterministic situation resembles the go of making long-short decisions. Secondly, RL trains an end-to-end agent who utilizes available market information as input and trading actions as output. This feature can bypass the challenging task of predicting future prizes used in the traditional predicting-then-executing modeling. Thirdly, RL-based methods optimize overall (discounted) profit directly. The discount factor in RL considers the cost of time, which aligns with the risk-free rate in financial modeling. Lastly, based on the function approximation method in RL, it is possible to generalize any market condition which requires extensive dimensional data [14].

This paper aims to design and compare the latest RL algorithms' performances in the A-share market, a market on which most past literature focuses little. Then we will combine the strategies of different algorithms based on optimal and weighted ensembling methods and search for any improvements in RL algorithms' collective wisdom. To the best of our knowledge, there are very few thorough tests of RL algorithm applications in the A-share market, especially considering ensembling methods.

Our paper is organized according to the following structure. In Chapter Two, we review some important literature closely related to our topic; in Chapter Three, we describe our modeling of the trading process as an RL framework; in Chapter Four, we propose the main algorithms used and the approaches to combine them; and in Section Five, we apply the proposed algorithms to trading and backtesting in A-share market; in the last part, we summarize our results and discuss some potential improvements for future research.

Chapter Two Related Works

Our works are generally related to three lines of literature: reinforcement learning, algorithmic trading, ensembling methods, and machine-learning-assisted trading.

2.1 Reinforcement Learning

Reinforcement learning is a popular subfield of statistical learning that studies complex control and decision-making problems. In Sutton and Barto's description [15], RL problems usually possess a closed-loop problem, an agent figuring out decisions by trial-and-error, and actions impacting short-term and long-term results. We typically call the decision-maker an agent and call everything except the agent the environment. The detailed formulations of RL will be discussed in Section Three.

Many algorithms have been proposed to solve RL problems, and generally, we can divide them into tabular and approximation methods. The value function for every action-state pair is presented in a tabular for tabular algorithms, and the agent may act according to the optimal decisions by checking the table.

Dynamic programming (DP) [16], Monto Carlo (MC)[17] and temporal difference (TD) [18] prove efficient in dealing with tabular settings. However, this tabular modeling suffers from the dimensional curse and can hardly work in a high-dimensional environment such as the financial market. Instead, function approximation methods aim to find a great approximate function of high-dimensional data. In approximation RL algorithms, we aim to generalize from previous experiences to unexplored states. Policy gradient methods such as Natural policy gradient [19], actor-critic [20], and two variants of actor-critic [21,22] gain great reputations for generalization ability. Further, with deep learning, RL with neural networks working as function approximators lead to great success in many domains like AlphaGo [9] in chess playing. Other famous

deep-learning-based RL algorithms include Q-network (DQN) [23], deep deterministic policy gradient (DDPG) [24], proximal policy optimization (PPO) [25]. Our paper mainly applies approximation-based RL algorithms in the finance field, famous for their high dimensionality.

2.2 Algorithmic Trading

Algorithmic trading is the process in which traders consistently make long-short decisions following a planned rule, given a set of financial assets aiming to maximize profits. Broadly speaking, any financial assets can be traded based on algorithmic. Based on trading frequency and style, algorithmic trading includes five categories [26]: position trading (long-time-holding), swing trading, day trading, scalp trading (short while every day), and high-frequency trading (tick level). According to the trading regulations, day trading is more realistic in the A-share market.

Traditional algorithmic trading is generally based on time-series analysis. For example, momentum strategies like Times Series Momentum [27] and Cross-Sectional Momentum [28], and mean-reversion strategies such as Bollinger bands [29] are all based on historical price patterns. However, some problems lie in most conventional algorithmic trading strategies. Among them, the most significant is the lack of generalization ability among different markets, concerning only a small fraction of assets, and the inability to deal with long-term and periodic patterns.

2.3 Machine-Learning-Assisted Trading

Machine learning (ML) refers to a large family of computer algorithms that can improve automatically through training and data. Ideally, financial data suit ML for its abundancy and clearness, and there have already been quite a lot of works applying ML in finance for different purposes. For example, Principal Component Analysis (PCA) [30] and Deep Neural Networks [31] are applied to extract features and patterns of stock

markets. For price forecasting, researchers have implemented different methods in financial data, from naïve algorithms (SVM [32], Lasso [33], and random forest [34]) to intricate neural networks (MLP [35], RNN [36], CNN [37] and LSTM). What is noticeable is that Recurrent Neural Networks (RNN) and its extension Long-Short-Term Memory (LSTM) neural networks are incredibly successful in time-series predicting.

Specifically related to our works is RL-assisted reinforcement learning. Recurrent reinforcement learning (RRL) [38] proves to have stable performance when exposed to noisy data such as financial data, and its extension adaptive RNN [39] outperforms most baselines Eur-US dollar exchange market. Trust Region Volatility Optimization (TRVO) [40] proposed based on the risk-averse purpose for option hedging beats traditional Black-Scholes delta strategies on simulated option price trajectories.

Chapter Three Modeling of Stock Trading in RL Frameworks

This section discusses how to present the trading process in the stock market as a Markov Decision Process (MDP), which is the basic framework for reinforcement learning. We first review some important concepts and conclusions for reinforcement learning; then, we present how to present the market information and long-short positions in an RL pattern.

3.1 Basics of Reinforcement Learning

We usually model the environment for RL as a Markov Decision Process with reward, $M = \{S, A, P, r, \gamma\}$. S is the set of possible states of the environment and in a financial setting. For a specific state $s \in S$, we can understand this as the current prices of all stock shares in the market. A is the feasible action set, and for an action $a \in A$, we can regard it as trading decisions like longing specific company's shares for a certain amount. $P(s, a)$ is the transition probability matrix describing system dynamics. Specifically, in an MDP, the distribution of state in step $t + 1$ only depends on (s_t, a_t), the state-action pair in step t. We formally present this Markovian property as:

$$P(\mathrm{s_{t+1}} \mid s_t, a_t, \mathrm{s_{t-1}}, \mathrm{a_{t-1}}, \dots \mathrm{a_1}, \mathrm{s_0}) = P(\mathrm{s_{t+1}} \mid s_t, a_t). \tag{3.1}$$

After taking action a_h in the t^{th} step, the agent will receive a reward (maybe even a stochastic one) based on s_t, s_{t-1}, a_t, $r_t = r(s_t, s_{t-1}, a_t)$. The last component γ is the discount factor. It is the same as the discounted factor in the calculation of net present value (NPV). In reinforcement learning, the cumulated reward in an epoch of game is discounted summation of reward in each step for a long time period with a length of T : $\sum_{k=0}^{T} \gamma^k r_{t+k+1}$. The aim of MDP is to maximize the (discounted) cumulative reward.

Besides these environmental factors, we always use a probability vector $\pi(a|s)$ to present the strategy, which means taking action a (with probability) $\pi(a|s)$ under state s. In market settings, we can illustrate policy ($\pi(a_t|s_t)$) as: "when given all the stock prices at time t (s_t)" "a trader would long certain stocks and short others." (a_t). Informally speaking, the main goal of RL is to find the policy π (long-short strategies), which could optimize the long-term return $\sum_{k=0}^{T} \gamma^k r_{t+k+1}$ (discounted market returns). In Figure 1, we present a brief loop of RL in finance.

For simplicity, in the following parts, we use $P_{ss'}^{a}$ to represent transition probability $P(s'|s,a)$, $r_{ss'}^{a}$ for the reward $r(s',s,a)$.

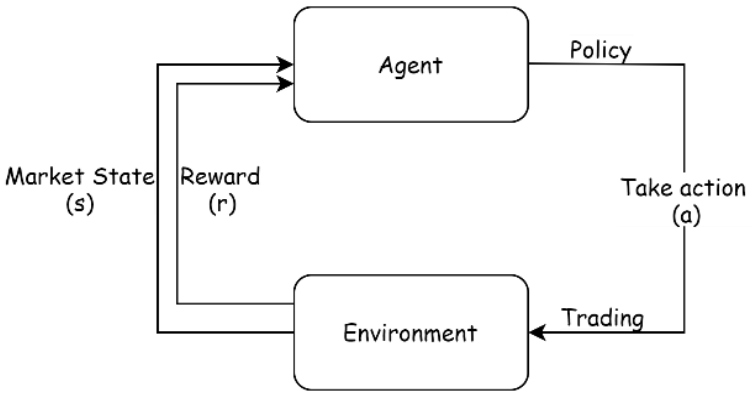


Figure 1.1: Structure of RL in Finance

Moreover, in MDP analysis, we usually introduce two critical measures of a policy π: $Q^{\pi}(s,a)$ and $V^{\pi}(s)$.

Definition 1(State Value Function). Given a state s and time t, the value of the state at t under a fixed policy π is the expected return of starting in the given state and then following the policy.

$$V^{\pi}(s) = E_{\pi}[R_t \mid s_t = s] = E_{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s\right]. \tag{3.2}$$

Definition 2(Action Value Function). Given a state s and action a at time t, the value of a state-action pair at t under a fixed policy π is the expected return of

starting in the given state, taking the given action, and then following the policy.

$$Q^{\pi}(s,a) = E_{\pi}[R_t \mid s_t = s, a_t = a] = E_{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right]. \quad (3.3)$$

By simple mathematical tricks, we can calculate the value function $V^{\pi}(\cdot)$ recursively according to Bellman Equation listed in Lemma 1.

Lemma 1 (Bellman Equation). For a fixed policy π, we have the following recursive relationship of the state value function $V^{\pi}(s)$ and state-action value function $Q^{\pi}(s,a)$

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in \mathcal{S}} \mathrm{P}_{ss'}^{a}\left[\mathrm{r}_{ss'}^{a} + \gamma V^{\pi}(s')\right]$$

$$= E_{\pi}[r_t + \gamma V^{\pi}(s_{t+1}) | s_t = s\ ].$$

$$= \sum_{a \in A} \pi(a|s) Q^{\pi}(s,a), \qquad \forall\, t \in Z^{+}. \quad (3.4)$$

Similarly, we can also break the $Q(s,a)$ into:

$$Q^{\pi}(s,a) = E[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | s_t = s, a_t = a]$$

$$= r_t(s,a) + \gamma \sum_{s' \in S} P_{ss'}^{a} V^{\pi}(s') \quad (3.5)$$

For small-scale problems like tabular MDP, if the model $M = \{S, A, P, r, \gamma\}$ is provided, we can directly compute V^{π} and Q^{π} based on the transition probabilities and expected reward dynamics. For larger problems, especially continuous state and action space problems like in the finance environment that we discussed in this paper, it is impossible to visit every state and calculate the value functions. Therefore, we rely on function approximations of the Q and V functions through parameter updating. Moreover, in continuous settings, the Bellman Equation can still work:

Corollary 1 (Integrated Bellman Equation) For a fixed policy π, we have the following recursive relationship of the value function:

$$V^{\pi}(s) = \int_{a \in A(s)} \pi(s,a) \int_{s' \in S} \mathrm{P}_{ss'}^{a} \left[\mathrm{r}_{ss'}^{a} + \gamma V^{\pi}(s') \right] ds' da. \tag{3.6}$$

In the financial setting, states (such as the price of specific stocks) can be modeled as continuous. One of the main differences in the algorithms we apply lies in presenting and calculating the aforementioned Bellam Equation. The algorithms we may include in the paper like Asynchronous Advantage Actor-Critic (A3C), Deep Q-Networks (DQNs), Deep Deterministic Policy Gradient (DDPG), and Evolution Strategies (ES), all have their unique ways of searching for the representation of value function and derive the optimal policy π^{*} to maximize $V^{\pi^{*}}(s_0)$. Once given the numerical representation of financial markets, we can apply them directly to stock trading.

Another important property for MDP and reinforcement learning is the Bellman Optimality Equation [15]. This theorem provably secures the existence of the optimal policy and value function. Formally, the Bellman Optimality Equation can be written as follows.

Theorem 1 (Bellman Optimality Equation) There exists an optimal policy $\pi^{\star}$, allowing the maximization of V-type and Q-type value functions at the same time. Formally written, we have that:

$$\pi^{\star} = \arg\max_{\pi \in \Pi} V^{\pi}(s) = \arg\max_{\pi \in \Pi} Q^{\pi}(s,a) \tag{3.7}$$

This equation also indicates the following relation under optimal policy $\pi^{\star}$:

$$V^{\pi^{\star}}(s) = \max_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} P_{ss'}^{a} V^{\pi^{\star}}(s') \right); \tag{3.8}$$

$$Q^{\pi^{\star}}(s,a) = r(s,a) + \gamma \sum_{s' \in S} P_{ss'}^{a} \max_{a' \in A} Q^{\pi^{\star}}(s',a'). \tag{3.9}$$

Bellman Optimality Equation shows the existence of the optimal strategy. It also provides us with the approach to finding the optimal $\pi^\star$ by finding the maximized representation of the value function $Q^{\pi^\star}(s,a)$ or $V^{\pi^\star}(s)$.

3.2Formulation of Stock Trading

Based on the stochasticity and interactivity in the trading process, we model our decision and the financial market as a Markov Decision Process (MDP), as shown in Figure 1.1. We depict the market information and our asset as state and the long-short decision as to the action in an MDP. For generality, we consider D stocks ($D \leq |Market|$) to trade. To better link our model with the real market, we also consider current restrictions on the A-share market. Therefore, in our model, leverage and short selling are not allowed.

The state $s = [\boldsymbol{p}, \boldsymbol{h}, \boldsymbol{i}, \boldsymbol{b}]$ is a set that includes the information on the prices of stocks $\boldsymbol{p} \in R_+^D$, the number of holdings of stocks $\boldsymbol{h} \in Z_+^D$, $\boldsymbol{i}$ is an auxiliary part incorporating other market information (e.g., market index, technical indicators, and exchange ratio), and the remaining balance $b \in R^+$.

The action $\boldsymbol{a} \in A^D$ is a vector of actions on all D stocks. The available actions of each stock include *selling*, *buying,* and *holding*; the action will directly influence the holding position h in state s. There will be some restrictions on the actions allowed to take under different s:

1. ***selling***: if the trader chooses to sell $k \in Z^+$ shares of stock d, the state will change to $\boldsymbol{h_{t+1}}[d] = \boldsymbol{h_t}[d] - k$. However, since we do not allow short selling in A-share, $k \in [1, \boldsymbol{h_t}[d]], \forall d \in [1,2,\cdots,D]$.

2. ***holding***: if the trader chooses to take a holding position on stock d, $\boldsymbol{h_{t+1}}[d] = \boldsymbol{h_t}[d]$.

3. ***buying***: if the trader chooses to buy k shares of stock d can be bought, and it

leads to $\boldsymbol{h_{t+1}}[d] = \boldsymbol{h_t}[d] + k, \forall d \in [1,2\cdots,D]$.

For notational simplicity, we can write $\boldsymbol{a_t} = \boldsymbol{k_t} \in Z^D$ as the position action vector. If we buy in stock $d, \boldsymbol{k_t}[d] > 0$, and for selling, $\boldsymbol{k_t}[d] < 0$.

The reward $r(s, a, s^{\text{next}})$ directly comes from the change of the portfolio value when action a is taken at state s and observing the new state s^{next}. The value is the portfolio is $p^\top h + b$, which is the sum of the value and cash of the stock. We also consider the transaction cost c_t to make our model closer to the real world. Specifically, the return of the action-state pair (s_t, a_t, s_{t+1}) can be written as:

$$r(s_t, a_t, s_{t+1}) = (b_{t+1} + p_{t+1}^\top h_{t+1}) - (b_t + p_t^\top h_t) - c_t, \tag{3.6}$$

and the transaction cost c_t here can be calculated by the sum of additional costs induced by selling and purchasing based on the commission ratio (usually around 0.1%~1%).

Moreover, since leverage is not allowed in our model, the cash should follow the restriction equation: $b_{t+1} = b_t + (p_t^\top h_t)^{sell} - (p_t^\top h_t)^{buy}$, where $(\cdot)^{sell}$ indicate the inner product of the selling fraction and $(\cdot)^{buy}$ indicating the purchased fraction.

Under the aforementioned setting, the policy $\pi(a_t|s_t)$, can be understood as when observing the market information at period t, how many shares of specific stocks to purchase or sell. Based on Q-learning and Bellman Optimality Equation, which is the most popular pattern of training modern reinforcement learning, we can try to find the π to maximize $Q_\pi(s_t, a_t)$. Thus, we can rewrite our training purpose as finding the trading strategy π to maximize the value function:

$$\max_{\pi \in \Pi} Q_\pi(s_t, a_t) = E_{s_{t+1}}\Big[r(s_t, a_t, s_{t+1}) + \gamma E_{a_{t+1}}[Q_\pi(s_{t+1}, a_{t+1})]\Big]. \tag{3.7}$$

Following the deep reinforcement learning algorithms we will introduce in Chapter Four, we can use the output of a neural network to approximate the value of Q. Therefore, concluding the aforementioned settings, we can summarize our aim as:

approximating the value function of trading processes based on reinforcement learning algorithms and finding a nearly-optimal strategy.

Chapter Four Deep Reinforcement Learning Algorithms in Trading

In this chapter, we would like to discuss some vital reinforcement learning algorithms based on neural networks. Theoretically, they are applied to solve almost any MDP problems once the $M = \{S, A, P, r, \gamma\}$ is well formulated. The significant differences among them are the approximation and exploring strategies. Some famous RL algorithms include Advantage Actor-Critic (A2C), Soft Actor-Critic Algorithm (SAC), Deep Deterministic Policy Gradient (DDPG), Double Deep Q-Network (DDQN), Twined Delayed Deep Deterministic Policy Gradients (TD3), and Proximal Policy Optimization (PPO). All of these mentioned algorithms can achieve near-optimal strategy by updating $Q(s, a)$-function.

In the following subsection, we first present the general approach of approximating $Q_\theta(s, a)$, where θ is a group of parameters that decide the function Q. The main differences among those mentioned algorithms lie in the way they update θ and choose the policy π to explore the environment.

4.1 General Description of Model-Free RL, Q-learning, and Policy Gradient

Model-free RL is a set of RL algorithms learning the optimal policy through maximizing the value function. This set of algorithms does not impose assumptions on the transition dynamics (i.e., the transition probability of MDP) but directly learns the optimal mapping function $Q_\theta^\star(s, a)$ or $V^\star(s)$. The most representative ones among them are the Q-learning and policy gradient algorithms.

Q-learning, as its name suggests, is a set of RL algorithms to learn the best representation of the state-action value function $Q_\theta(s, a)$, where θ is the parameter denoting the structure of $Q(s, a)$. Based on Bellman Optimality Equation, under the

optimal policy $\pi^{\star}$, we have the relationship: $\forall(s,a,s'), Q_{\theta}^{\star}(s,a) = r(s,a) + \gamma \sum_{s' \in S} P_{ss'}^{a} \max_{a' \in A} Q_{\theta}^{\pi^{\star}}(s',a')$. It means that once we achieve the $Q^{\star}$, the optimal policy is to take greedy actions by $a^{\star}(s) = \pi^{\star}(s) = \arg_{a \in A} Q(s,a)$.

This equality property also suggests that we can iteratively update the theta in a certain way until convergence, reaching the $Q_{\theta}^{\star}$. The unique ways we update θ and construct Q^{π}, is the key to understanding different Q-learning algorithms. Generally speaking, we can update the Q-function as:

$$Q_{\theta_{t+1}}(s,a) = (1-\alpha)\, Q_{\theta_t}(s,a) + \alpha \left(r(s,a) + \gamma \max_{a' \in A} Q_{\theta_t}(s',a') \right), \tag{4,1}$$

where α represents the learning tendency to the local optimality. We can also rewrite Equation (4.1) in a parameterized representation:

$$\theta_{t+1} = \theta_t + \alpha \left(Y_t^{Q} - Q_{\theta_t}(s_t,a_t) \right) \nabla_{\theta_t} Q_{\theta_t}(s_t,a_t), \tag{4.2}$$

where the Y_t^{Q} is the optimization target of Q-function. For example, if taking a greedy updating strategy, we will take $Y_t^{Q} = r(s,a) + \gamma \max_{a} Q_{\theta_t}(s_{t+1},a)$, and learning rate α =1, and the (4.1) will be $Q_{\theta_{t+1}}(s,a) = r(s,a) + \gamma \max_{a' \in A} Q_{\theta_t}(s',a')$. In the following sections, we will discuss some more computationally efficient or more accurate algorithms than the greedy ones.

Another important branch of model-free algorithms is ***policy gradient*** algorithms [52]. They mainly aim to learn the representation of policy $\pi_{\theta}(a|s)$ (as well as the mapping function of the value function $Q^{w}(s,a)$). Sutton et al. [52] prove that under the assumption of $Q^{w}(s,a) = \nabla_{\theta} \log \pi_{\theta}(a|s)^{\top} w$, the expected loss is written as:

$$J_{\theta}(\pi_{\theta}) = E_{s \sim \rho^{\pi}, a \sim \pi_{\theta}} [Q^{w}(s,a) - Q^{\pi}(s,a)]^2, \tag{4.3}$$

can be minimized through gradient update:

$$\nabla_{\theta} J_{\theta}(\pi_{\theta}) = E_{s \sim \rho^{\pi}, a \sim \pi_{\theta}} [\nabla_{\theta} \log_{\theta} \pi(a|s)\, Q^{w}(s,a)]^2. \tag{4.4}$$

4.2 Deep Q-Network (DQN)

The Deep Q-network (DQN) algorithm aims to use a multi-layered neural network (or other structural networks) to approximate the value function $Q_\theta(s,a)$ for any given state-action pair (s,a) [46]. This algorithm is one of RL's first modern empirical successes by playing the game Atari, and its result was published in Nature. Its main algorithm is described below.

Algorithm: DQN with Experience Replay.
Initialize the buffer of replay memory D with capacity N
Initialize action-value function Q_θ with random weights θ_0
Initialize target action-value function $\hat{Q}_{\theta^-}$ with weights $\theta^- = \theta$
For episode $t = 1, \cdots, M$ **do**
  Initialize the first state s_1
  For $t = 1, \cdots, T$ **do**
    With a probability of ε select a random available action a_t; otherwise, select based on

$$a = \arg\max_{a \in A} Q_{\theta_t}(s_t)$$

    Execute a_t and observe r_t and s_t
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions $\{(s_j, a_j, r_j, s_{j+1})$ from D
    Set $y_j \begin{cases} r_j & if\ terminate \\ r_j + \gamma \max_{a'} \widehat{Q_{\theta^-}}(s_{j+1}, a') & else \end{cases}$.
    Perform gradient descent on squared loss $L_j(\theta) = \left(y_i - Q_\theta(s_j, a_j)\right)^2$ regarding θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

The key of DQN can be summarized as using one Q-networks with two sets of parameters θ^- (target) and θ (online) and experience replay. θ^- is the parameter of the target network and is updated comparatively slowly. This slow updating secures the convergence of gradient descent. Besides, the updating process can be understood as minimizing the squared loss each timestep by gradient:

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)\nabla_{\theta_i} Q(s, a; \theta_i)\right]. \quad (4.5)$$

Then after updating the online selection parameter θ for C timesteps, we will update the evaluation network $\theta^- = \theta$.

4.3 Double Deep Q-Network (DDQN)

As its name suggests, DDQN is an improved version of Deep Q-Network. The main aim of DDQN is to solve the overestimation induced by using one network to select an action and evaluate the value function. [50]. So DDQN decouples the evaluation and selection network into two different networks. In DQN, the evaluation target can be written as the output from the evaluation parameter:

$$Y_t^{DQN} = r(s_t, a_t) + \gamma \max_{a' \in A} Q_{\theta^-}(s, a'). \quad (4.6)$$

However, in DDQN, we iteratively update two networks:

$$Y_t^{Q} = r(s_t, a_t) + \gamma Q_{\theta_t}\left(s_t, \arg\max_{a \in A} Q_{\theta_t'}(s_t, a)\right) \quad (4.7)$$

$$Y_t^{DoubleQ} = r(s_t, a_t) + \gamma Q_{\theta_t'}\left(s_t, \arg\max_{a \in A} Q_{\theta_t}(s_t, a)\right) \quad (4.8)$$

This double learning strategy is proved to be efficient both theoretically [49] and empirically [50]. Intuitively, DDQN can be regarded as a pessimistic version of DQN.

4.4 Advantage Actor-Critic (A2C)

The advantage actor-critic algorithm was proposed by Bator et al. [51]. It is a typical policy gradient descent strategy, by learning the parameterized policy π_θ. This algorithm is constructed based on the temporal difference approach [18] to approximate V-type value function. Generally speaking, it employs the neural network approximation of $V_\phi(s)$ (critic) and $\pi_\theta(s)$ (actor), and uses an n-step temporal difference value to approximate the target of the value function. The primary approach

of A2C is shown in Figure 4.1.

The '***advantage***' in A2C indicates the ***advantage function***, which can be understood as the difference between the current value function and the target value. In fact, this advantage function has already frequently appeared in previously mentioned algorithms in the form of $Y_t^Q - Q_\theta(s,a)$.

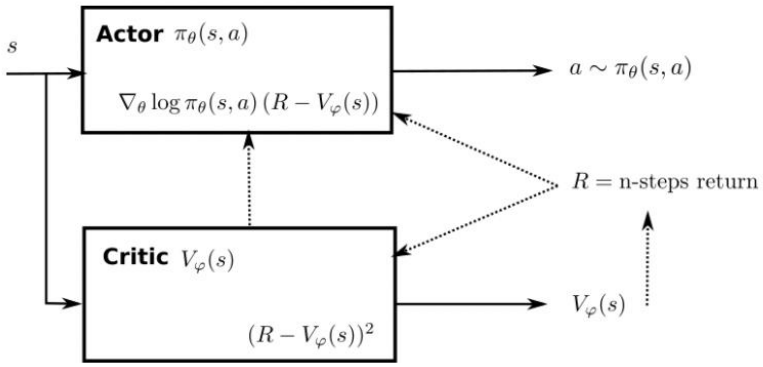


Figure 4.1: Advantage Actor-Critic Architecture [22]

The agent takes action based on π_θ and update the parameter of policy network θ based on gradient descent of entropy loss:

$$\nabla_\theta J(\theta) = \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) \left(R_t - V_\phi(s_t)\right), \tag{4.9}$$

and update the value function network ϕ based on the gradient of squared loss:

$$\nabla_\phi L(\phi) = \sum_t \nabla_\phi V_\phi(s_t) \left(R_t - V_\phi(s_t)\right), \tag{4.10}$$

where the $R_t = \sum_{k=1}^{n} \gamma^{k-1} r_{t+k} + \gamma^n V_\phi(s_{t+n})$, which is the n-step lookahead value function. Besides, we can easily convert this V-type critic into Q-type.

4.5 Soft Actor-Critic (SAC)

The soft actor-critic algorithm is a stable off-policy adaption of the actor-critic algorithm proposed by Haarnoja T et al. [47]. Different from A2C, it aims to maximize

the discounted reward and policy entropy at the same time. Formally speaking, it deploys three sets of networks (Q_w, V_ϕ, π_θ) and incorporates the log value of policy into the loss function. In the updating part, it applies gradient descent on all three parameters:

$$J_V(\phi) = E_{s_t \sim D}\left[V_\phi(s_t) - E_{a_t \sim \pi_\theta}[Q_w(s_t, a_t) - \log \pi_\theta(a_t|s_t)]\right] \quad (4.11)$$

$$J_Q(w) = E_{(s_t,a_t)\sim D}\left[\left(Q_w(s_t, a_t) - r(s_t, a_t) - \gamma E_{s_{t+1}}\left[V_\phi(s_{t+1})\right]\right)\right] \quad (4.12)$$

$$J_\pi(\theta) = E_{s_t \sim D}[D_{KL}(\pi_\theta(\cdot \mid s_t)| \exp \frac{Q_w(s_t,\cdot)}{Z_w(s_t)})]. \quad (4.13)$$

4.6 Deep Deterministic Policy Gradient (DDPG)

Silver D. et al. [48] propose the Deep Deterministic Policy Gradient (DDPG), which is especially efficient in dealing with continuous control problems. Considering that we can always buy the integer number of shares closest to the real number from continuous problems, DDPG is still applicable in stock trading. The critical feature of DDPG is the deterministic policy gradient by updating parameter θ through:

$$\theta^{k+1} = \theta^k + \alpha E_{s\sim\pi_\theta}\left[\nabla_\theta \pi_\theta(s) \nabla_\pi Q^{\pi_\theta}\left(s, \pi_\theta(s)\right)|_{a=\pi_\theta(s)}\right] \quad (4.14)$$

For real practice, it is still computationally efficient to employ an approximation function $Q_w \approx Q^{\pi_\theta}(s, a)$, and similar to SAC, we can iteratively update θ and w through:

$$w_{t+1} = w_t + \alpha_w \left(r_t + \gamma Q_{w_t}\left(s_{t+1}, \pi_{\theta_t}(s_{t+1})\right) - Q_{w_t}(s_t, a_t)\right) \nabla_w Q_{w_t}(s_t, a_t) \quad (4.15)$$

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \pi_\theta(s) \nabla_a Q^w(s_t, a_t)|_{a=\pi_\theta(s)} \quad (4.16)$$

4.7 Twin Delayed Deep Deterministic Policy Gradients (TD3)

TD3 algorithm is designed to overcome the overoptimistic problem in DDPG for specific tasks [53], which can be understood as a pessimistic version of DDPG. TD3 is inspired by Double Q-learning and DDQN [49,50]. The overestimation problem still

exists in Equation (4.15) since the value estimation, and decision making are both based on the same Q^w. This makes the agent too optimistic about the value of the current state and his/her action.

To deal with the overestimation issue, Fujimoto S. et al. [53] propose the Twin Delayed DDPG method by incorporating two actors and critics and updating their parameters iteratively. Technically, the key feature of TD3 lies in using four networks $\left(Q_{w_1}, Q_{w_2}, \pi_{\theta_1}, \pi_{\theta_2}\right)$, which satisfies the relationship of $i, j = 1,2$ separately:

$$w_{t+1}^i = w_t^i + \alpha_w \left(r_t + \gamma Q_{w_t}^j \left(s_{t+1}, \pi_{\theta_t}^i(s_{t+1})\right) - Q_{w_t}^i(s_t, a_t)\right) \nabla_{\mathrm{w}}^{\mathrm{i}} Q_{w_t}^i(s_t, a_t) \quad (4.17)$$

$$\theta_{t+1}^i = \theta_t^i + \alpha_\theta \nabla_\theta^{\mathrm{i}} \pi_\theta^i(s) \nabla_a Q_w^i(s_t, a_t)|_{a \sim \pi^i{}_\theta(s) + \epsilon} \quad (4.18)$$

4.8 Proximal Policy Optimization (PPO)

PPO is a typical policy gradient algorithm designed for RL by Shulman J. et al. [25]. It seeks more stable convergence by assuring the updated policy will not differ too much from the previous policy. A similar algorithm is Trust Region Policy Optimization (TRPO) [54], which is also proposed by Shulman. These two algorithms share the same motivation by adding penalty terms on drastic policy changes.

Technically speaking, the PPO and TRPO solve the policy gradient of:

$$\max_\theta E_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t\right], \qquad s.t. E\left[KL\left(\pi_{\theta_{old}}, \pi_\theta\right)\right] \leq \delta. \quad (4.17)$$

What is special about PPO is the fact that it incorporates a 'clip' form penalty loss in the minimization. This turns the optimization problem (4.17) into:

$$\max_\theta E_t \left[\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t, clip\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) A_t\right], \quad (4.18)$$

which simplifies the optimization stage for calculations.

Chapter Five Empirical Estimation in the A-share Market

5.1 General Introduction of Trading SSE.50, CSI. 300 and CSI.500

SSE.50 Index is one of the main capitalization-weighted stock indices of the Shanghai Stock Exchange. It subsumes the top 50 companies by "float-adjusted" capitalization. Apart from this 50-company index, SSE 180 and SSE 380 also incorporate the SSE 50 Index. Meanwhile, SSE 50 and SSE 180 are sub-indices of the SSE Composite Index. Some representative stocks are shown in Table 1.

Table 5.1 Some Components of SSE 50 Index

Name	Industry	Ticker
Shanghai Pudong Development Bank	Banking	600000
China Petroleum & Chemical Corporation	Oil & Gas	600028
CITIC Securities	Financials	600030
Foxconn	Manufacture & Internet	6001138
Poly Real Estate	Real estate	600048
WuXi AppTec	Medicine	603259

CSI.300 Index and ***CSI.500 Index*** are the main stock market indices. They are designed to demonstrate the performance of the top 300 and top 300-800 (market-value based) stocks traded on the Shanghai Stock Exchange and the Shenzhen Stock Exchange. CSI 300 is usually deemed as the Chinese version of the S&P 500 Index. Some of the CSI 500 components are listed in Table 2.

Table 5.2 Some Components of CSI 300 Index

Name	Industry	Ticker
Ping An Insurance	Financials	601318
BOE Technology Group	IT	000725
Midea Group	Consumer Discretionary	000333
China Vanke	Financials	000002
Kweichow Moutai	Consumer Staples	600519
Industrial Bank	Financials	601166

5.2 Basic MDP Settings for Trading

5.2.1 Trading Environment

In this part, we discuss how to design an exact RL training and validation environment for a given number of stocks (here, we say components of an index), which is a realization of the setting discussed in Chapter Three.

Taking the CSI. 300 as the example, for ***state space***, we have that:

[1] Since there are 300 stocks available for trading, we have $p_t \in R_+^{300}, h_t \in Z_+^{300}$, and the balance value is a positive real value $b_t \in R_+$.

[2] For auxiliary information I_t, we incorporate the following technical indicators for tradable stocks: $MACD \in R_+^{300}, DX(30) \in R_+^{300}, RSI(30) \in R_+^{300}$ $SMA(30.close) \in R_+^{300}, BOLL \in R_+^{300}$; besides, we also incorporate daily $SHIBOR \in R^1$, $Fund\ index \in R^1$ (SSE Fund Index: SSE. 000011), $Bond\ index \in R^1$(SSE Government Bond Index: SSE.000012) and $VIX \in R^1$(CBOE China ETF Volatility Index).

The ***action space*** $a_t \in R^{300}$ represents the long-short decisions on trading day t. Considering the convenience of the training process, we restrict the action space in

$\{-h_{max}, -h_{max}+1, \ldots, 0, 1, \ldots . h_{max}-1, h_{max}\}(h_{max} = 1 \times 10^5)$ for each kind of stock. Since h_{max} is set to be comparatively large, this setting can subsume most trading settings.

The pipeline of our framework is demonstrated in Figure 5.1. The RL agents choose long-short actions for all considered stocks based on their algorithms (when ensembling, they cooperatively provide a decision). The trading environment will calculate the change in net values and send it back to the agents as a reward. Meanwhile, the market will turn to the next day, and new states are passed to agents. In the training stages, the agents will update their policy networks according to the rules mentioned in Chapter Five, while in the trading (validation) stage, we would not update the policy networks. We will describe the details of training and validation in the following section.

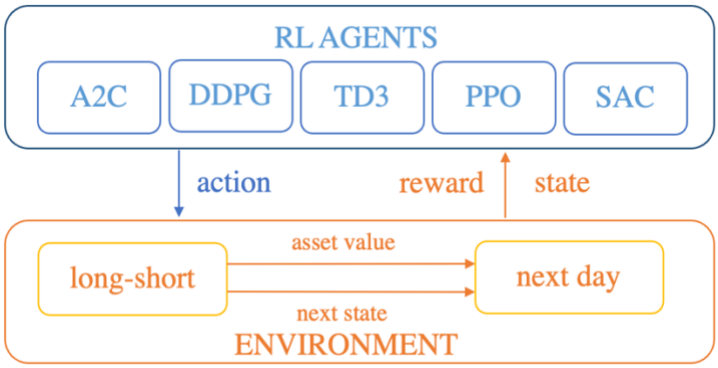


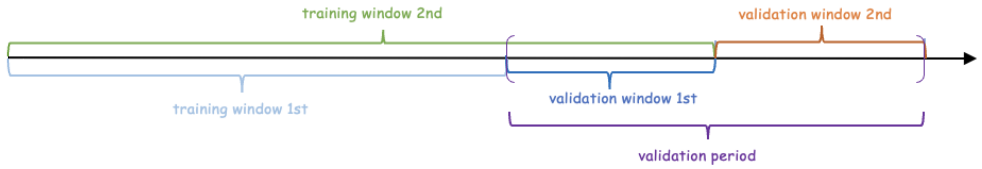
Figure 5.1: Interactive pipeline of RL-trading Framework

5.2.2 Training and Validation Approach

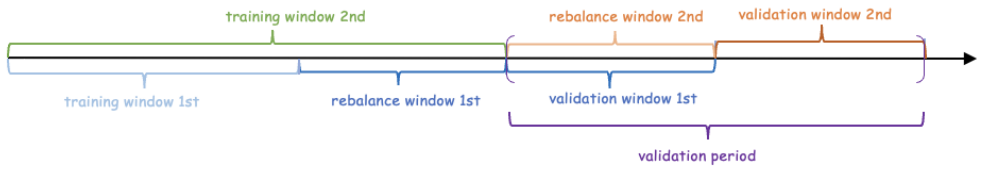
The training and validation procedure in our backtesting setting are conducted according to the fixed-length 'rolling window' scheme in Figure 5.2 (Figure (5.2a) shows trading with only one algorithm, and Figure (5.2b) shows trading based on ensembling methods). The whole validation period is divided into nonoverlapped, consecutive, and same-length validation windows. Before each validation window, RL algorithms needed to be trained in a broader training window, during which the parameters in the algorithms are updated based on gradient descent. Then all parameters

are kept fixed throughout the following validation window. After one 'training-validation' pair, the window moves forward by the length of the validation window and starts a new training round. This rolling-window scheme is quite close to the validation process in the Fama&French three-factor model and has no in-sample estimation errors.

The procedure of training and validating ensembling models differs slightly from that of single-agent ones. To choose or assign weight to different algorithms in an ensembling model, we need an extra 'rebalancing window' to evaluate the algorithms' most recent performance. As shown in Figure 5.2b, there is one rebalance window between validation and training. After training, the algorithms are fixed and evaluated by their Sharpe ratio during the rebalance window. Then the optimal model (Bayesian optimal) or the weights (Bayesian ensembling) are decided for the validation window. After one 'training-rebalance-validation' epoch, the window moves forward by a validation window length. Since we only use historical information during the validation, there is no risk for in-sample or look-forward errors.



(a) rolling window under single agent setting



(b) rolling window under ensembling setting

Figure 5.2: Rolling-window validation workflow

5.2.3 Optimal Ensembling and Model Combination

In this section, we will explain how to implement ensembling based on optimal and weighted strategies. As mentioned in the previous section, when applying the ensembling strategy for trading, we select an action regarding all available algorithms' Sharpe ratio in 'rebalance windows.' Naturally, one can choose the 'best-performed' algorithm, holding the belief that it can still perform well, at least for a short period. This strategy can be understood as choosing the Bayesian optimal model when ensembling, which has been discussed by Yang et al. [63].

Another approach to incorporating historical performance is to assign a confidence level for each agent's suggested action. Since the Sharpe ratio is one of the key measurements for investment, in this paper, we use the Sharpe ratio during the 'rebalance window' as the criteria for assigning weights. Specifically, for algorithm m with a Sharpe ratio of $Sharpe_m^r$, we assign a confidence weight $w_m = \frac{Sharp_m^r}{\sum_i^M Sharpe_i^r}$. In this way, we can decide our final trading action according to all agents' suggestions: $a_{final} = \sum_m w_m a_m$. This approach considering advice from more agents, might be more robust and stable compared with only deploying one agent.

5.3 Details of Backtesting

All experiments are conducted on a server with two GeForce RTX 3090 GPUs and 256 GB RAM. RL algorithms and trading environments are written with Python; neural networks for all RL agents are written with Pytorch; visualizations are based on packages Pyfolio and Empyrical.

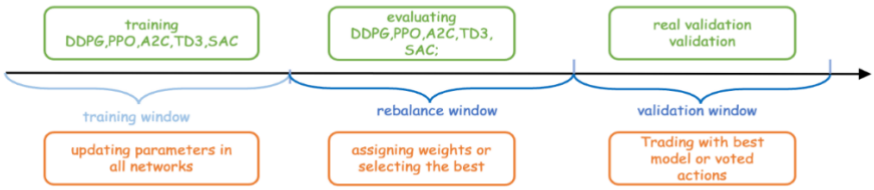


Figure 5.3: Training and validation details

We obtain all stock market data from the CSMAR database, collecting daily stock close, open, high, and low prices with trading volume for all components in SSE.50, CSI.300, and CSI.500 indices from January 1st, 2015, to January 28th, 2022. The first training epoch begins on January 1st, 2015, and ends on January 1st, 2019, and then we begin the first epoch of rebalancing and validating. Then we move the validation window forward by one window length. For training simplicity, we set the rebalancing window and validation window to be the same all the time, and there are three different sizes (21 days, 42 days, and 63 days) of window lengths being tested.

We deploy five RL algorithms: DDPG, A2C, TD3, PPO, and SAC for single-agent trading and ensembling. The value-function and policy networks are multilayer perceptron (MLP). The γ of every algorithm is its corresponding daily spot rate. In the training stage, considering the different convergence properties, we assign algorithm m with $step_m$ as training epoch. During the first period of training, with the aim of collecting more possible paths, agent m will be updated by $8 \times to\_step_m$ times. For the latter training epoch, the parameters will be updated for $\frac{3 \times round}{total\ round} \times to\_step_m$ times. The $\frac{3 \times round}{total\ round}$ grows larger when the validation window forwards to help our agents learn and update parameters more when training periods move on, which is essential to capture more recent market information and patterns.

For macro-level information I_t, we include daily SHIBOR and half-year spot rate to capture daily interest information; CNY-USD to represent exchange information; and

Shanghai Securities Composite Index (SSCI) to capture market-level fluctuation. In order to avoid look-forward traps, all information used for trading lags for one day.

In addition, some critical parameters for each agent during the training process are listed below:

A2C parameters: *updating steps=5 (the number of steps to run before each update); total step (to_step_{a2c})=80000; learning rate=0.0005; learning decay=0.99.*

DDPG parameters: *τ=0.0015 (soft update rate); actor learning rate=0.0001 and critic learning rate=0.001; to_step_{DDPG}=60000.*

TD3 parameters: *learning rate=0.0003; τ=0.005 (soft update rate); gradient steps=100 (the number of parametric updates between two steps); total step (to_step_{td3})=100000.*

SAC parameters: *updating steps=3 (the number of steps to run before each update); total step (to_step_{sac})=120000; learning rate=0.0003; learning decay=0.99; τ=0.005 (soft update rate)*

PPO parameters: *updating steps=3 (the number of steps to run before each update); total step (to_step_{ppo})=150000; learning rate=0.0003; learning decay=0.99; ϵ_{clip} =* 0.2.

5.4 Performance Evaluation

5.4.1 Performance measurement

To examine the effectiveness of the investment results of our trading strategy, we use a large number of performance metrics. Apart from the widely used annualized rate of return and Sharpe ratio, we would like to introduce some other well-known indicators in this section.

Max drawdown evaluates the portfolio's downside risk, measuring the maximum

values from peaks to troughs. We can calculate it by $MD = \frac{(V_t - V_p)}{V_p}$, which is always nonpositive in value.

Treynor ratio is a measure of risk-adjusted portfolio performance, which is constructed based on systematic risks, following $Treynor_p = \frac{r_p - r_f}{\beta_p}$. The β_p value we use here represents our strategy's linear reaction towards the market portfolio, which we take as the corresponding index return. A portfolio or trading strategy with a high Treynor ratio indicates more returns from a unit of risk.

Sortino ratio is a variant of the Sharpe ratio by only considering downside risks. We can calculate it from $Sortino_p = \frac{r_p - r_f}{\sigma_d}$, where the $\sigma_d^2 = \int_{-\infty}^{r_f} (r_f - r_p)^2 f(r_p) dr_p$. Intuitively, we can also rewrite it as $Sortino_p = \frac{r_p - r_f}{\sqrt{E[max(r_p - r_f, 0)^2]}}$. Different from Sharpe, Sortino only considers the risks in the periods when the risk-free financial instruments outperform the portfolio.

Calmar ratio measures the risk-adjusted return by max drawdown ratio and can be regarded as replacing the standard deviation risk in Sharpe ratios with downside risk. Usually, we will calculate it through the equation: $Calmar_p = \frac{r_p - r_f}{|MD_p|}$. Fund managers always desire a larger Calmar ratio since more returns are obtained with unit risk.

Omega ratio is initially defined as $\Omega(r_f) = \frac{\int_{r_f}^{\infty} \bar{F}(r) dr}{\int_{-\infty}^{r_f} F(r) dr}$, which is the times' probability of gaining more than r_f over that of less than r_f. With a simple calculation, we may find that $\Omega(r_f) = 1 + \frac{r_p - r_f}{E[\max(r - r_f, 0)]}$. A larger Omega ratio indicates more chances of getting positive profits and thus better investment performance.

Tail ratio estimates the stability of portfolios and investment strategies. For stock-based funds, we usually use the first top 95 percent and the last 5 percent to construct the Tail ratio: $Tail = \frac{r_{0.95}}{r_{0.05}}$. A larger tail ratio indicates more chances of getting a

positive return in extreme cases.

5.4.2 Backtesting Evaluation on SEE.50 components

This section will demonstrate some of the backtesting results of trading components in the SEE.50 index. The training stage starts on January 1st, 2015, and the first validation period starts on January 1st, 2020. We stop the validation period until February 28th, 2022. For the validation of ensembling algorithms, we try different lengths of rebalancing windows as 21 days (one trading month), 42 days (two trading months), and 63 days (three trading months). Besides, in order to avoid unexpected systematic market risks, our agents will clear all positions when the market volatility exceeds the 90-percent quantile of the historical turbulence in training periods. The initial cash for backtesting is 1×10^7.

Figure 5.3 below illustrates the performance of all single agents, the optimal ensembling agent, and the weighted ensemble agent for a 21-day window length. We use the return of passively investing SEE.50 as the baseline to better illustrate each agent's performance. We can see that all agents achieve annualized returns over 30% and Sharpe ratios over 1.2, while the baseline only maintains an annualized return of 7.25% and a Sharpe ratio of 0.44. However, the risk control of RL agents is generally poor, which we will discuss in Chapter Six.

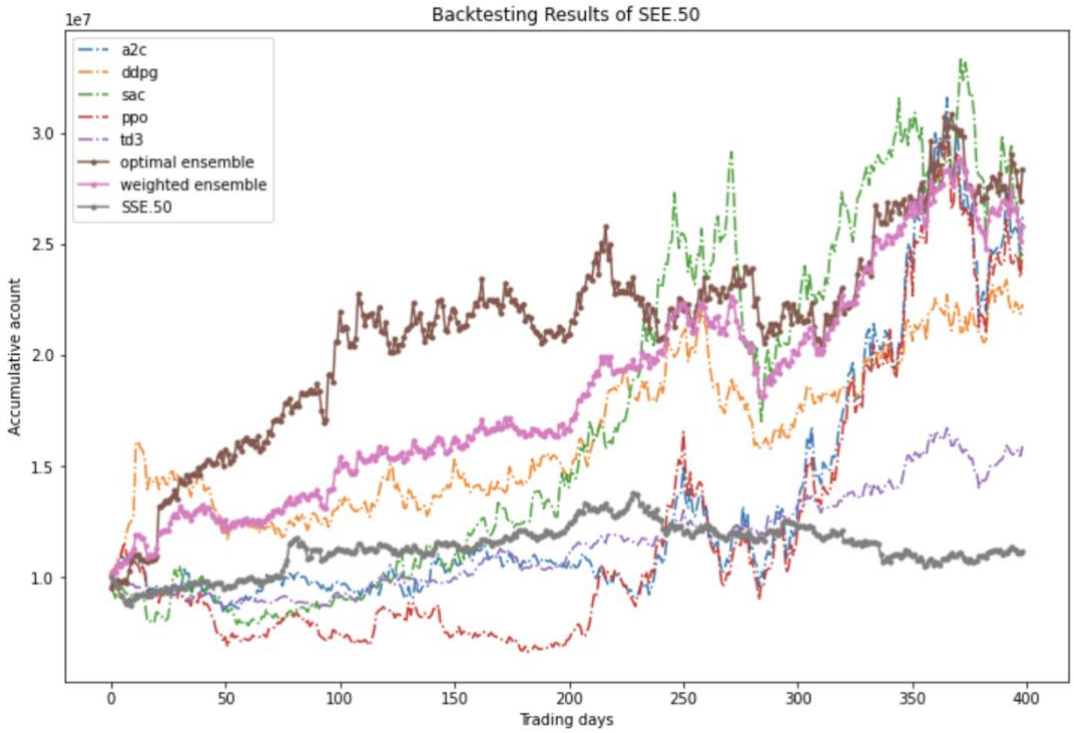


Figure 5.4: Accumulative return of RL-SSE.50 strategy

Table 5.1 below summarizes the investment performance of the **optimal ensembling strategy**. From the table, we see that from all performance measures, SAC achieves the most prominent return, reaching over 100% annualized return, and this return secures a comparatively high Sharpe ratio of 1.84, regardless of the fact that the MMD is -44%. Besides, all other agents generally achieve better results than the baseline if only profitability is considered. Among them, two ensemble agents achieve the most plausible results, balancing both return and risk.

Table 5.2 Performance summaries of different agents in SEE. 50

Agent	Acc. return	Ann. return	Sharpe	Calmar	Sortino	Stability	Max drawdown	Omega	Tail
A2C	162.23%	83.89%	1.30	2.32	1.98	0.67	-41.94%	1.24	1.12
TD3	59.08%	34.07%	1.42	2.11	2.20	0.92	-16.08%	1.27	1.28
DDPG	123.03%	65.96%	1.34	2.33	2.25	0.79	-28.34%	1.26	1.11
PPO	150.26%	78.35%	1.22	1.73	1.88	0.63	-45.42%	1.23	1.26
SAC	161.83%	89.40%	1.40	2.03	2.10	0.91	-41.13%	1.27	1.04
Optimal ensemble	167.47%	86.15%	1.79	4.42	3.14	0.70	-21.75%	1.38	1.47
Weighted ensemble	163.01%	84.18%	2.04	4.04	4.08	0.95	-19.21%	1.52	1.54
SEE. 50	11.73%	7.25%	0.44	0.30	0.64	0.28	-23.72%	1.01	1.01

From Eastmoney (http://fund.eastmoney.com/trade/zs.html), we can see that stock-based funds' top 5% quantile investment performance (annualized return) is only 21.3%. From Table 5.1, we can see that our ensembling strategy's annualized performance (return and Sharpe) can rank at least around the three.

The following Table 5.2 documents the ensembling details of the optimal ensembling model (rechoose models every month). We present the exact model used by the optimal ensembling agents in this table during specific validation periods. It can be observed that all models are employed during the backtesting, and with the advantages of choosing the 'best' among all agents, the ensembling algorithm only has a -15% MMD, which is comparable to the baseline MMD, while keeping a 10-time annualized return.

Table 5.3 Some ensembling details of the optimal ensemble model

start date	end date	used model	start date	end date	used model
2019-01-01	2019-02-01	SAC	2021-07-04	2021-08-04	A2C
2020-02-02	2021-03-03	TD3	2021-12-04	2022-01-03	PPO
2021-03-03	2021-04-03	TD3	2022-01-03	2022-01-30	DDPG

Figure 5.5 to Figure 5.8 separately demonstrate the accumulative return of our trader during the whole validation period, 6-month volatility, 6-month-based annualized Sharpe ratio, top five drawdowns, beta (based on the index), and the distribution of returns. We can see that, apart from some small drawdowns, the ensembling agent can obtain continuous profits. Besides, considering risk control, the ensembling agent has a stable six-month beta and volatility across the backtesting periods.

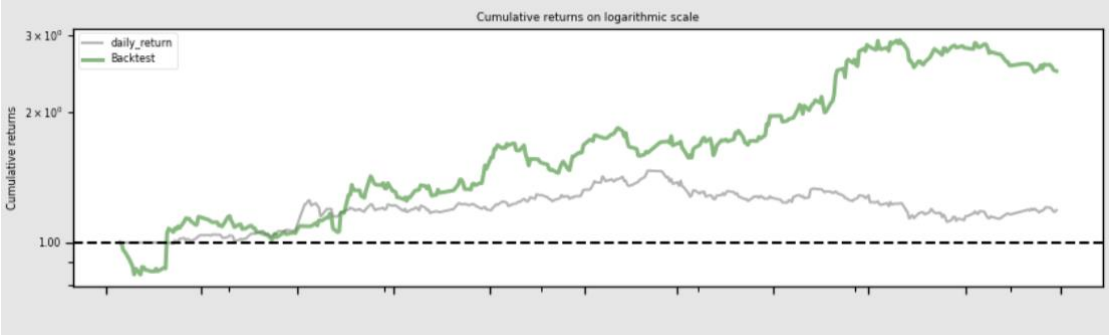


Figure 5.4: Accumulative return of the optimal-ensembling RL-SSE.50 strategy

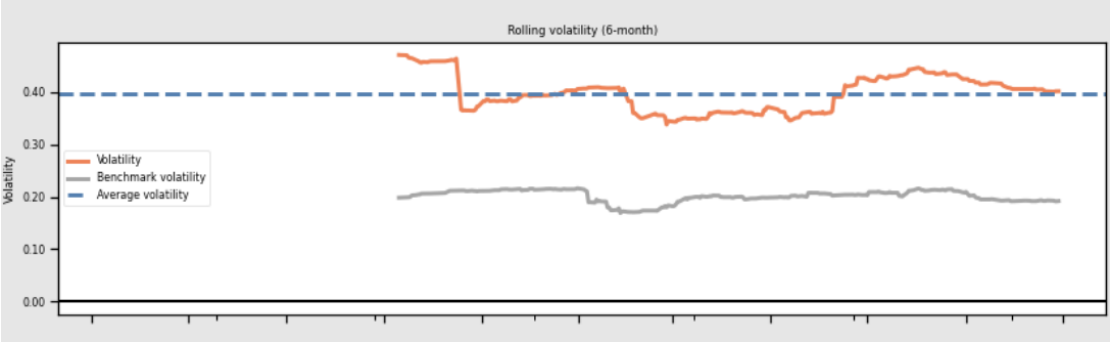


Figure 5.5: Six-month volatility of the optimal-ensembling RL-SSE.50 strategy

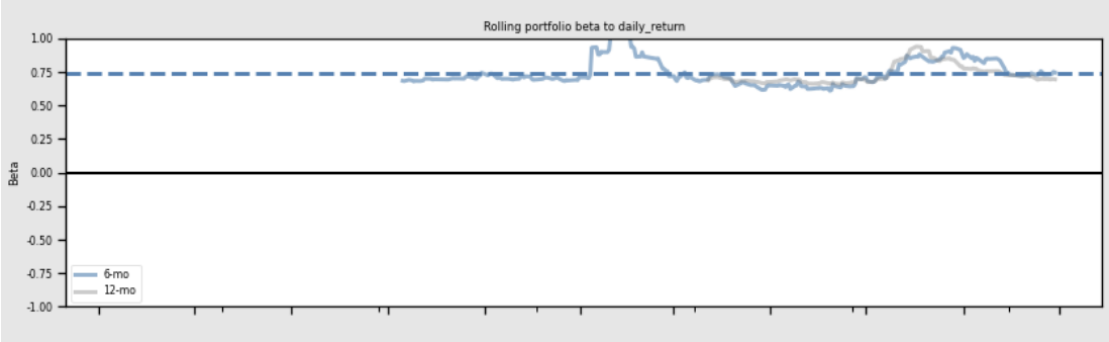


Figure 5.6: Beta of the optimal-ensembling RL-SSE.50 strategy

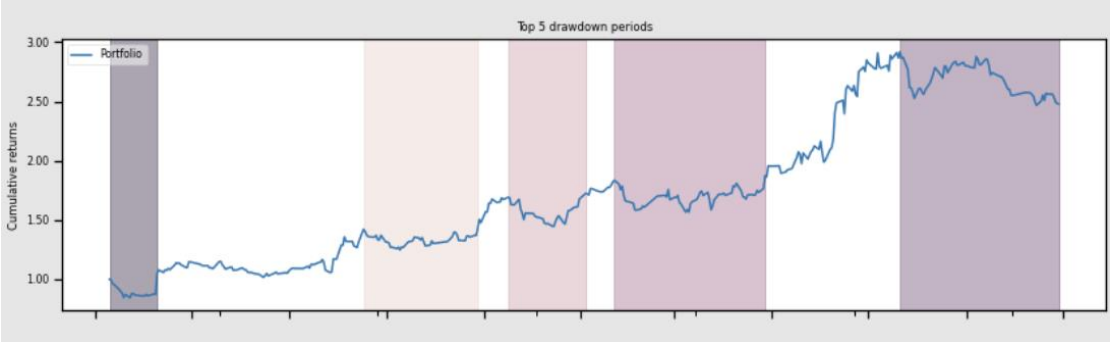


Figure 5.7: Five largest MMDs of the optimal-ensembling RL-SSE.50 strategy

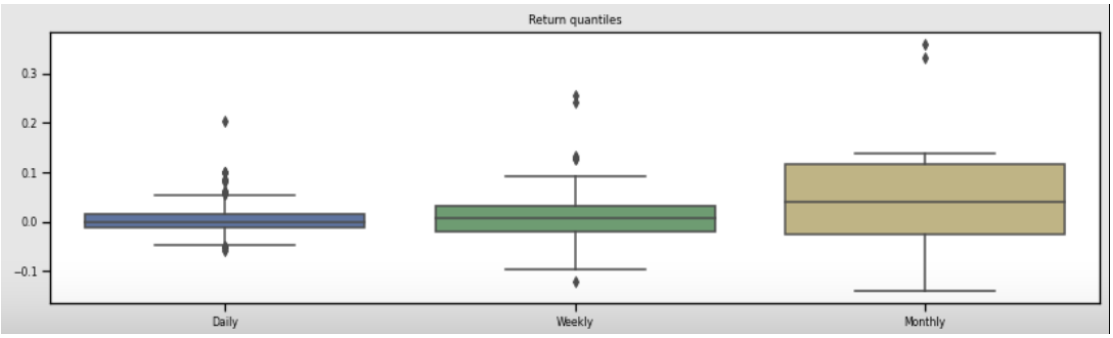


Figure 5.8: Return quantiles of the optimal-ensembling RL-SSE.50

Generally, we can see that the optimal ensembling strategy in SEE.50 may introduce a bit more risk while bringing much more return compared with baseline, which therefore leads to an over 1.8 Sharpe ratio. The performance of weighted ensemble is also remarkable, and we will detailly analyze it when discussing CSI.300 back testing.

5.4.3 Backtesting Evaluation on CSI.300

This section will demonstrate some of the backtesting results of trading components in the CSI.300 index. Since its long-term training pattern is quite similar to SEE.50, we would like to look more detailly into the short-run performance pattern of our ensembling agents.

We demonstrate the result of backtesting, whose training stage starts on January 1st, 2015, and the first validation period starts on October 1st, 2020. We end the validation

period until July 31st, 2021, which indicates a 9-month validation. For the validation of ensembling algorithms, we test the rebalancing window length as 63 days. Besides, in order to avoid unexpected systematic market risks, our agents will clear all positions when the market volatility exceeds the 90-percent quantile of the historical turbulence in training periods.

Table 5.2 summarizes all agents' performance during this period. We can see that among all agents, TD3 performs the best considering return metrics, while optimal and weighted ensembling achieve desirable returns and volatility at the same time. To better compare, we add the return of passively investing CSI.300 as the baseline to better illustrate each agent's performance. We can see that all agents reach an annualized return over 23% and a Sharpe ratio over 1.3, while the baseline only holds an annualized return of 16.29% and a Sharpe ratio of 0.88. This demonstrates the superiority of utilizing RL in trading frameworks.

Table 5.4 Performance summaries of different agents in CSI. 300

Agent	Acc. return	Ann. return	Sharpe	Calmar	Sortino	Treynor	Max drawdown	Omega	Tail
A2C	15.39%	21.03%	0.90	0.95	1.21	0.41	-17.54%	1.12	1.31
TD3	68.80%	101.12%	1.73	2.54	2.26	2.13	-51.62%	1.54	1.51
DDPG	25.02%	34.69%	1.46	1.41	1.89	0.34	-37.21%	1.04	1.01
PPO	34.13%	47..93%	1.53	1.69	2.23	0.73	-35.77%	1.42	1.13
SAC	23.99%	33.12%	0.97	0.80	1.27	0.63	-27.14%	1.31	1.07
Optimal ensemble	63.74%	93.21%	1.81	3.78	2.23	0.71	-24.64%	1.53	1.31
Weighted ensemble	85.,49%	127.91%	2.07	6.27	3.79	0.86	-20.39%	1.45	1.66
CSI. 300	11.98%	16.29%	0.88	1.07	1.29	0.14	-15.18%	1.16	1.00

Considering two ensemble agents that achieve plausible results with accumulative returns of 63.74% and 85.49%, the Sharpe of 1.81 and 2.07, Calmar ratio of 3.,78 and 6.27 and Sortino ratio of 2.23 and 3.79. From Eastmoney, we can see the average investment performance of stock-based funds seldom achieve more than 30% for a nine-month return. Thus, we can see that our strategy's annualized performance (return and Sharpe) can rank at least within the top three for both ensembling methodologies.

Details of ensembling weights in the **weighted ensembling strategy** are documented in Table 5.3. From the table, we can see that the weights of different agents vary based on their performance in the corresponding validation window. We may see that since TD3 always stands out from all agents, its weights are always the largest.

Table 5.5 Ensembling details of weighted ensemble model

start date	end date	TD3	SAC	A2C	PPO	DDPG
2020-09-01	2020-01-05	1.04	-1.16	-0.22	1.25	0.093
2020-01-05	2021-04-12	0.81	-0.03	0.15	-0.14	0.21
2021-04-12	2021-07-30	0.49	0.08	0.02	-0.07	0.46

Figures 5.9 to 5.15 separately demonstrate the accumulative return of our weighted-ensembling trader during the whole validation period, 6-month volatility, 6-month-based beta, 6-month-based annualized Sharpe ratio, top five drawdowns, beta (based on the index), and the distribution of returns.

Our strategy may introduce a bit more risk while bringing much more return, which therefore leads to an over 2 Sharpe ratio. We can see that; our strategy significantly outperforms the passive baseline. Moreover, what is noticeable is that the beta of this trading strategy is only 0.25, and the alpha is 1.34. This demonstrates its great robustness against systematic risks and gains abnormal returns.

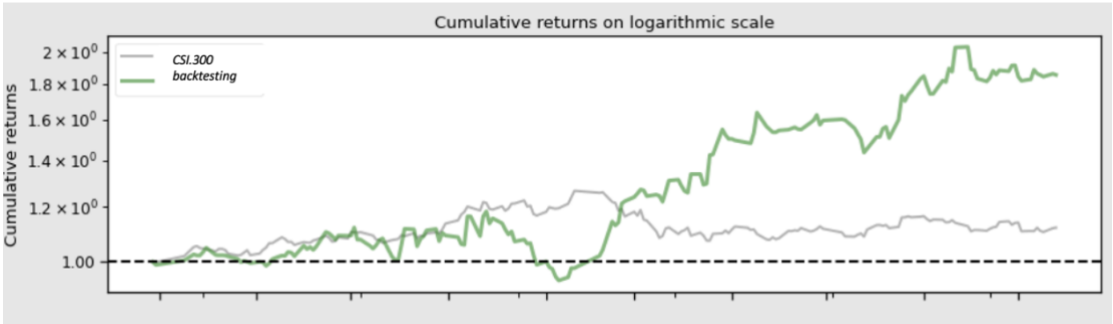


Figure 5.9: Accumulative return of weighted-ensembling RL-CSI.300 strategy

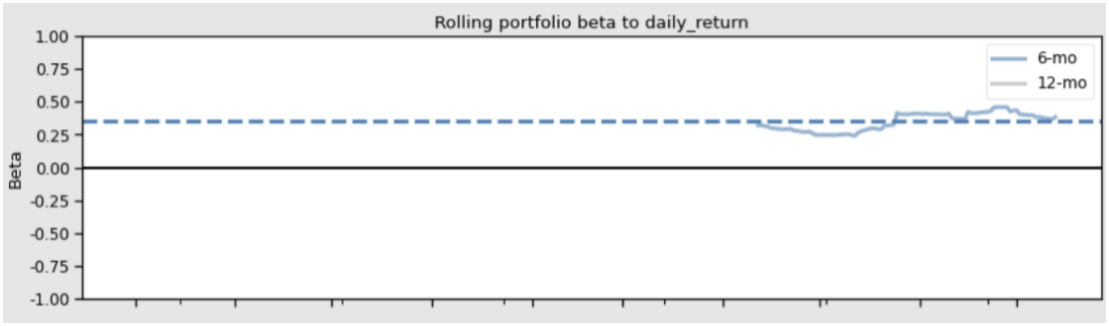


Figure 5.10: Beta of weighted-ensembling RL-CSI.300 strategy

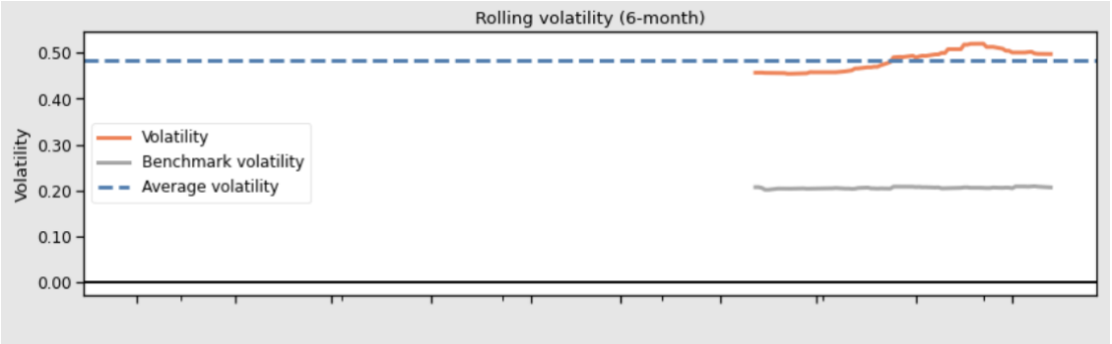


Figure 5.11: Volatility of weighted-ensembling RL-CSI.300 strategy

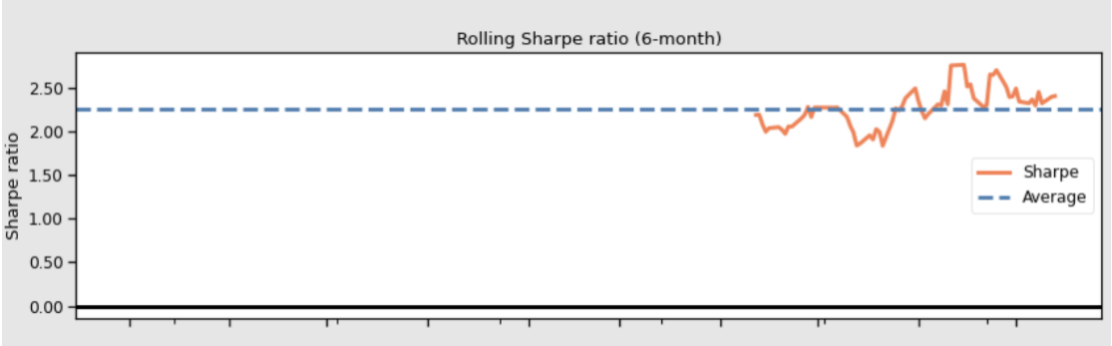


Figure 5.12: Sharp ratio of weighted-ensembling RL-CSI.300 strategy

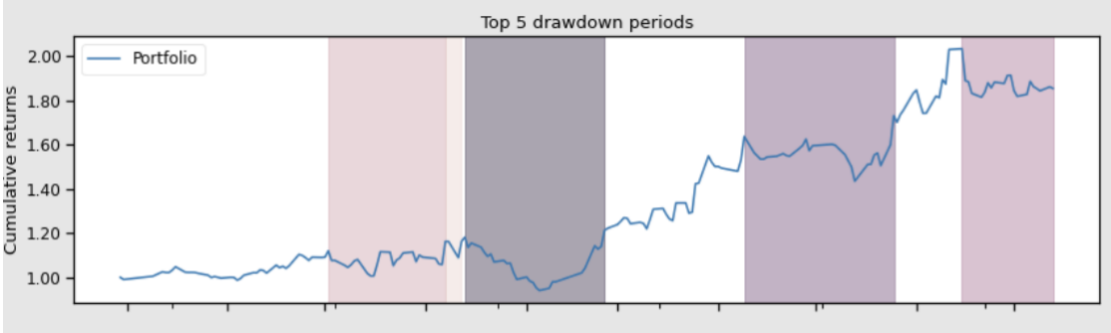


Figure 5.13: Five main drawdowns of weighted-ensembling RL-CSI.300 strategy

Figure 5.14: Summary of returns of weighted-ensembling RL-CSI.300 strategy

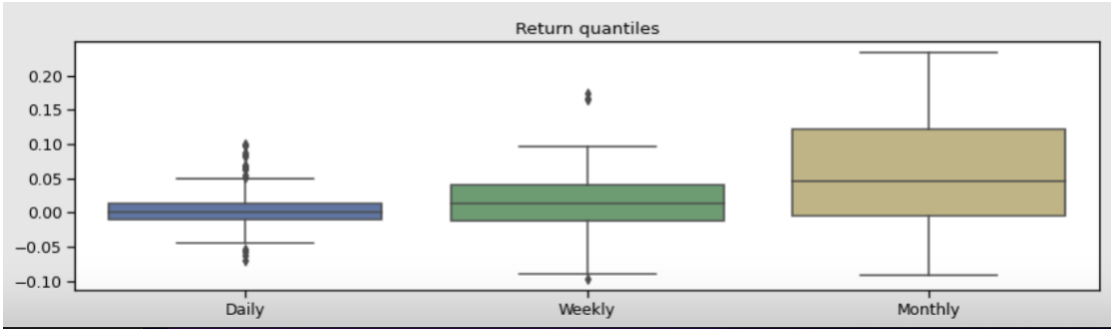


Figure 5.15: Return quantiles of weighted-ensembling RL-CSI.300 strategy

5.4.4 Backtesting Evaluation on CSI.500

This section demonstrates some backtesting results of trading components in the SEE.50 index. The training stage starts on January 1st, 2015, and the first validation

period starts on January 1st, 2020. We end the validation period by February 28th, 2022. For the validation of ensembling algorithms, we try different lengths of rebalancing windows, including 21 days (one trading month), 42 days (two trading months), and 63 days (three trading months). Besides, in order to avoid unexpected systematic market risks, our agents will clear all positions when the market volatility exceeds the 90-percent quantile of the historical turbulence in training periods. For convergence concern, we double the training epoch considering the high dimensionality.

The following Table 5.3 summarizes the investment performance of all single-algorithm traders, the optimal ensembling strategy and the weighted ensembling strategy, with a 63-day window length. For better comparison, we add the return of passively investing CSI.500 as the baseline to better illustrate each agent's performance.

We can see that all agents reach an annualized return of over 20%, which is double of benchmark return. They also achieve Sharpe ratios over 0.9, while the baseline only holds an annualized return of 10.95% and a Sharpe ratio of 0.6. Among all single-agent traders, TD3 performs best in return measures, beating the baseline by five times. However, a significant problem related to this is the poor risk control, reaching a max drawdown over 40%, which indicates consecutively losing 40% net value during specific periods.

Fortunately, our ensembling methods keep a good balance between returns and risks. From the table, we can also see that from all performance measures, both ensemble agents achieve good results with an annualized return of 40.61% and 40.24%, the Sharpe of 1.73 and 1.77, Calmar ratio of 1.87 and 2.33, and a max drawdown of only 17.2% and 15.2%. From the famous fund evaluation website Eastmoney (http://fund.eastmoney.com/data/fundranking), we can see the average investment performance of stock-based funds seldom exceeds 10%, and we can see that our strategy's annualized performance (return and Sharpe) can rank at least around the top five.

Table 5.6 Performance summaries of different agents in CSI. 500

Agent	Acc. return	Ann. return	Sharpe	Calmar	Sortino	Treynor	Max drawdown	Omega	Tail
A2C	63.39%	27.93%	0.91	0.83	1.44	0.37	-33.73%	1.16	1.22
TD3	99.15%	41.12%	1.57	1.02	2.97	1.73	-40.34%	1.43	1.77
DDPG	74.06%	31.69%	1.13	1.28	1.73	0.90	-24.68%	1.21	1.03
PPO	88.26%	37.21%	1.53	1.36	2.23	1.15	-27.45%	1.25	1.19
SAC	49.94%	22.45%	0.90	0.61	1.43	0.42	-37.00%	1.17	1.14
Optimal ensemble	97.71%	40.61%	1.73	1.87	2.16	0.39	-21.64%	1.62	1.47
Weighted ensemble	96.67%	40.24%	1.77	2.33	2.47	0.44	-17.21%	1.67	1.54
CSI. 500	23.10%	10.95%	0.60	0.72	0.80	0.77	-15.24%	1.11	0.92

We would now take the example of trading CSI.500 components to analyze the performance differences between weighted detailly. In all settings, the optimal ensembling agent outperforms the weighted ensembling one. This conclusion is relatively straightforward since the weighted ensembling method dilutes the power of choosing the optimal agent's action. However, this voting scheme also brings benefits considering risk measurement since it avoids putting all eggs in the same basket, even though this basket is the so-called best one. This situation is especially true when there is an unexpected market shock, in which recent performance no longer holds for the next period. By considering action advice from all agents, weighted ensembling sacrifices some returns for better stability, which can be observed from the higher Sharpe, Calmar, and Sortino ratio and lower max drawdown.

5.4.5 Summary of RL agent's performance

From the three sets of experiments conducted above, we now summarize five main properties of the reinforcement trader: 1) profitability, 2) risks, 3) optimism and pessimism, 4) long-run and short-run performance 5) data dimensionality.

For profitability, RL agents reveal strong power in capturing the rising trend of the financial market and gaining more from proactive actions. From repetitive experiments, RL agents, on average, earn 68.7% more from the rising market trend. Besides, most RL agents achieve annualized returns over 50%, and the return rates are even more

prominent (over 70%) considering the short-run (backtesting for only one year). From the perspective of the Sharpe ratio, most agents still obtain desirable results (in SEE. 50 backtesting, DDPG with 1,34, and TD3 with 1.43)

For risks, single-RL agents tend to be more aggressive investors with comparatively poor risk control, while ensembling methods incorporate a more risk-aversion attitude. Taking the standard deviation (std), for instance, under CSI.300 setting, the annualized standard deviation of TD3 is nearly 73%, SAC is around 69%, OPP is over 76%, weighted ensembling is 44.80%, and optimal ensembling is 50.04%. For comparison, the baseline index volatility is only 19.22% during this period. Besides, the max drawdown ratio also demonstrates the same results. Moreover, the single-agent algorithm with high returns tends to possess more risks, while the ensembling scheme can somehow improve the return-risk tradeoff. All of these demonstrate the risk reduction potential possessed by ensembling methods.

The belief (optimism and pessimism) of the market also influences investment performance. The differences between optimistic and pessimistic RL algorithms are always discussed for computational convergence in the field of computer science and artificial intelligence. The reinforcement learning community usually endorses pessimistic algorithms since optimistic ones always cannot find the global optimal regarding over-optimism. However, in a trading setting, optimism works better in blooming periods. For example, considering Figure 5.4 (backtesting in SEE.50), the DDPG outperforms TD3 (DDPG's pessimistic and dual-actor version) by 47% from 100d to 200d when the market experienced prosperity. This phenomenon also widely exists in other backtesting scenarios.

To compare the long-run and short-run performance, we truncate the backtesting (CSI.300) of the optimal ensembling model on January 1st, 2021. We summarize annualized return, Sharpe ratio, and MDD in Table 5.4. Besides, we append the nine-month testing results mentioned for comparison. We can see that RL agents perform

much better in short-period backtesting, with one-year backtesting possessing an average of 29% more annualized return than two-year testing. The difference between nine-month testing and the other two is even more drastic, with the nine-month annualized return having twice of one-year results. Besides, the Sharpe ratio and max drawdown (risk control) in short-term evaluations.

This 'short period beats long period' possibly only generally comes from lack of training. Since we spend most of our training epochs in the first stage, when the validation window moves forwards, the pattern captured may no longer be useful. Though our training scheme has already taken this problem into consideration, long-run success still cannot be guaranteed. This indicates that retraining is still required for long-run evaluation.

Table 5.7 Comparison between long-run and short-run performance

	Ann. return	Sharpe ratio	MMD
Nine-month backtesting	127.91%	2.07	-20.39%
One-year backtesting	79.24%	1.81	-23.21%
Two-year backtesting	47.17%	1.53	-25.33%

We would also like to compare the computation and scalability problems, which may be more related to computational or practical concerns. As the dimensionality of financial data increases, within the same training epoch, the achieved returns fall, and risks soar. This generally derives from lacking convergence since the larger model contains more parameters. Besides, the training time also lasts longer for a larger model, which is demonstrated in Table 5.5. This reveals the problem of scalability, which we need to solve in further works.

Table 5.8 Training time comparison for different sizes of input

Index	State dimension	Experiment time (s)
SEE. 50	459	2.70h
CSI. 300	3011	17.64h
CSI. 500	4672	23.18h

Chapter Six Conclusions and Discussions

6.1 Summaries and contributions

From the backtesting results, we prove the feasibility of applying state-of-the-art reinforcement learning algorithms in the A-share market and test their performance on portfolio construction of different sizes. All algorithms perform much better than the baseline algorithms after long training epochs. Further, we explore the possibility of ensembling-RL and Bayesian ensembling-RL, which possess better profitability and stability. Besides, we analyze in detail the action patterns of optimistic and pessimistic RL algorithms under different economic scenarios. When the market is booming, we find that optimistic algorithms (A2C and DDPG) perform better. In gloomy periods, pessimistic algorithms (SAC and TD3) may work better, which looks like the investment patterns of human investors.

6.1.1 Feasibility of RL algorithms in A-share market algorithmic trading

In this paper, we extend the use of reinforcement learning into the field of the A-share market and demonstrate its great potential in handling high-dimensional financial inputs. We find that reinforcement learners are remarkably suitable for algorithmic trading and can well capture the system dynamics and give comparatively 'correct' decisions compared with the market index (passive investment). By consecutively interacting with the market and exploring historical paths repeatedly, the agent can generally understand how to avoid loss and maximize long-run profits. Among them, taking trading CSI.500 index for two years as an example, the A2C agent achieves 63.39% returns, TD3 agent achieves 99.15%, DDPG achieves 74.06%, and SAC achieves 49.94%, while the passive baseline only has 23.10%.

However, one main limitation of employing an RL agent is instability and volatility. From experiments, we find that even after training for longer epochs, desirable returns are not secured, though with high probability. The instability may derive from the intrinsic stochasticity learning pattern of RL, since each time, the learning path is totally different, and this will result in different learning results. Another possible reason for this is that the market itself might not satisfy the MDP setting, and deploying a reinforcement learning algorithm may only reach nearly-optimal results rather than converging to the global optimality. Therefore, each time of backtesting will lead us to somewhere closer to the best results instead of a fixed one. Besides, this instability may come from lacking skillful fine-tuning or fewer layers in networks to extract complex value functions.

Return volatility mainly comes from the risk-return tradeoff (single-task trap) and pattern detection failure. Since we only feed one single agent with one day return as a reward and train it with the target to maximize the discounted long-run profits, risk-aversion may be neglected. Therefore, the agent might only fix our target at the profits and consider less about potential risks. Another culprit to volatility (especially the downside one) is pattern detection failure. This indicates that the actual market fluctuates frequently and can be detrimentally influenced by unpredictable information outside of the market (e.g., wars and plagues). An RL agent will not be able to capture these patterns since it can only learn from its experiences. To cope with this problem, an ensembling approach with rebalancing windows will help since it provides us a tool to test which agent can deal with the financial market best and collect wisdom from different knowledge bases.

6.1.2 Two ensembling schemes for RL algorithms in the A-share market

The ensembling method seeks to incorporate the information and knowledge from

different constituents and combine them with the aim of obtaining more robust and excellent performance. It can also overcome the single-task trap encountered when only having one agent for two reasons:

1) Multi-agents with rebalancing windows provide an extra validation chance. Taking optimal ensembling strategy as the example, if we need to choose an agent for the next 63 days' trading, before taking action, we can use the rebalancing window to evaluate each agent's recent 21 days' performance. Holding the belief that market patterns will not change so rapidly, a recently wise strategy has a higher probability of performing well.

2) Ensembling strategy provides more robust investment advice. A simple illustration of robustness is that if deploying n identical agents (with an approximated value function of $f_i(s,a)+\epsilon$) and feed them with independent paths, the averaged value function will have a standard deviation of $\frac{\epsilon}{\sqrt{n}}$. This is also true for our ensembling scheme. Taking the $\frac{Sharp_m^r}{\sum_i^M Sharpe_i^r}$ can avoid 'putting all eggs in the same basket,' Therefore, if the so-called best model in optimal ensembling fails to obtain desirable returns, other agents' advice will make some compensations.

3) Ensembling provides more chances to win under different market trends. Since we incorporate optimistic who always overestimate currently obtained value function and pessimistic agents who always underestimate that, ensembling introduces more possibilities to explore and exploit well under different economic settings.

For all these reasons, we suppose that ensembling methods might be a more stable and robust choice, and the backtesting results also justify our expectations. For instance, in the backtesting of CSI. 500, we may find that the weighted ensembling outperforms optimal ensembling in Sharpe (1.77 vs. 1.73) while performing worse in annualized

return rate a little bit (40.24 vs. 40.61). This indicates lower volatility, therefore avoiding some controllable risks. However, sometimes choosing to trust only one agent might not be a good choice since continuous investment success is not always secured. This can be seen in the backtesting of CSI.300, where weighted ensembling beats the optimal one in both returns and stability.

6.2 Limitations

Though our ensemble RL agent has achieved quite successful results in backtesting, there are still some limitations in our work, such as simplified trading settings, unsatisfying drawdown rate, and long training time. Some future works are needed to deal with these challenges.

6.2.1 Simplified trading environment

In our work, we may oversimplify the stock trading process in the A-share market. We assume that all bids are traded under the close price, and there are sufficient shares for traders to purchase and sell. However, all aforementioned assumptions are not realistic for real-world trading. In the exact trading scenario, we may need to consider more about ask-bid prices and the limited volume of available stocks. We adopt this trading setting since, currently, in China, the market still keeps on the 'T1' trading policy, which indicates that we may only allow making purchases once a day for the same type of stock. However, our research ultimately desires to analyze and develop algorithms for the exact trading procedure.

To make the trading environment closer to the real market, tick-level data needed to be considered. Considering tick level data, however, may raise two significant challenges. The first one is that we need to consider the market timing, which means when to make deals, since, under current regulation, only one trade is allowed every day. Secondly, the new model will incorporate exponentially larger volumes of information since, on every trading day, thousands of trading steps should be included.

These two issues make training an ML/RL-based agent significantly more challenging. The same data challenge also arises when considering the exact available volume of stocks since this adds another dimension for every stock.

6.2.2 Unsatisfying drawdowns

The maximum drawdown (MDD) measures how much the largest observed value loss between two near peaks. During our backtesting, though achieving comparatively high annualized return and Sharpe ratio, the maximum drawdown during all trading days is comparatively large (around 20% to 30%), which still can be improved compared with the average MDD of 20.1%.

A possible solution to lowering the drawdown rate is to incorporate an additional term describing the consecutive drawdown as a reward for the agent. For example, we can rewrite the reward as $r(s_t, a_t, s_{t+1}) = (b_{t+1} + p_{t+1}^{\top} h_{t+1}) - (b_t + p_t^{\top} h_t) - c_t - \zeta DD(p)$, where $\zeta DD(p)$ provides an additional penalty if there are consecutive p days of loss. Besides, a clearing strategy is another approach. This indicates clearing all positions if the net value of the portfolio has consecutively dropped for specific days.

6.2.3 Long training time and lack of convergence

The average training time of our models is listed in Table 5.5. We may find that the whole training time cost is comparatively long, especially for CSI. 500. Besides, we also find that the RL agents' performance is significantly poorer in CSI. 500 and CSI.300 than CSI. 50. A possible explanation for this phenomenon is non-convergence, which is quite usual for ML in high-dimensional data.

Compared with the trading environment for CSI. 50, CSI. 500 setting has around 3000 more features, which definitely leads to a higher computation burden. Therefore, it will be harder for RL agents to explore the hidden transition dynamics since larger input networks and more weights are needed to approximate the value function.

Because of this immense computation burden, within the same training budget, worse investment returns are understandable.

A possible solution to this computational issue is dimension reduction. Technically speaking, we can pre-train an embedding $\mathcal{E}(s)$, projecting raw feature space to an exacted low-dimension space. This $\mathcal{E}(s): S \to S^-$ can lower the dimension of state space faced by the agent, and then it is possible for our model to obtain better results.

6.3 Future works

In order to tackle all aforementioned challenges and to build more robust and computationally efficient algorithms, some further works, such as embedding framework, multi-task RL, and environment improvements, may be useful.

6.3.1 Embedding framework for dimensional reduction

Since one main challenge we face is dealing with high-dimension portfolios, employing dimension reduction technics for preprocessing turns out to be a straightforward approach. After compressing the states, the agent may face easier learning tasks, which may save online training time and improve learning quality. This dimensionality reduction approach can be compared to extracting factors from the noisy market in portfolio management theories.

Autoencoder is a promising approach for pretraining the embedding space. Various types of autoencoders, including autoencoder [60], variational autoencoder [61], and Wasserstein autoencoder [62], are widely applied to unlabeled coding data. The encoding is 'supervised' by a corresponding decoder, whose objective is to regenerate the uncompressed data. To be specific, the encoder and decoder pair $(\mathcal{E}(s), \mathcal{D}(s^-))$ work cooperatively to improve the encoding efficiency by minimizing:

$$\mathcal{E}^*, \mathcal{D}^* = \arg\min_{\mathcal{E},\mathcal{D}} dist(\mathcal{E}(s), \mathcal{D}(\mathcal{E}(s))).$$

We have tried to train RL agents on the embedding space based on a Wasserstein

autoencoder. The *'Embedding the RL'* framework achieves higher training performance and lower time consumption (briefly illustrated in Table 6.1). Further extension work has been submitted to the UTD-24 journal, INFORMS *Journal on Computing.*

Table 6.1 Training time comparison for embedding and non-embedding

	State dimension	Experiment time (s)	Embedded dimension	Embedded experiment time
SEE. 50	459	2.70h	200	57min
CSI. 300	3011	17.64h	450	2.67h
CSI. 500	4672	23.18h	700	4.74h

6.3.2 Multi-task agent for robust learning

To cope with volatility and drawdown ratio, we propose to employ multi-task learning in the RL framework. Multi-task means that our RL agent may face a vector of rewards containing different objectives, and our algorithms need to balance these tasks. Some works have already addressed robust RL learning based on a multi-task strategy [55, 56, 57, 58]. These works achieve more stable results by sacrificing some profits from a single task. However, paying more attention to the variance and risks in the market is even more important to many investors, and this provides some place for further work in a multi-task setting.

Further exploration under our setting could include adding more objectives such as maximizing monthly return, minimizing weekly drawdown, and controlling daily variance. Besides, an important type of meta-learning algorithm named distillation could be implied in a multi-task manner [55]. All these can be considered in future projects.

6.3.3 Limit Order Books Market Setting

Real market trading rules in the A-share market differ from our experimental

environment. In a real trading scenario, the ***Limit Order Books*** (LOB) setting is closer to the exact trading environment. In this type of market, traders submit their actions (buy or sell) with preferred amounts for specific financial instruments. The lowest selling price is the ***ask*** price, and the highest-selling price is the ***bid*** price. Whenever there is an ask price lower than the bid price, a deal is executed based on their average. This LOB setting provides investors with much more market transactional information and links closer to the actual trading setting.

Sun et al. [59] have tried to explore a reinforcement trading facing LOB settings; however, considering trading only one piece of asset, the computation burden is too large to implement with portfolio management. Nevertheless, we can still briefly model how to set up the environment. For example, at a given moment t, we consider trading n types of stocks and can observe the top five selling and purchasing orders (ten prices and ten purchasable quantities in total), then the total environment would be $20n$. This setting may not be difficult to handle without 'T1' regulation, since the state space remains small. However, for a real-world trading environment in the A-share market, during each trading day the investor also needs to decide when to make a deal, and he/she also needs to decide on the offer price and quantity, which adds burdens to the action space.

REFERENCE

[1] Fama E F, French K R. The capital asset pricing model: Theory and evidence[J]. Journal of economic perspectives, 2004, 18(3): 25-46.

[2] Fama E F, French K R. Common risk factors in the returns on stocks and bonds[M]. University of Chicago Press, 2021.

[3] De Spiegeleer J, Madan D B, Reyners S, et al. Machine learning for quantitative finance: fast derivative pricing, hedging and fitting[J]. Quantitative Finance, 2018, 18(10): 1635-1643.

[4] Bengio Y, Courville A C, Vincent P. Unsupervised feature learning and deep learning: A review and new perspectives[J]. CoRR, abs/1206.5538, 2012, 1: 2012.

[5] Sutton R S. Generalization in reinforcement learning: Successful examples using sparse coarse coding[J]. Advances in neural information processing systems, 1996: 1038-1044.

[6] Zheng G, Zhang F, Zheng Z, et al. DRN: A deep reinforcement learning framework for news recommendation [C]. //Proceedings of the 2018 World Wide Web Conference. 2018: 167-176.

[7] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of go without human knowledge[J]. Nature, 2017, 550(7676): 354-359.

[8] Kober J, Bagnell J A, Peters J. Reinforcement learning in robotics: A survey[J]. The International Journal of Robotics Research, 2013, 32(11): 1238-1274.

[9] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484-489.

[10] Dietterich T G. Ensemble methods in machine learning[C]//International workshop on multiple classifier systems. Springer, Berlin, Heidelberg, 2000: 1-15.

[11] Oza N C, Russell S J. Online bagging and boosting[C]//International Workshop on

Artificial Intelligence and Statistics. PMLR, 2001: 229-236.

[12] Margineantu D D, Dietterich T G. Pruning adaptive boosting[C]//ICML. 1997, 97: 211-218.

[13] Svetnik V, Liaw A, Tong C, et al. Random forest: a classification and regression tool for compound classification and QSAR modeling[J]. Journal of chemical information and computer sciences, 2003, 43(6): 1947-1958.

[14] Nevmyvaka Y, Feng Y, Kearns M. Reinforcement learning for optimized trade execution[C]//Proceedings of the 23rd international conference on Machine learning. 2006: 673-680.

[15] Sutton R S, Barto A G. Introduction to reinforcement learning[M]. Cambridge: MIT press, 1998.

[16] Busoniu L, Babuska R, De Schutter B, et al. Reinforcement learning and dynamic programming using function approximators[M]. CRC press, 2017.

[17] Bouzy B, Chaslot G. Monte-Carlo Go reinforcement learning experiments[C]//2006 IEEE symposium on computational intelligence and games. IEEE, 2006: 187-194.

[18] Menache I, Mannor S, Shimkin N. Basis function adaptation in temporal difference reinforcement learning[J]. Annals of Operations Research, 2005, 134(1): 215-238.

[19] Kakade S M. A natural policy gradient[J]. Advances in neural information processing systems, 2001, 14.

[20] Konda V R, Tsitsiklis J N. Actor-critic algorithms[C]//Advances in neural information processing systems. 2000: 1008-1014.

[21] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning[C]//International conference on machine learning. PMLR, 2016: 1928-1937.

[22] https://openai.com/blog/baselines-acktr-a2c/

[23] Gu S, Lillicrap T, Sutskever I, et al. Continuous deep q-learning with model-based acceleration[C]//International conference on machine learning. PMLR, 2016:

2829-2838.

[24] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.

[25] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

[26] Gomber P, Haferkorn M. High frequency trading[M]//Encyclopedia of Information Science and Technology, Third Edition. IGI Global, 2015: 1-9.

[27] Moskowitz T J, Ooi Y H, Pedersen L H. Time series momentum[J]. Journal of financial economics, 2012, 104(2): 228-250.

[28] Jegadeesh N, Titman S. Cross-sectional and time-series determinants of momentum returns[J]. The Review of Financial Studies, 2002, 15(1): 143-157.

[29] Bollinger J. Bollinger on Bollinger bands[M]. New York: McGraw-Hill, 2002.

[30] Wold S, Esbensen K, Geladi P. Principal component analysis[J]. Chemometrics and intelligent laboratory systems, 1987, 2(1-3): 37-52.

[31] Sze V, Chen Y H, Yang T J, et al. Efficient processing of deep neural networks: A tutorial and survey[J]. Proceedings of the IEEE, 2017, 105(12): 2295-2329.

[32] Suykens J A K, Vandewalle J. Least squares support vector machine classifiers[J]. Neural processing letters, 1999, 9(3): 293-300.

[33] Tibshirani R. Regression shrinkage and selection via the lasso[J]. Journal of the Royal Statistical Society: Series B (Methodological), 1996, 58(1): 267-288.

[34] Breiman L. Random forests[J]. Machine learning, 2001, 45(1): 5-32.

[35] Tang J, Deng C, Huang G B. Extreme learning machine for multilayer perceptron[J]. IEEE transactions on neural networks and learning systems, 2015, 27(4): 809-821.

[36] Zaremba W, Sutskever I, Vinyals O. Recurrent neural network regularization[J]. arXiv preprint arXiv:1409.2329, 2014.

[37] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

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

[39] Dempster M A H, Leemans V. An automated FX trading system using adaptive reinforcement learning[J]. Expert Systems with Applications, 2006, 30(3): 543-552.

[40] Bisi L, Sabbioni L, Vittori E, et al. Risk-averse trust region optimization for reward-volatility reduction[J]. arXiv preprint arXiv:1912.03193, 2019.

[41] Cumming J, Alrajeh D D, Dickens L. An investigation into the use of reinforcement learning techniques within the algorithmic trading domain[J]. Imperial College London: London, UK, 2015.

[42] Deng Y, Bao F, Kong Y, et al. Deep direct reinforcement learning for financial signal representation and trading[J]. IEEE transactions on neural networks and learning systems, 2016, 28(3): 653-664.

[43] Moody J E, Saffell M. Reinforcement learning for trading[J]. Advances in Neural Information Processing Systems, 1999: 917-923.

[44] Moody J, Wu L, Liao Y, et al. Performance functions and reinforcement learning for trading systems and portfolios[J]. Journal of Forecasting, 1998, 17(5-6): 441-470.

[45] Pendharkar P C, Cusatis P. Trading financial indices with reinforcement learning agents[J]. Expert Systems with Applications, 2018, 103: 1-13.

[46] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529-533.

[47] Haarnoja T, Zhou A, Abbeel P, et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor[C]//International conference on machine learning. PMLR, 2018: 1861-1870.

[48] Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms[C]//International conference on machine learning. PMLR, 2014: 387-395.

[49] Hasselt H. Double Q-learning[J]. Advances in neural information processing systems, 2010, 23.

[50] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning[C]//Proceedings of the AAAI conference on artificial intelligence. 2016, 30(1).

[51] Barto A G , Sutton R S , Anderson C W . Neuronlike adaptive elements that can solve difficult learning control problems[J]. Systems Man & Cybernetics IEEE Transactions on, 1983, 13(5):p.834-846.

[52] Sutton R S, McAllester D, Singh S, et al. Policy gradient methods for reinforcement learning with function approximation[J]. Advances in neural information processing systems, 1999, 12.

[53] Fujimoto S, Hoof H, Meger D. Addressing function approximation error in actor-critic methods[C]//International conference on machine learning. PMLR, 2018: 1587-1596.

[54] Schulman J, Levine S, Abbeel P, et al. Trust region policy optimization[C]// International conference on machine learning. PMLR, 2015: 1889-1897.

[55] Teh Y, Bapst V, Czarnecki W M, et al. Distral: Robust multi-task reinforcement learning[J]. Advances in neural information processing systems, 2017, 30.

[56] Andreas J, Klein D, Levine S. Modular multi-task reinforcement learning with policy sketches[C]// International Conference on Machine Learning. PMLR, 2017: 166-175.

[57] Guo Z D, Pires B A, Piot B, et al. Bootstrap latent-predictive representations for multi-task reinforcement learning[C]// International Conference on Machine Learning. PMLR, 2020: 3875-3886.

[58] Wilson A, Fern A, Ray S, et al. Multi-task reinforcement learning: a hierarchical bayesian approach[C]// Proceedings of the 24th international conference on Machine learning. 2007: 1015-1022.

[59] Sun S, Wang R, He X, et al. DeepScalper: A Risk-Aware Deep Reinforcement

Learning Framework for Intraday Trading with Micro-level Market Embedding[J]. arXiv preprint arXiv:2201.09058, 2021.

[60] Ng A. Sparse autoencoder[J]. CS294A Lecture notes, 2011, 72(2011): 1-19.

[61] Kingma D P, Welling M. Auto-encoding variational bayes[J]. arXiv preprint arXiv:1312.6114, 2013.

[62] Tolstikhin I, Bousquet O, Gelly S, et al. Wasserstein auto-encoders[J]. arXiv preprint arXiv:1711.01558, 2017.

[63] Yang H, Liu X Y, Zhong S, et al. Deep reinforcement learning for automated stock trading: An ensemble strategy[C]//Proceedings of the First ACM International Conference on AI in Finance. 2020: 1-8.

ACKNOWLEDGMENT

During this graduation project, I received great support and encouragement from many important people. I would like to express my sincerest to my research thesis and graduation supervisor. His always help, support, and guidance throughout my admission and graduation seasons provide me with great encouragement to dive deeper into the field of operations and reinforcement learning. Starting working under his guidance last year, I have learned far more beyond reinforcement and machine earning knowledge and am deeply amazed and influenced by his life philosophy.

I would show my greatest gratitude to my research supervisor from UC, Berkeley IEOR. I started to work with him more than one year ago, and his careful guidance, warm support, and always willingness to help greatly influenced me and encouraged me to pursue further the intersection of operations, quantitative finance, and machine learning, which will be my further research interest and focus during my upcoming Ph.D. period.

I would also show my earnest thanks to my research advisors from Statistics & Data Science at Yale University and EECS at the University of Maryland. It is your help and guidance that instructs me to explore deeper into the field of theoretical reinforcement learning. Your patience, tolerance, and careful guidance led me to get familiarized with the domain of RL and provided me with the idea for my graduation thesis.

I would like to sincerely show my gratitude to my parents from the bottom of my heart. I have always been someone who may cause trouble and worries for you, but you have always been there for me. Thank you so much for creating such a warm and sweet environment for me to grow up in and become someone ready to step into society by himself independently. I love you forever and ever.

Finally, I would like to express my warmest thanks to my girlfriend. Your love, company, and encouragement give me the courage to overcome all difficulties and provide me with the harbor of my life's voyage. I cannot help imagining our bright, sweet and wonderful future.