基于目标分布学习的策略搜索算法及其量化应用

（申请清华大学工学硕士学位论文）

培 养 单 位 ： 交叉信息研究院

学　　科 ： 计算机科学与技术

研 究 生 ： 李 元 齐

指 导 教 师 ： 唐 平 中 副 教 授

二〇二〇年五月

Policy Search by Target Distribution Learning and Application in Quantitative Trading

Thesis Submitted to

Tsinghua University

in partial fulfillment of the requirement

for the degree of

Master of Science

in

Computer Science and Technology

by

Yuanqi Li

Thesis Supervisor : Associate Professor Pingzhong Tang

May, 2020

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容。

本人保证遵守上述规定。

（保密的论文在解密后遵守此规定）

作者签名：_____ 导师签名：_____

日　　期：_____ 日　　期：_____

摘 要

强化学习自近些年来与深度神经网络、大数据、高性能算力结合后，在很多领域大放异彩，其核心是凭借策略网络根据环境观测给出智能的动作决策。然而，在当前常见的基于梯度下降的非确定性策略更新方法中，存在着参数混合更新以及网络接近确定性策略时梯度会异常增大的问题，导致学习过程不稳定，使模型难以收敛到最佳策略。为此，本文提出基于目标分布学习的策略搜索算法（TDL 算法族）。算法用一种监督学习的方式，在通过策略搜索采样得到行动的目标分布和朝目标分布方向学习之间迭代，进行策略更新。我们的算法通过进化算法和自适应技术分别地提出目标分布的均值和方差，既能够有效的限定策略单次更新的幅度，又由于朝着固定的目标优化，训练过程更加稳定。算法在模拟实验平台（Mujoco）的性能优于或持平现有最优算法，同时能提升样本利用率，对超参数也更鲁棒。

另一方面，量化交易的发展，从早先基于诸如贝叶斯网络，马可维兹理论的统计方法，到近些年基于神经网络、决策树等有着更强大拟合能力的非线性模型，无不包含大量的人力监督。机器学习仅作为整个量化系统的组件，通常只给出方向性预测，实际的交易决策，仍需要大量的人工经验才能完成。而强化学习的出现使得传统框架得以突破，在某种程度上来说，强化学习是智能预测与智能策略的结合，必将成为量化交易的下一大热点。本文进一步地将提出的 TDL 算法用在数字货币高频交易上，提出一套完整的基于强化学习和监督学习的智能化和自动化交易框架，在模拟回测中取得的优异结果证明了算法的应用潜力。

关键词：强化学习；策略搜索；量化交易；人工智能驱动；

Abstract

In recent years, Reinforcement Learning, combined with the deep neural network, big data, and giant computing power, has made great success in many fields. The core of reinforcement learning is to propose intelligent action on given observation using its policy network. However, in common non-deterministic gradient descent algorithms, there is a problem that the gradient will increase abnormally when the network approaches the deterministic strategy, which leads to the instability of the learning process and makes it difficult converging to the optimal strategy. Therefore, we propose a new strategy search algorithm (TDL) based on target distributed learning. The algorithm uses a supervised learning style to update the strategy by iterating between obtaining target distribution by policy search and updating toward the target. Our algorithm can not only effectively limit the range of single update, but also make the training process more stable due to the optimization target is fixed. The performance of the algorithm in the experiment platform (mujoco) is better than or equal to the existing optimal algorithm, and more robust to hyper-parameter.

On the other hand, the development of quantitative trading, from statistical methods like Bayesian network and Markowitz theory to the nonlinear models like neural network and decision tree which have more powerful learning ability, are all filled with artificial supervision. Machine learning, serving as a component of the whole trading system, commonly, only gives price prediction. But translating them to the actual transactional operation can still only be completed by human. However, this traditional framework might be improved by reinforcement learning algorithms. To some extent, reinforcement learning is the combination of intelligent prediction and intelligent strategy, which will become the next hot spot of quantitative trading. In this paper, the TDL algorithm is further applied to the high-frequency transaction in the cryptocurrency market, and a set of intelligent and automatic transaction systems based on reinforcement learning and supervised learning is developed. The excellent results obtained in the backtest trading prove the application potential of the algorithm.

Key words: Reinforcement Learning; Policy Search; Quantitative Trading; AI-driven

Contents

Chapter 1 Introduction

1.1 Reinforcement learning

Reinforcement Learning (RL), derived from the Bellman optimality equation, is first proposed in the 1950s. In recent years, RL is combined with proficient deep learning techniques and giant computing power thus become the hottest spot in the domain of Artificial Intelligence. A Reinforcement learning agent updates its policy, a projection from observation to action space, by interacting with the environment with the target to maximize long term reward. It is believed that RL is a crucial part of realizing Artificial Intelligence since it renders an agent the ability to explore and learn, which imitates the learning process of human beings. Some researchers even find the similarity between big data RL learning and concept learning of human nature[1] and RL will be more and more explainable with the continuous progress of human brain science.

Equipped with the powerful expression ability of Deep Neural Network, intelligent decision ability and available big data, Deep Reinforcement learning algorithm has conducted excellent work on complicated control tasks, such as famous AlphaGO[2], Alpha Zero[3] from DeepMind which beat a professional player in GO game and computer video games, robot self-control problems[4], computer vision tasks[6], and it also plays a key role in the intelligent car and intelligent home system, plus, some combine RL with Mechanism Design[9] and apply it into auction[8] and pricing systems[10]. It is believed RL will play a greater role in all aspects of human life.

Research on reinforcement learning can be divided into different categories according to different classification methods. According to the update approach, reinforcement learning includes value-based methods such as DQN[12], which selects an action by estimated value, these methods achieve good performance in discrete action spaces. And methods based on policy gradient utilize deep neural networks to model policy and use optimized loss gradient to update network. It could further be divided into deterministic and indeterministic methods. These methods perform well on complex problems with continuous action spaces. Another category of methods is based on search and supervision, for example AlphaGo[2] which we mentioned above. These methods aim to utilize extra supervision information and policy search techniques to improve learning

efficiency[13].

In terms of optimization targets, it can be decomposed into representation, optimization method and task design(modeling)[14]. For example, some recent works show that simpler representations such as linear[15] or RBF representation[14] also work well for Mujoco tasks (a continuous environment designed for performance testing), some works focus on reward engineering or invent different kinds intrinsic rewards[17]. By doing so, tasks are either easier to learn or gain a better ability to explore. ES[15] and ARS[16] use policy search but their work only works for representations with few parameters (such as linear policy).

Most of the works focus on the optimization method. For continuous control problems, the most commonly used optimization methods based on the policy gradient. REINFORCE[21] is the most simple form of policy gradient algorithm where Q-values are estimated from Monte Carlo sampling. The baseline is used to reduce variance[23]. DDPG[24] solves continuous control tasks by a deterministic policy gradient method combined with target network and exploration noise techniques to ensure stability. A2C[25] is an actor-critic method where approximated value function is used in the policy network update. This greatly reduces variance which leads to faster learning, but introduces instability. Conservative policy update[26] is proposed to ensure policy improvement which indicates stability, followed by practical algorithms such as TRPO[27] and PPO[28]. Although SQL[29] and SAC[30] introduce different optimization methods, this work does not compare with them, since they work under maximum entropy framework which introduces a different policy representation. CMA-PPO[31] introduces ideas of ES into PPO, but it is in nature still a policy gradient algorithm.

Our work here belongs to the optimization method, in specific my work focus on determined distributional optimization in the classic Actor-Critic framework. In the rest of the section, based on A-C framework， we will start by introducing basic concepts and definitions of Reinforcement Learning for Continuous Control algorithms, and then talk about previous work in this direction.

1.1.1 Reinforcement learning for Continuous Control

Denote s_t, a_t as current state and the action taken by policy, and r_t, s_{t+1} as the observed reward and next state. Given a trajectory $(s_0, a_0, r_0, s_1, \dots, s_k, a_k, r_k, s_{k+1} \dots)$, the optimization objective of an RL algorithm is the cumulated reward:

$$J(s,a,\theta) = E[\sum_{t=0}^{\infty}\sum_{s'} p(s_t = s'|s_0 = s, a_0 = a, \pi_\theta)\gamma^t r_t] \quad (1\text{-}1)$$

Where π_θ is action policy $\pi_\theta(s,a) = P(a|s,\theta)$ with parameter θ which output the probability of a candidate action a at given state s, $J(s,a,\theta)$ is the cumulative reward and γ is the discount rate.

The Actor-Critic framework is the most popular and effective RL algorithm which combines the advantage of value-based network as well as the probabilistic action used by policy gradient. In the process of a game, Actor choose actions via policy and critic evaluates the value of each action, then two network updates collaboratively, where the value function is updated by TD error between the real value and estimated one, policy network is updated by calculated gradient including orientation and the value of each action. The Workflow is shown in Figure1.1.

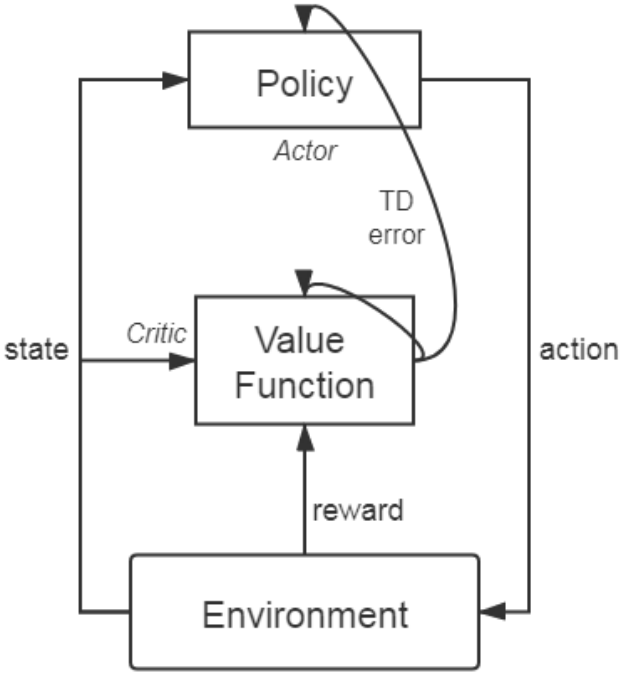

Figure 1.1 Workflow of Actor-Critic Network. Actor-Network interacts with the environment by taking action and gets the next state and reward. The Critic Network evaluates each action and calculates TD error. The two networks are updated collaboratively.

For the critic network, the value function is learned to estimate cumulative reward of state (V^π) or state-action pair:

$$V^{\pi}(s) := \mathbb{E}_{s_0=s} \sum_{t=0} \gamma^t r_{t+1}$$

$$Q^{\pi}(s,a) := \mathbb{E}_{s_0=s,a_0=a} \sum_{t=0} \gamma^t r_{t+1}$$

The two value function is related:

$$Q^{\pi}(s,a) := \mathbb{E}_{s' \sim p(s'|s,a)}[r(s') + \gamma V^{\pi}(s')]$$

For value-based methods, such as DQN[12], Dueling DQN[32], IQN[33], they only maintain critic network while keeping a deterministic policy that is always choosing the action with a maximum estimated value, for this kind of algorithms solving the optimal Q function is actually to find the optimal strategy.

The Critic network is updated via a standard supervised learning way, the learning target is:

$$y_t = r_t + \gamma Q_{\theta_q}(s_{t+1}, \pi_{\theta}(s_{t+1}))$$

Then the Critic network is updated by TD error:

$$minimize\ (y_t - Q_{\theta_q}(s_t, a_t))^2$$

On the other hand. For the update of actor-network, also known as the policy network, there are two main approaches. One is based on policy gradient, such as DDQG[24],A2C[25],ACER[34]. Denote π_{θ} as policy and Q_{θ_q} as value function. When given a batch of experience, the gradient are calculated by:

$$\nabla_{\theta}\eta(\pi_{\theta}) = E_{s\sim\rho_{\pi},a\sim\pi}[\nabla_{\theta} log\pi_{\theta}(a|s) Q_{\theta_q}(s,a)] \quad (1\text{-}2)$$

where

$$\rho_{\pi}(s) = E_{s_0}[\sum_{t=0}^{\infty} \gamma^t p(s_t = s|s_0, \pi)],$$

some methods replace the value function Q_{θ_q} with advantage function by subtracting the value of states $A_t(\mathrm{s,a}) = Q_{\theta_q}(s,a) - \mathrm{V(s)}$. Intuitively, the derivative term determines the update direction in parameter space and value term assigns weight to a given direction.

The derivative of policy π_{θ} to parameter θ indicates the direction of the update while value function endows weight in this direction. By running a BP algorithm, the θ is updated by:

$$\theta_{t+1} = \theta_t + \lambda\Delta\theta = \theta_t + \lambda\nabla_\theta\eta(\pi_\theta).$$

Unfortunately the step-size λ is usually hard to select since for a large λ, the algorithm might be fluctuating and fails to converge but a small one will incur too much time consumption. Plus, along the training process when the network tends to be steady, a fixed step-wise is not appropriate, some use decayed step-wise. However the change of step-size whether is human-designed or generate by rules is hard to choose either.

Another policy-based method bounds the update for policy into the trust region. Supported by the Conservative policy iteration theory (proposed in [26]), if it guaranteed that the change in policy space is not too large within a single update, we can make sure that the new strategy is better than the old one. Thus the

$$\eta(\pi_{new}) - \eta(\pi_{old}) \leq L_{\pi_{old}}(\pi_{new}) + k * \alpha^2,$$
$$Where\ L_{\pi_{old}}(\pi_{new}) = \sum_s \rho_{\pi_{old}}(s) \sum_a \pi(a|s) A_{\pi_{old}}(s,a),$$

and k is calculated by hyperparameters. In short, the single update range is controlled by α.

To avoid artificial determination of step size and approximate conservative policy iteration, TRPO[27] updates its model within the trust region. [28] adds hard constraint based on conservative policy update[26], then selects new policy by:

$$\pi_{\theta_{t+1}} = argmax_{\theta_{t+1}} \eta(\pi_{\theta_i}) + \sum_s \rho_{\pi_{\theta_i}} \sum_a \pi(a|s) A_{\pi_{\theta_i}}(s,a) \quad (1\text{-}3)$$

$$s.t.\ D_{kl}^{max}(\theta_{old}, \theta) \leq \delta$$

Where the constraint term is called the trust region, which is a hard constraint, D_{kl} is KL-divergence also called relative entropy which is employed to measure the distance between two distributions:

$$KL(P,Q) = \int P(x) log \frac{P(x)}{Q(x)} dx$$

PPO[28] eliminates the hard constraint for simplicity of calculation and use a soft constraint clip to prevent the policy updates too large (clip version):

$$L^{CLIP}(\theta) = \mathbb{E}_t[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)] \quad (1\text{-}4)$$

Where $r_t(\theta) = \frac{\pi_{\theta_{new}}(a_t, s_t)}{\pi_{\theta_{old}}(a_t, s_t)}$, and $\theta = maximize_\theta L^{CLIP}$.

1.1.2 Problems in Previous Methods

In this part, I will propose two drawbacks of previous methods, all in all, they are instability issues. To better propose our algorithm, I will introduce and illustrate the first problems, and leave the discussion of another in the main part of this thesis.

Let's first recall the presentation of policy in previous methods. No matter for DDPG or PPO, the policy is a Gaussian distribution modeled by a neural network:

$$\pi_\theta(a, s) = N(a; \mu_{\theta_1}(s), \Sigma_{\theta_2}(s))$$

Where the statistics: mean $\mu_{\theta_1}(\mathrm{s})$ and variance $\Sigma_{\theta_2}(s)$ of the probability distribution are mixed and updated simultaneously during updates.

The word 'mixed' means: firstly, the gradient of parameters is the implicit function, we are not able to directly see the update direction of each parameter in parameter space, secondly, the update of each parameter (mean and variance) are not separable, we don't know at what extend each parameter has changed. The control is not direct. Let's take PPO and TRPO for example for a detailed description.

Firstly TRPO and PPO employ plenty of work to constrain the change in the probability distribution of action between the old and new policy, which incurs a huge amount of calculation in probability space, such as TRPO calculates Hessian Matrix and PPO calculates the ratio of new to old parameters on each sample. Secondly, success in supervised learning and the instability of RL suggests that the supervised style is much more efficient in learning, network updated via supervised learning usually converge quickly and easier to learn in empirical, which for Actor-Critic network the parameter of value function converge faster and loss declines relatively smoother than policy function does. The problem is, although KL constraint is assigned on each sample, there still exists chanced that the mean of π_θ be pushed outside the safe region and still going outward pushed by other samples. Recall the update of PPO:

$$L^{clip} = E_t[min(\frac{\pi_{\theta_{t+1}}(a_t, s_t)}{\pi_{\theta_t}(a_t, s_t)}, clip(r_t(\theta_t), 1-\epsilon, 1+\epsilon)A_t)]$$

When $r_t(\theta) = \frac{\pi_{\theta_{t+1}}(a_t, s_t)}{\pi_{\theta_t}(a_t, s_t)} \in [1-\epsilon, 1+\epsilon]$, the mean is updated like:

$$f \leftarrow f + \lambda \frac{A_t}{\pi_{\theta_{old}}} \frac{\partial \pi}{\partial f} = f + \lambda \frac{A_t}{\pi_{\theta_{old}}} \frac{at - f}{g^2}$$

This equation reveals two disadvantages of the PPO update. First, when the updated

mean $f(s_t)$ is pushed outside the safe region, PPO desensitizes pushing it outward by a gradient from this sample but the chance remains that the mean $f(s_t)$ still going outward pushed by other samples as illustrated by the first sample on the left of figure 2. Updated means too far away from the old action mean will leads to large KL divergence between the updated and old policy which will bring instability. Second, when the advantage is negative, the updated mean is pushed to the other unexplored direction which brings additional instability as illustrated by the last two samples on the left of Figure 1.2. To overcome the drawbacks mentioned above, we intend to propose a target learning method that can separate the update of mean and variance and train them in a supervised way by setting a learning target.

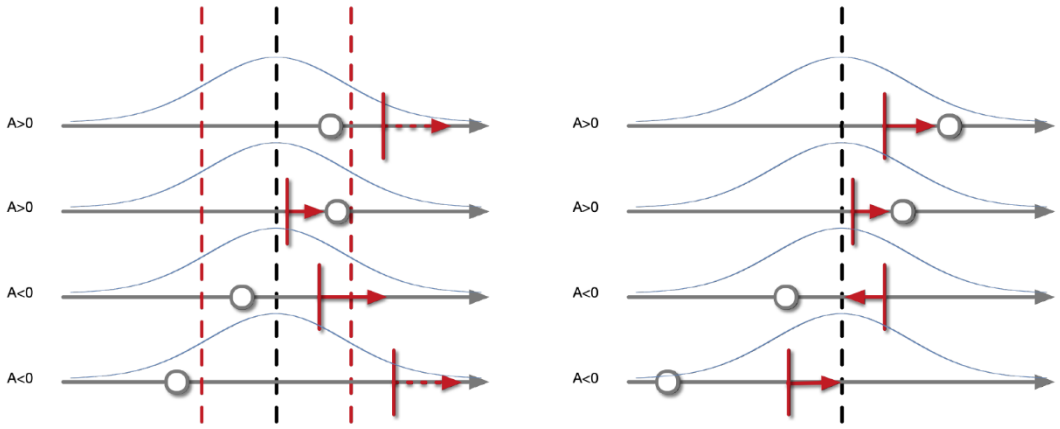

Figure 1.2 Left figure shows the mean update behavior of PPO and right shows the ideal update. Four samples (gray circles) are sampled from old action distribution and signs of sample advantages are labeled on the left. Black dashed line represents old action means and the red dashed line represents the equivalent clipping on action space. The red vertical line shows the updated action means of the new policy. Red arrow for the gradient update and the red dashed arrow for the possible update from the gradient of other samples.

It is worth mentioning that, ironically, in paper [73], which is newly published in ICLR 2020 and got full remarks, the author found what truly brings the improvement of PPO and bounds the update within trust region is not the gradient clip between the new policy $\pi_{\theta_{new}}$ and the old $\pi_{\theta_{old}}$, but is stem from some trick in code-level, which further supports our argument.

Besides the above, we will point out another instability issue literally and illustrate it by experiment results in chapter 2.

1.1.3 Mujoco Platform

Like other machine learning directions, Reinforcement Learning also has some classic experimental scenes, such as Mountain-car, Cart-pole, Humanoid, etc. Along with the rise of Deep Reinforcement Learning nowadays, more and more new and complex experimental scenarios emerged, like openAI gym, MuJoCo, DeepMind Lab, TORCS, and so on.

MuJoCo (Multi-Joint dynamics with Contact) is a simulator for the research of robot control, optimization, etc. The environments in MuJoCo are designed for continuous action control, where the RL agent controls a robot to keep running, swimming, or standing. Two typical humanoid environments are shown in the Figure below.

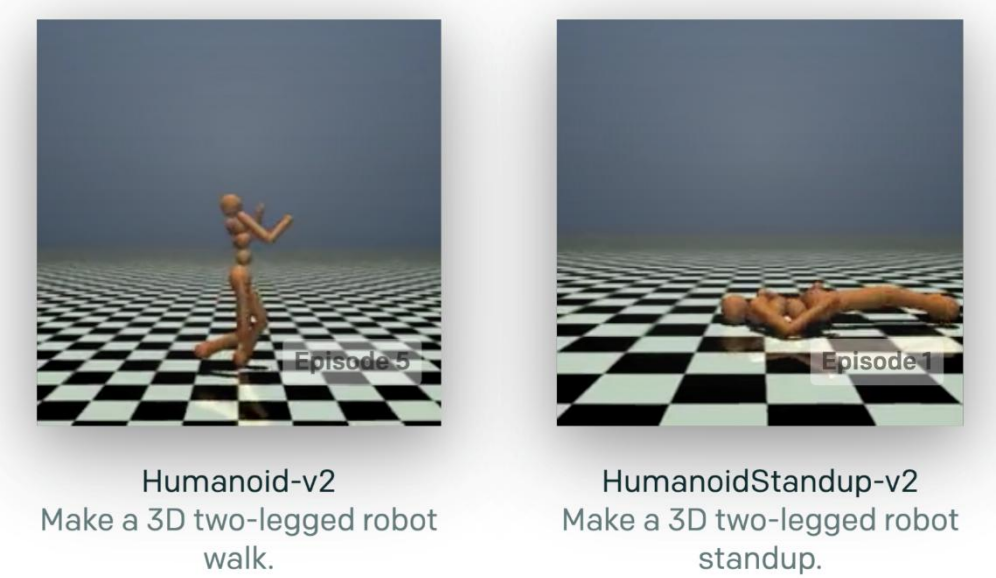

Figure 1.3 Humanoid environments in MuJoCo. These two are the hardest scenarios in MuJoCo because of the high dimension of action and state space, plus, the reward feedback is not in time which requires more exploration and stability of the algorithm.

To test an RL algorithm in MuJoCo, after import OpenAI gym package and load correspond environment, the RL agent first load state observation which has been reduced to the numerical array, then generate action according to policy π_θ then interact with the environment and get the next state observation and the reward of this action, then the agent will be updated from those (action, state, reward) trajectories by a specific learning algorithm. The MuJoCo platform provides an open and unified standard of experiments which helps us get feedback and compare with others. In this thesis, we conduct experiments on 13 MuJoCo experiments to compare our methods with other state-of-the-

art methods, the results will be shown in chapter 3.

1.2 AI in Quantitative Trading

In this part, I will first briefly introduce the history of Quantitative Trading and enumerate the common application of artificial intelligence algorithms in this field, which in general the investors replace artificial prediction with supervised learning models. Then some of the potential advantages of Reinforcement Learning will be discussed.

1.2.1 Quantitative Trading

Simply speaking, Quantitative trading refers to using an advanced mathematical model to replace human judgment and formulating strategies to select "high probability" events that can bring excess profits from huge historical data with the help of a computer.

Previously, the traditional Quantitative Trading approach or theory stems from Financial Mathematics. It is broadly believed the Pioneer of quantitative investment is Markowitz who established Modern portfolio management theory (PMT) based on the mean-variance model in 1952. For a given return, the Markowitz model is trying to form a portfolio with minimum risk, i.e. minimum covariance. For mathematical formula, when you have n risky assets with mean returns $\mu = (\mu_1, \mu_2, \dots, \mu_N)^T$ and covariance matrix $\Sigma = [\sigma_{i,j}]_{i=1\ to\ N, j=1\ to\ N}$, Markowitz model solves an optimization problem,

$$Min\ \frac{1}{2} w^T \Sigma w \ \ subject\ to\ \ w^T \mu = \mu_p, \mathbf{1}^T w = 1,$$

where w is the weight list of the portfolio. Later, Sharpe, Lintner and Mossion proposed the Capital Asset Pricing Model[35] independently. CAPM study the equilibrium structure of asset price and relates the expected return rate of an individual asset to its risk, in terms of the market portfolio. According to CAPM, the expected return of financial assets is determined by one factor, that is, the market excess return. If the market portfolio M is efficient, the expected return $E(R_i)$ of any asset i satisfies:

$$E(R_i) - R_f = \beta_i \big(E(R_M) - R_f\big) = \frac{\sigma_{iM}}{\sigma_M^2} \big(E(R_M) - R_f\big),$$

where β_i is the beta of asset i and $E(R_i) - R_f$ is the expected excess return of the market portfolio and $E(R_M) - R_f$ is the expected return of asset i.

After that, two important and basic theories are proposes. Samuelson and Famma

proposed the Efficient Market Hypothesis (EMH) in 1965. EMH that in a fully competitive market, all valuable message has been reflected in stock price timely, accurately and thoroughly, it is impossible for investors to obtain excess profits higher than the market average by analyzing the past prices unless there is market manipulation. Efficient Market Hypothesis is a basic theory to describe and understand the market, which has an important and far-reaching impact. Arbitrage pricing theory (APT)[36] is established by Ross in 1976, it says that an asset's expected return is influenced by a variety of risk factors. The APT models the return of an asset by a linear combination of different risk factors, that is:

$$R_i = E(R_i) + \beta_{i,1}F_1 + \beta_{i,2}F_2 + \cdots + \beta_{i,H}F_H + e_i,$$

where R_i is the rate of return on asset i and $E(R_i)$ is the expected value, $\beta_{i,\mathrm{k}}$, F_{k} represent the sensitivity of the asset to kth factor and kth common risk factor respectively, e_i, is the unsystematic return for assets.

Plus, there are many other models to either create or as a supplement to various fields in quantitative investment. Like Option Pricing Model[37], VAR(value at risk), MM (Modigliani and Miller) theory, and so on.

Among all Quantitative models, the multi-factor model is one of the most frequently used models for financial asset pricing and market prediction. The multifactor model prices the asset or predicts the market movement based on multiple features (or factors), each of which reflects a type of risk or market anomaly. The multi-factor model is employed to mine alpha return, also known as, market abnormal return. Fama and French first propose a three-factor model[38], they found the beta in CAPM cannot fully explain the difference of return rate of each stock, instead, it stems from other factors for example PB, PE, and Market Value of that company. In today's perspective, the Fama-French model is just a linear regression and the so-called alpha factors are the feature. Indeed, the Multi-factor model is inherited and further developed by modern science and technology.

1.2.2 AI-Driven Algorithm and Cryptocurrency Market

In modern applications, the main role of AI algorithms is to predict the direction of price fluctuation, or simply speaking whether the price is rise or drop, the prediction targets may be in different forms, for example, the rate of return, the rank in the whole stock market and so on. Then investors will embed these prediction signals into their

strategies in a different market or with different time-frequency. It can be said that common machine learning algorithms in nowadays quantitative trading are all developed from the multi-factor model. The improvements in the modern machine learning approach are mainly in two aspects: feature engineer and model representation.

But before going into concrete applications, we need first to discuss the predictability of machine learning in the financial market. Recall that EMH indicates that it is impossible to make economic profits based on market information. However, several market anomalies such as reversal[39] and momentum effect[40] have been found later in empirical studies. The market anomalies refer to the situations when the asset price economically deviates from the pricing model deduced from market efficiency. For more market anomalies, we refer the readers to[41]. These anomalies may stem from a variety of reasons such as the psychology or irrationality of the market participants which indicates that there are chances that we can exploit the history of the market and extract patterns to predict the movement of the market profits based on the market information.

Nevertheless, it is hard to exploit the historical data from the financial market to predict the future and make profits due to the following reasons. First, financial data is highly noisy and volatile. This is partially due to the existence of "noise traders" (or irrational traders)[42] or hidden factors that impact the movement of the market, such as government policy changes and breaking news. Second, the movement of the market is driven by multiple patterns originated from either different investing styles of the traders (e.g., value investors, technical traders) with different trading frequencies, or different psychological phenomena. In a nearly efficient market, exploiting one single pattern can hardly cover the friction (e.g., transaction fees, taxes, etc.) and make profits. Third, a good model for prediction does not necessarily lead to a good trading strategy. Conventionally, we evaluate a prediction model by the accuracy of all the samples. However, we do not trade on all these samples (e.g., all the stocks in the same cross-section or all the possible trading points), due to various reasons such as risks, liquidity, and certain regulations. For a trading strategy, we care more about the accuracy of the samples that trigger trading signals. Return to major two improvements of modern approach: feature engineer and model representation. Some predictability issues have been conquered or relieved by current algorithms, however, there still exist lots of core issues to be addressed.

For AI-driven feature engineering, two main reasons facilitate its emergence. First, the explainable factors with practical financial significance have been dug up by investors,

for instance the earnings' yield[44], the book-to-market ratio[45], etc., the remaining alpha of this part become limited up. Second, the complicated non-linear AI model also needs the support of diversified features. Meanwhile, the availability of big data and giant computing power also brings convenience to AI algorithm. Some scientist automatically generates features through data mining, for example[46] employ a modified genetic algorithm to mining alpha factors, 4paradigm develop *TemporalGo* to automatically generate temporal features and *FeatureGo* to automatically combine basic features to high-level features, some use AutoEncoder[47] to extract useful information of the whole dataset in an unsupervised way. Apart from feature generation, lots of AI-driven feature selection methods have been proposed, for example, lots of investors use the score of information gain in boosting tree model, some method analysis shuffled importance of each feature in deep neural network[48]. Automatic feature engineering broadly belongs to AutoML, and quantification has attracted more and more attention in this field.

For model representation, a typical AI model $F_\theta(X)$ which is parameterized by θ learns a mapping from feature space X i.e. various trading factors to target space Y i.e. the trend of stock future price. Conventionally, linear regression is the standard algorithm for the multifactor model. Such linear models are simple, robust and easy to interpret. But the linear combination has poor expression and fitting ability. Recently, with the development of AI algorithms, different forms of non-linear $F_\theta(X)$ become popular due to their larger model capacity to discover complex patterns such as gradient boosting decision trees like XGBoost and LightGBM[50], deep learning models like MLP, LSTM and so on[49] or deep reinforcement learning models. At present, a relatively mature method used in trading is in a supervised way that is the model gives the prediction first then the investment portfolio is constructed according to the prediction results. In this case, the goal of machine learning is to maximize prediction accuracy.

But reinforcement learning is different. Reinforcement learning directly outputs position weight or trading operation, to maximize profits. That is to combine the specific tasks from the downstream with the upstream model, this idea is very similar to the development of Natural Language Processing. The NLP methods, in the early days, usually transform each language word to feature vector first for example use Word2Vec[58], then fit vectors into RNN model or other sequential processing models. Afterward the BERT[59] came out, it directly applied specific tasks into model training process and overwhelmingly defeat all methods. The RL algorithms may enjoy several

advantages: first, the financial market is relatively chaotic and the signal-to-noise ratio is low, so it is very difficult to predict accurately. However, it is relatively easier to realize to strengthen learning through the interaction with the environment and directly explore the relationship between factor market and future earnings. Second, the traditional model will give predictions under any environmental conditions, so it must be combined with risk control, timing model, etc., and reinforcement learning can make the decision not to buy when the market is unstable or uncertain. Third, the Quantitative Trading scenes are very suitable for RL framework. Notice that the objective (1-1) resembles the Long-term profits in Economics where γ is the discount rate of the money, and the state and action are naturally corresponding to market and trade operations. Furthermore, Reinforcement Learning can operate with complex or indeterminate tasks in which the label is ambiguous for a supervised learning model or even hard to find appropriate label assignment, for example how to execute trading order, it is hard to formulate this kind of tasks under supervised learning framework. In general, reinforcement learning can be seen as intelligent prediction and intelligent strategy, or in other words, the RL framework further learns a policy from prediction to execution based on the map from data distribution to prediction target. In the author's scope of knowledge, the application of RL in quantitative trading is focused on high-frequency trading tasks in both the stock market as well as cryptocurrency market and can be in the late '90s of last century[87]. The application can be broadly divided into three domains: first, the Portfolio Management tasks[51], where the agent manages to trade security by timely adjusting the position weight of each stock. Second, the order execution problem[83], where the agent is trying to trade at least or exactly V volume in timespan T with minimum average price, normally the final execution prices ought to beneath the VWAP or TWAP(Volume or Time Weighted Average Price). The third application is market making[74] In this case the main goal of the RL agent is to trade at least V volume in certain period T to bring trading liquidity meanwhile seeking profit via arbitraging through the liquidity and keeping a low inventory risk. In this thesis, we aim to solve the multi-security portfolio management problems in the cryptocurrency market.

The blockchain technology or cryptocurrency like BTC, ETH, is the hottest fields nowadays, but in this thesis, we only treat them as a commodity and don't care the trading or confirm mechanism behind them. The reason we choose Cryptocurrency Market to conduct our experiments is that cryptocurrency trading is more 'friendly' to Machine

Learning tasks. Because, firstly traders around the world can trade between different cryptocurrencies in 24 hours a day in the cryptocurrency exchange like Okex, that's to say the data volume is much higher than stock market where people can only trade 4 hours per day and only open in weekday. Secondly, the API to either query real-time data or send trading orders is complete and convenient while in Chinese stock market automatic trading especially in high-frequency trading is not allowed. Thirdly, the price curve is more volatile so there are more trading opportunities, what's more, the ratio of algorithmic trader is higher thus the market is more regular.

1.3 Thesis Organization

The thesis is organized as follows: In chapter 2, we will first point out the instability issue of current method by experiment results, then formally propose our Target Learning Algorithm. We will introduce three forms of our methods which are TDL-direct, TDL-ES, and TDL-ESr and these three methods are the revised version of the former one in turn. All of the three methods efficiently set a learning target within the KL trust-region thus guaranteed policy improvement. The last two algorithms select learning targets by an update rule of ES (evolutionary strategy)[60], while different from previous methods[61] which conduct ES to search in parameter space, we employ ES to search a better action distribution. In chapter 3, we conduct extensive experiments on 13 continuous control tasks on Mujoco platform. According to results, our method is comparable to state-of-the-art algorithms where our methods outperform in some tasks and not significantly weak than the best methods in all tasks. Meanwhile, we also found our methods are more efficient to impose restrictions in policy space than TRPO and PPO, further, out method is more robust of sample reuse thus improve data efficiency thanks to its supervised way of learning. In chapter 4, we propose our automated trading framework which mainly uses TDL methods to provide portfolio weight and further filtrate by supervised learning signal. We introduce the whole process in detail include data process, model training, and trade setting. And we conduct back-test trading in the simulation market. we summarize the thesis and discuss the current result and possible future research directions in chapter 5.

The main contribution of this thesis is as follows: 1) we point out the instability issue of current models by both theoretical analysis and experimental proof and proposed a new policy improvement method TDL to conquer them. Our methods first apply ES in action

space and update policy networks in a supervised way. And extensive experiments are conducted on mujoco platform to demonstrate the ability of our algorithm. 2) We describe a completely AI-driven trading framework in detail from data process, model training, and translate signal into trading. We propose a way to combine Reinforcement Learning model and Supervised Learning model into trading systems. Plus we also design several reward functions to realize controllable risk-tendency. The excellent results obtained in the backtest trading prove the application potential of the algorithm.

Chapter 2 Target Distribution Learning

2.1 Instability Issue in Policy Gradient Methods

In 1.1.2, we have already revealed the instability issue of current policy gradient method, where the mixed update of mean $\mu_{\theta_1}(s)$ and variance $\Sigma_{\theta_2}(s)$ might push action distribution to an unsafe region even with KL divergence restriction. In this part, we will go beyond to discuss the gradient explosion problem when policy near deterministic.

Let's rewrite the update function within trust-region bound. Assume the new policy $\pi_\theta(a,s) = N(a;\mu_{\theta_{new}}(s),\Sigma_{\theta_{new}}(s))$, in one iteration, is updated from the old policy $\pi_{\theta_{old}}(a,s) = N(a;\mu_{\theta_{old}}(s),\Sigma_{\theta_{old}}(s))$. In TRPO[27], the policy network is going to maximize the following objective within KL divergence:

$$L(\theta) = \mathbb{E}_{s\sim\rho_{\pi_{old}}}\left[\sum_a N\left(a;\mu_{\theta_{new}}(s),\Sigma_{\theta_{new}}(s)\right)A^{\pi_{old}}(s,a)\right]$$

$$s.t.\ max_{s\epsilon S}KL(N(\mu_{\theta_{new}}(s),\Sigma_{\theta_{new}}(s)||N(\mu_{\theta_{new}}(s),\Sigma_{\theta_{new}}(s))) \le \delta$$

Where $\rho_\pi(s) = \mathbb{E}_{s_0}[\sum_{t=0}\gamma^t p(s_t = s|s_0,\pi)]$ and $KL(.||.)$ indicates the KL divergence of two probability distributions. According to [26], when δ is small, policy improvement is guaranteed in this iteration, that is $\eta(\pi_{new}) > \eta(\pi_{old})$. We can further approximate the above objective by MC (Monte Carlo) samples:

$$\hat{L}(\theta) = \frac{1}{T}\sum_1^T[\widehat{A_t}\frac{N(a_t|\mu_{\theta_{new}}(s_t),\Sigma_{\theta_{new}}(s_t))}{N(a_t|\mu_{\theta_{old}}(s_t),\Sigma_{\theta_{old}}(s_t))}]$$

Where s_t, a_t are samples on the trajectory $(s_0,a_0,r_0,s_1,\dots,s_k,a_k,r_k,s_{k+1}\dots)$ at time t and follows old policy. $\widehat{A_t}$ is an estimation of $A_t(s_t,a_t) = Q_{\theta_{old}}(s_t,a_t) - \mathrm{V(s)}$ i.e. advantage function at time t. There are many methods to generate $\widehat{A_t}$ and a popular choice is GAE (generalized advantage estimator)[64].

Then we show that when $\Sigma_{\theta_{new}}$ is small, i.e. the policy tends to be deterministic, the gradient of objective (2-1) for θ will explode which leads to an unstable training process.

Let's consider a case that the parameter $\Sigma_{\theta_{new}}$ is state-independent, that's to say the standard deviation of action distribution degenerates into a variable, denote as σ. In

this case, the (2-1) tends to be :

$$\hat{L}(\theta) = \frac{1}{T}\sum_{1}^{T}[\widehat{A_t}\frac{N(a_t|\mu_{\theta_{new}}(s_t),\sigma_{new})}{N(a_t|\mu_{\theta_{old}}(s_t),\sigma_{old})}]$$

Thus the gradient of $\hat{L}(\theta)$ with aspect to θ is:

$$\frac{\partial\hat{L}(\theta)}{\partial\theta} = \frac{1}{T}\sum_{1}^{T}\left[\widehat{L_t}(\theta)\frac{\partial logN(a_t|\mu_\theta(s_t),\sigma)}{\partial\mu_\theta(s_t)}\frac{\partial\mu_\theta(s_t)}{\partial\theta}\right]$$

Note that the gradient of the mean of the logarithm probability density (the first fraction term) is $(a_t - \mu_\theta(s_t))/\sigma^2$, thus :

$$\frac{\partial\hat{L}(\theta)}{\partial\theta} = \frac{1}{T}\sum_{1}^{T}\left[\widehat{L_t}(\theta)\frac{\partial\mu_\theta(s_t)}{\partial\theta}\frac{(a_t - \mu_\theta(s_t))}{\sigma^2}\right] \propto \frac{1}{\sigma^2}$$

So, the gradient with aspect to θ is an inverse correlation to the standard deviation of old policy and in a square order. Obviously, in the later stage of training, when policy is near deterministic i.e. $\sigma \to 0$, the gradient of θ in this iteration is large which will push the mean of action distribution $\mu_\theta(s_t)$ to a place far from previous deterministic one $\mu_{\theta_{old}}(s_t)$, which might already be an optimal solution. Then the next sample action might be 'bad' actions, and will inversely push policy to the optimal region. One can image in if we use gradient descent method but with a large step size, the solution will fluctuate around the optimal. Unfortunately, other policy gradient-based algorithms built on Actor-Critic framework may suffer the same issue because they can't avoid using the gradient of the probability density function. Some works found similar results to ours, like [64], which points out the same relation in gradient update of REINFORCE[21] and PEPG[65].

We design a toy example to illustrate the instability issue of PPO we mentioned above. Consider this simple scenario, the state is unrelated from action, the environment randomly samples a state from $s_t \sim U([0,1])$, and the cost of an action is $c(a) = a^2$. Although the cost is state-independent it is still fed into policy network. The goal of an RL agent is to minimize the one-step cost, that is $min_\theta[\pi_\theta(a|s)c(a)]$. Clearly, in this setting, the optimal policy should be deterministic: always play $a = 0$ with probability 1 at any state. According to experiment results, PPO suffers an oscillating and diverging learning process but our methods TDL (see next section) avoid using computation of

gradient and can easily find a better target thus not suffer from aforementioned instability issue. Notice that we adopt some common and non-essential tricks in PPO like setting a lower bound of the variance of the action distribution to avoid the variance goes to small, or adding an entropy regulation in loss functions, or setting a small clip in PPO. These tricks make no contributions to stabilize the learning process. The results are shown in Figure 2.1. As shown in the figure, PPO fails to either converge to deterministic policy or learn an optimal mean of action distribution.

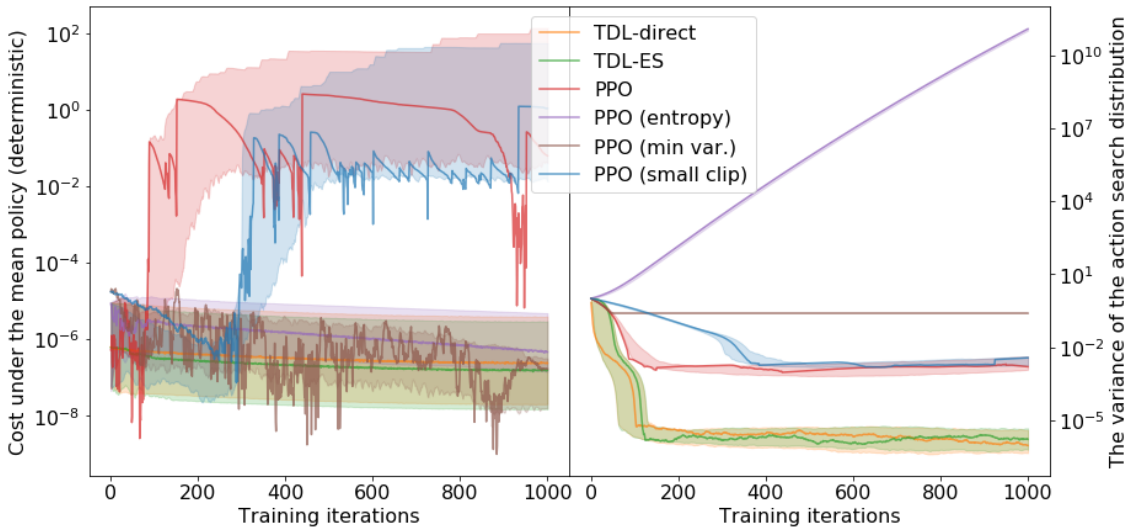

Figure 2.1 The performance of toy example. We independently run 100 times for each method, the solid line is median performance and shaded areas indicate the ranges from 10% to 90% quantiles. The left figure is the cost by executing the mean of action distribution in each step. The right figure is the variance of action distribution along with the training.

2.2 Target Distribution Learning

Here we propose Target Distribution Learning algorithm which separates the update of mean and variance and train the network by supervised learning. The algorithm alternate between two steps: In first step, instead of directly optimizing $L(\theta)$, TDL first proposes statistical parameters (target) of action distributions where the expected advantage function under the old policy is improved. Then in second step, the policy network is trained to match proposed target parameters (target action distribution).

To be specific, in first step, TDL for each state sample s_t proposes target parameters which is trying to maximize the surrogate objective of s_t , that is:

$$L_{t,1}(\mu,\sigma) = \mathbb{E}_{a\sim N(\mu,\sigma)}[A^{\pi_{old}}(s_t,a)] \quad (2\text{-}1)$$

Or maximize the probability that proposed target distribution get improved with old value function:

$$L_{t,2}(\mu,\sigma) = \mathbb{E}_{a\sim N(\mu,\sigma)}[\mathbb{I}\{A^{\pi_{old}}(s_t,a) > 0\}]$$

Both the optimization of $L_{t,1}$ and $L_{t,2}$ are subject to KL constraint:

$$KL(N(\mu_{\theta_{new}}(s_t),\sigma_{\theta_{new}}(s_t))||N(\mu_{\theta_{old}}(s_t),\sigma_{\theta_{old}}(s_t))) \leq \delta$$

The second step turns into a regression problem, where the network is trained to minimum MSE or MAE between the output and target parameters.

TDL independently proposes mean and variance target thus different search methods can be implemented on each parameter. For policy search in first step, note that only the estimation $\widehat{A_t}$ is known thus the solution might be biased. In second step, there is a trade-off during the update, that is to explore i.e. keep variance relatively large or to explore i.e. move to the recommend point exactly and shrink the variance, this trade-off in our algorithm will be controlled by certain hyper-parameter.

Let's consider a simpler form of Gaussian distribution, diagonal Gaussian distribution and isotropic Gaussian distribution. For diagonal Gaussian distribution, the variance $\sigma_\theta(s) \in R^n$ is a vector with the dimension the same as that of action. For isotropic Gaussian distribution, the variance $\sigma_\theta(s) \in R$ is a scalar, where the variance is the same across all the dimensions of action. This is an on-policy reinforcement learning algorithm. Given a trajectory $\{s_t, a_t\}$, each action a_t is sampled from $N(\mu_{\theta_1}(\mathrm{s}), \sigma_{\theta_2}(s))$, where $\mu_{\theta_1}(s_t), \sigma_{\theta_2}(s_t)$ are outputs of the policy network. $(1,1)-$ ES[67] is performed for each state-action pair (s_t, a_t). Target mean and variance are $\hat{\mu}_t, \hat{\sigma}_t$ set. θ_1 is updated to minimize $(\hat{\mu}_t - \mu_{\theta_1}(s_t))^2$ and θ_2 is updated to minimize $(\hat{\sigma}_t - \sigma_{\theta_2}(s_t))^2$.

Based on ES we propose two forms of TDL algorithms, the difference of the first two algorithms is the selection of the mean target. We first show the update of variance which is the same in two methods. We use self-adaptation[67] method. The updating rules allow us to explore adaptably that is changing the variance slowly to prevent premature convergence and violation of constraints.

In popular policy representations, the variance of action selection is independent of states. Therefore $\hat{\sigma}_t(s_t) \to \hat{\sigma}_t, \sigma_{\theta_2}(s_t) \to \sigma$ which itself is a parameter of the policy network.

For the update of isotropic Gaussian distribution:

$$y_i \sim N(\mathbf{0}, \boldsymbol{I}) \;\; \xi_i \sim N(0, 1/\sqrt{2n}) \quad a_i = \mu_{\theta_t}(s_i) + \bar{\sigma}_t exp(\xi_i) y_i$$

$$\bar{\sigma}_{t+1} = \bar{\sigma}_t exp(\frac{1}{N}\sum_{i=1}^{N} \xi_i \, \mathbb{I}\{\widehat{A}_i \geq 0\})$$

For the update of diagonal Gaussian distribution:

$$y_i \sim N(\mathbf{0}, \boldsymbol{I}) \quad a_i = \mu_{\theta_1}(s_i) + \bar{\sigma}_t y_i$$

$$\bar{\sigma}_{t+1} = \bar{\sigma}_t \sqrt{\frac{1}{N}\sum_{i=1}^{N} y_i{}^2 \, \mathbb{I}\{\widehat{A}_i \geq 0\})} \quad (2\text{-}2)$$

Intuitively, the variance contour will stretch in the direction of where advantage samples locate meanwhile compress the direction where the advantage samples are sparse. This uniform updates of variance that uses mean of all samples play a positive role in stabilizing the training process and helping impose a restriction, while on the other hand it also limits the solution space and upper bound of the algorithm.

Meanwhile, represented by a neural network, the variance can also be dependent on states. If so, the update of isotropic Gaussian distribution becomes:

$$\hat{\sigma}_{i,t+1} = \hat{\sigma}_{i,t} exp(\xi_i \mathbb{I}\{\widehat{A}_i \geq 0\})$$

The diagonal Gaussian distribution becomes:

$$\hat{\sigma}_{i,t+1} = \hat{\sigma}_{i,t} y_i \mathbb{I}\{A_i \geq 0\} + \hat{\sigma}_{i,t} \mathbb{I}\{\widehat{A}_i < 0\} \quad (2\text{-}3)$$

We balance the advantages of both state-dependent and state independent methods by taking a geometric weighting combinations of two candidates.

$$\sigma_{i,t+1} = \hat{\sigma}_{i,t+1}{}^{1/(\varphi+1)} \bar{\sigma}_{t+1}{}^{\varphi/(\varphi+1)}$$

Where φ is a trade-off factor and in later experiments in Mujoco, we set $\varphi = 1$.

2.2.1 TDL-ES Algorithm

For the update of the mean target, we use (1,1)-ES, where the sample with better fitness value between the parent and the offspring will be updated as the new mean. The action mean of the old policy serves as the parent and the sampled action serves as the offspring. The advantage of the sampled action serves as the relative fitness function of the sampled action over the action mean of the old policy. Given the sampled action, state and estimated advantage $(s_t, a_t, \widehat{A_t})$ the target is proposed by:

$$\mu_{t+1}(s_t) = \mu_t(s_t) + \varepsilon \mathbb{I}\{\widehat{A_t} > 0\}(a_t - \mu_t(s_t)) \quad (2\text{-}4)$$

Where ε is step-size. Intuitively, the mean will move towards a 'good' sample and remain static when the sample is a bad suggestion. Note that the objective $L_{t,1}(\mu, \sigma)$ is intrinsically same as the objective of (1,1)-ES when setting $\mu_{old} = \mu_t(s_t)$ and $x = a_t$ and $\widehat{A_t}$, which is the estimation of $Q^{\pi_{old}}(s_t, a) - V^{\pi_{old}}(s_t)$ is essentially the same as $f(x) - f(\mu_{old})$, so we can make a safe conclusion that the update rule of TDL optimize $L_{t,1}$.

Our algorithm has two important properties. Firstly, TDL-ES can be efficiently bounded within trust-region by KL divergence thus, according to [27], it is in accord with the conservative iteration process.

In our method, KL divergence on sample (s_t, a_t) can be written as:

$$KL_t = KL[N(f_{old}, g_{old}) || N(f, g)]$$

$$= \frac{1}{2}\sum_i [2log\left|\frac{g_i}{g_{old_i}}\right| + \left((\frac{g_{old_i}}{g_i})^2 - 1\right) + (f_i - f_{old_i})^2/{g_i}^2]$$

We can suppose $\left|g_i/g_{old_i}\right| \in [1-\epsilon, 1+\epsilon]$, where $\epsilon \ll 1$ and $f \approx \hat{u}_t$, we have

$$E_t[KL] \leq \frac{1}{2}[\varepsilon^2 n + o(\epsilon^2)]$$

Where n is the dimension of action space, thus we can safely bound KL by setting step-size ε as a small value.

Secondly, the update of target mean and variance is equivalent to a single step of stochastic gradient descent. Rewrite (2-3) to replace y_i by a_i, μ_{θ_t}:

$${\hat{\sigma}_{i,t+1}}^2 = (a_t - \mu_t(s_t))^2 \mathbb{I}\{\widehat{A_\iota} \geq 0\} + {\hat{\sigma}_{i,t}}^2 \mathbb{I}\{\widehat{A_\iota} < 0\} \quad (2\text{-}5)$$

Thus, (2-4), (2-5) can be regarded as SGD step of $L_{t,2}$ with some step-sizes $\lambda_\mu, \lambda_\sigma$:

$$\mu_{t+1} = \mu_t(s_t) + \lambda_\mu \frac{\partial L_{t,2}(\mu, \hat{\sigma}_{i,t}(s_t))}{\partial \mu}|_{\mu=\mu_t(s_t)}$$

$${\hat{\sigma}_{i,t+1}}^2 = \hat{\sigma}_{i,t}(s_t)^2 + \lambda_\sigma \frac{\partial L_{t,2}(\mu_t(s_t), \sigma)}{\partial \sigma}|_{\sigma=\hat{\sigma}_{i,t}(s_t)}$$

The policy update in a single step based on TDL-ES is shown in Figure 2.2.

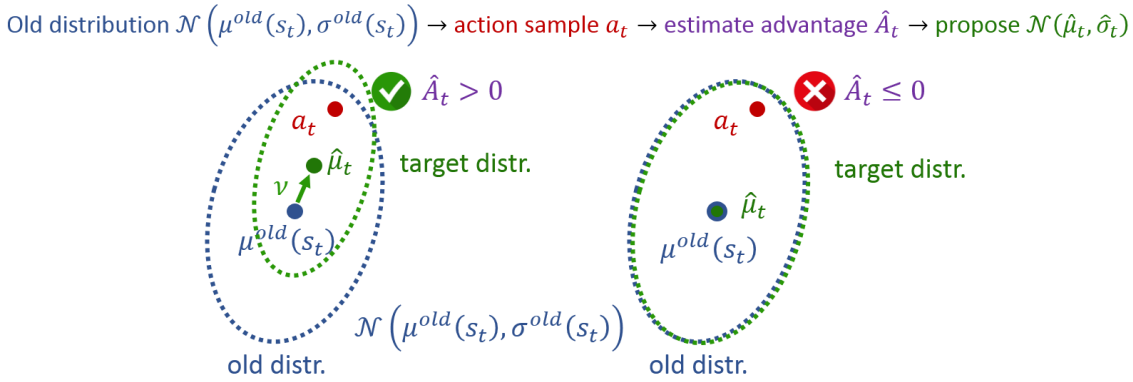

Figure 2.2 Policy update in single step based on TDL-ES. Left: 'good' target with positive advantage, so the action distribution moves towards that sample. Right: 'bad' target whose advantage beneath 0, the policy doesn't update.

2.2.2 TDL-ESr Algorithm

Due to the inherent property of MDP, it is infeasible for one RL agent to run in parallel thus there is only one offspring on a trajectory. So in TDL-ES we set the target solely according to the original sample and its offspring, meanwhile, we ignore the temporal structure of MDP. Because state observation and action distribution doesn't change too much in adjacent, we revise TDL-ES algorithm with more information in adjacent samples, we call this method TDL-ESr.

Firstly, we replace the target mean with the weighted average target mean of neighboring 2N+1 points $\{a_{t-N}, a_{t-N+1}, \dots, a_t, a_{t+N-1}, a_{t+N}\}$:

$$\mu_{t+1}^{target\prime} = \sum_{i=-N}^{N} \frac{\mathbb{I}(\widehat{A_{t+i}} > 0)\widehat{A_{t+i}}}{\sum_{i=-N}^{N} \mathbb{I}(\widehat{A_{t+i}} > 0)\widehat{A_{t+i}}} a_{t+i}$$

We change the update of target mean for two reasons: on the one hand, due to the similarity of adjacent states, we can assume that several consecutive actions are within the same distribution thus the (1,1)-ES evolves into (1, N)-ES, where N is the number of neighbors. On the other hand, if the difference is large between successive actions, the magnitude of the new target mean will be small even might close to 0 and the average direction is random. On the contrary, if the successive target actions are in the same direction, this direction will be updated. That's to say, we are more willing to use samples with similar neighbors because this direction is more reassuring and treats others as a

disturbance.

Secondly, we also add a step size parameter η to balanced the update, so that the final target means is calculated by:

$$\mu_{t+1}^{target} = \eta\mu_t(s_t) + (1-\eta)\sum_{i=-N}^{N} \frac{\mathbb{I}(\widehat{A_{t+i}}>0)\widehat{A_{t+i}}}{\sum_{i=-N}^{N}\mathbb{I}(\widehat{A_{t+i}}>0)\widehat{A_{t+i}}} a_{t+i} \quad (2\text{-}6)$$

The update process in one step of TDL-ESr is illustrated in Figure 2.3.

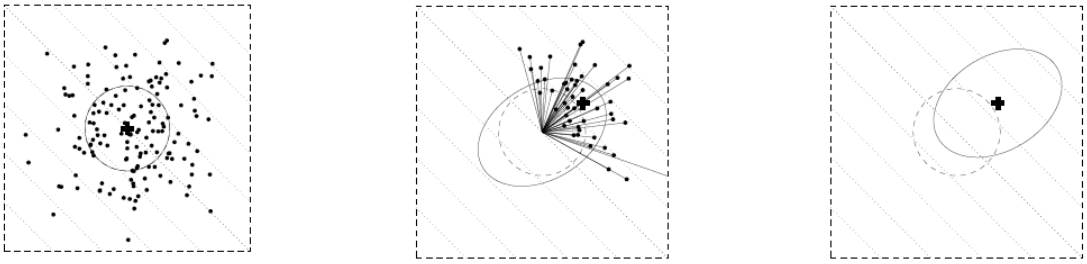

Figure 2.3 The update process of TDL-ESr in one step. Step1: generate samples via policy search. Step2: mean update via (2-6). Step3: mean-variance via (2-3).

2.2.3 TDL-direct Algorithm

We also propose TDL-direct method to directly maximize the following objective:

$$maximize_{\mu,\sigma} \sum_{a} N(a|\mu,\sigma)A^{\pi_{old}}(s_t,a)$$

With the KL constraint $KL(N(new)||N(old)) \leq \delta$. Given state sample s_t, action sample a_t and estimated advantage $\widehat{A_t}$, the target mean is set by:

$$\widehat{\mu_t} = \mu^{old}(s_t) + sign(\widehat{A_t})\, min\left(1, \frac{\sqrt{2\alpha}}{|y_t|_2}\right) y_t\sigma^{old} \quad (2\text{-}7)$$

Where $a_t = \mu^{old}(s_t) + y_t\sigma^{old}$ and α is a hyper-parameter (positive real number) related to the size of the trust region. Consider that:

$$\mu_{ti}{}^2 \leq (min\left(1, \frac{\sqrt{2\alpha}}{|y_t|_2}\right) y_t)^2 \leq 2\alpha$$

And we assume the change of variance which is updated separately thus is easy to bound between one iteration is not too much, that to say, we can assume $\sigma_{i,t+1}/\sigma_{i,t} \in [1-\epsilon, 1+\epsilon]$ where ϵ is a small value. The KL divergence between the old policy and the new one updated by (2-7) is:

$$KL[N(\mu_{old}, \sigma_{old})||N(\hat{\mu_t}, \hat{\sigma})] = \frac{1}{2}\sum_i [2log\sigma_i + \frac{1+{\mu_{ti}}^2}{{\sigma_i}^2} - 1]$$

$$\leq \frac{1}{2}\sum_i \left[2\,log(1-\epsilon) + \frac{1+2\alpha}{(1-\epsilon)^2} - 1\right] = n\alpha(1+2\epsilon) + o(\epsilon^2) \approx d\alpha$$

In conclusion, for both TDL-ES and TDL-direct methods, we efficiently satisfy the KL constraint on each sample or sample-wise. However, in TRPO or PPO, they relax the constraints from max to mean or global KL divergence. The experiment in chapter 3 demonstrates the effectiveness of our sample-wise constraint.

Chapter 3 Experiments on Mujoco

We first conduct extensive experiments on the Mujoco platform based on OpenAI gym[69] environment. In general, we want to show algorithm's capability on solving fundamental issues thus we design three aspects of experiments to demonstrate the following: 1) The performance on continuous control tasks. Based on the experiments of 13 Mujoco tasks, our algorithm is superior or equivalent to state-of-the-art benchmarks on both high cumulative reward and repeatability. 2) The data efficiency issues that our algorithm is safe for on-policy sample reuse. In other words, due to our supervised learning style, we can safely improve the training epoch on each sample without damaging the stability of the model. 3) The conservative iteration issue, our algorithm can satisfy maximum KL divergence constraint on policy space more than TRPO and PPO do, thus more efficiently guarantee policy improvement.

3.1 Performance on continuous control tasks

We compare our three algorithms: TDL-ES, TDL-ESr, TDL-direct with two famous policy-based methods: TRPO, PPO which we mentioned above, plus we further implement MPO[70], PGPE[71] and improved CACLA[72]. For MPO, it differs from TDL in that MPO uses an estimated Q function for target probability density to on each sample which is harder to train (than the state value function used in TDL) for continuous actions and MPO is hardly invariant to reward scaling. For PGPE, we implemented PGPE with the same policy network as TDL (the original PGPE used a linear policy) for a fair comparison. Originally, PGPE was proposed to alleviate the variance explosion problem when the episodic length is large. But it does not address the problem when the policy is close to deterministic. We observed that the training process of PGPE is not as stable as TDL in tasks that require precise control such as *InvertedPendulum-v2*. Considering the performance of MPO and PGPE is similar, to save the space on the figure, we don't show the results of PGPE. Plus, we implemented a batch version of CACLA with the same network architecture and it can be regarded as an ablated version of TDL-ES without setting targets. TDL-ES benefits from conservative updates by setting targets and performs better than CACLA in terms of asymptotic performance. Notice that we also implement traditional policy-gradient based methods DDPG and A2C by ourselves, the

performance on mujoco is even better than OpenAI baseline, however, they are still much worse than ours, especially in complex environments like humanoid. Because these kinds of methods are not directly related to ours and to further save the space, we won't show their results either, one can easily make comparisons from the benchmarks provided by OpenAI themselves.

All in all, the comparison is shown in Figure 3.1.

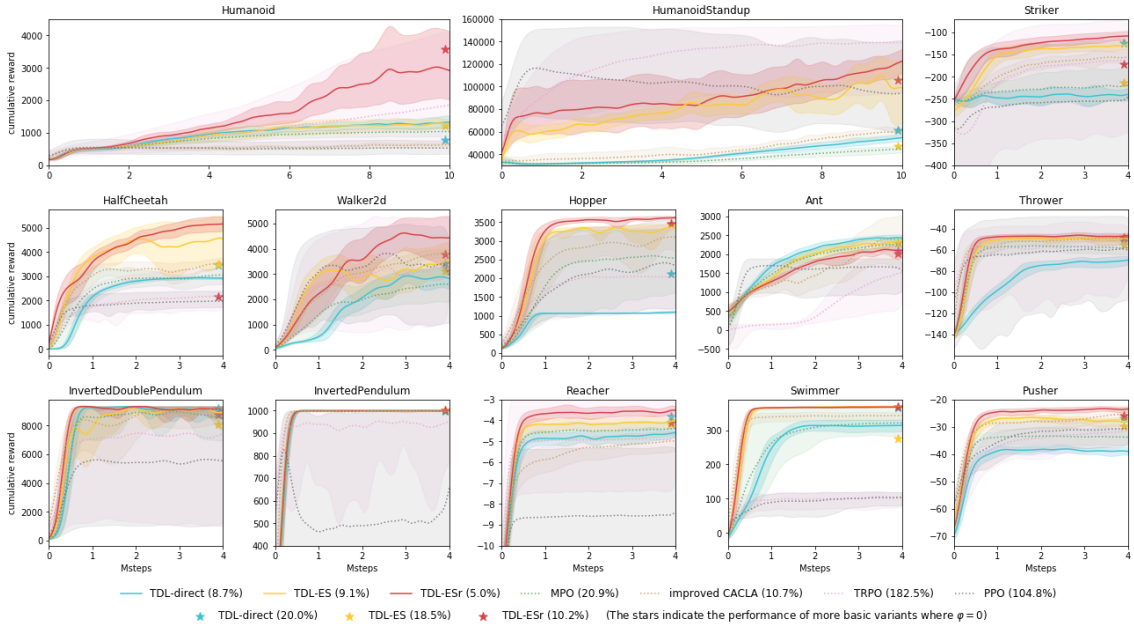

Figure 3.1 The comparison on several Mujoco tasks. The moving average of five independent runs is shown as solid line. The shade areas are range from 10% to 90% quantiles. The percentage in brackets at the end of legend is the mean fluctuation of the scores over the last 100 iterations among all tasks. Notice that the three star marks of TDL algorithms indicate more basic cases which variance update jointly by (2-2).

As we can see from figure 3.1. TDL-ESr almost gets the highest rewards in all tasks except *Ant* where TDL-direct is the best and *HumanoidStandup* where TDL-ESr is slightly lower than TRPO. Thus we can safely say our TDL method, overall, outperforming all benchmarks. In specific: 1) compared with MPO and improved CACLA, TDL-ES and TDL-ESr outperform those two methods in all tasks which demonstrate the effectiveness of the way we set learning target. 2) compared with TRPO and PPO. We found their performance varies drastically among different environments. For precisely control tasks like *Reacher* and *InvertedPendulum*, their performance are far worse than ours because we can address the instability issue mentioned in 2.1. For two

humanoid tasks where TRPO plays good results, but the learning process fluctuates which is shown by the percentage number in the legend of figure 3.1. 3) Our algorithms with the lowest fluctuation values are more stable and steadily improved along the training process.

For the comparison of three models, TDL-direct is quite stable since it strictly subjects to the constraint, but it is more prone to converge to a local optimum. TDL-ES performs better than TDL-direct when fine-tuned since it balances exploration and exploitation. TDL-ESr performs the best among the three especially in complex environments since it exploits the regularity over the transition dynamics.

The hyper-parameter in the experiments are shown in the table below:

Table 3.1 The Hyper-parameter in the experiments

H-param	Value
Policy network	3 hidden layers with 64 neurons each
Critic network	3 hidden layers with 64 neurons each
Nums. Steps in each iteration	2048
Discount rate (γ)	0.995
GAE parameter (λ)	0.97
Num. epochs	60
Minibatch size	256
Adam learning rate	1e-4
Init std of action distribution (σ_0)	0.3
2α in TDL-direct	0.05
Step size in TDL-ES and TDL-ESr	1.0 (0.05 for *Ant* and *Humanoid*, 0.5 for *HumanStandup*)
Num. adjacent points N in TDL-ESr	2 (5 for *HumanStandup*)
Revising ratio η in TDL-ESr	0.9 (0 for *HumanStandup*)

3.2 Stable for Sample Reuse

Data efficiency plays an important role in training which directly affects the training time and performance. Off-policy methods usually utilize history data called experience replay to train model to improve the sample efficiency, while on-policy methods can train more epoch on the same on-policy samples in each iteration but this may incur unstable issues.

To compare the performance against sample reuse, we test the performance of TDL

and PPO on different levels of training epoch. The results are shown in Figure 3.2. Simply compare TDL-ESr with PPO. The performance of TDL-ESr gets better with the training epoch goes up, even in extreme circumstances when training epoch is 100, it's a little bit worse than a low level, but it also guarantees a steady improvement over iterations. On the contrary, the training process corrupts drastically when the training epoch is larger than 60. That's to say, the sample reuse in PPO might not improve the performance, but made the training unstable. For another valuable minor comparison, the sample reuse in TDL-ESr leads to a big improvement in performance (the accumulative rewards of 60 epochs is around 2 times of 10 epochs) while the improvement of PPO is relatively small.

Notice that PPO updates the policy network by policy gradient along with the clipped surrogate objective, on the one hand, as we mentioned in 1.1.2 the gradient of mean and variance is mixed, we can't explicitly display the update direction of the action distribution, on the other hands, when repeatedly optimizing the surrogate objective, more samples will be clipped, i.e. masked and the action distribution on the corresponding state samples may deviate, which may lead to an unstable update as shown in Figure 1.2. Yet the model under TDL algorithms is repeatedly trained to match the proposed target distributions which are known stable thus sample reuse can be safely improved.

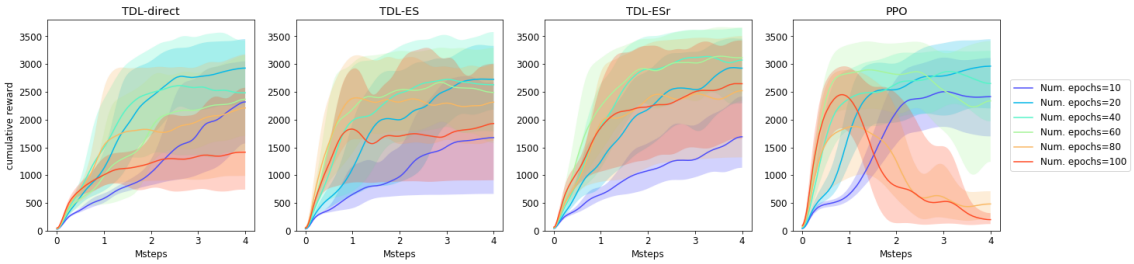

Figure 3.2 The illustration of sample reuse on different levels. The solid line indicates the moving average of five independent runs. The shaded areas are ranging from 10% to 90% quantiles. The lines in different colors show different level of sample reuse, where Num.epoch indicates the training epoch of on-policy samples in each iteration.

3.3 Constraint on KL-divergence

The algorithms who rely on CPI (conservative policy iteration), like TDL, TRPO, and PPO, requires constraints on policy update in the state-action space. One most popular used constraint is KL divergence. Notice that in TRPO, they use and in PPO. We want to test how well is the KL constraint bounded by these algorithms. We follow the intention

in TRPO to record the maximum KL divergence on all samples in each iteration and the experiment, we first sample 2048 (the number of steps) transitions in each iteration then we calculate the KL divergence on each state-action samples and finally record the max one. The result is shown in Figure 3.3.

Two observations from the experimental results: 1) The maximum KL divergence of TDL algorithms are effectively bounded by the limit we set (that is, both red and green curve are beneath the horizontal upper limit we set), while for TRPO, the maximum KL exceeds the pre-set boundary in all tasks, even 2 order higher (100 times higher) than boundary in specific tasks. 2) The maximum KL divergence of our algorithms especially for TDL-direct methods is much smaller than TRPO and PPO (with the KL bound even higher than TRPO), which demonstrates the capability of TDL in restricting KL divergence and leading a stable improvement. This property attribute to our sample-wise constraint style. Based on the observations above, we can safely conclude that TDL restricts the change of policy in state-action space more effectively thus guaranteed a more conservative policy iteration. By the way, our experiment provides more support on the finding in [73].

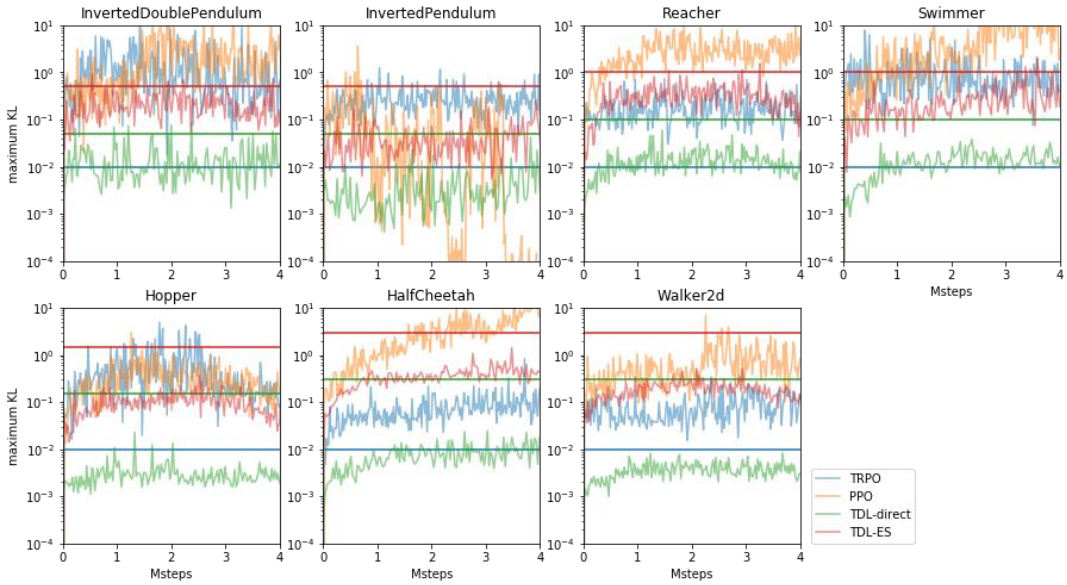

Figure 3.3 The maximum KL divergence on several tasks. The blue, green and red horizontal straight line denotes the KL constraints of TRPO, TDL-direct and TDL-ES respectively. Notice the ordinate is logarithmic.

Chapter 4 Application in Quantitative Trading

The portfolio management problem is a series of decisions to redistribute capital among securities. In other words, it is a process of dynamic adjusting weight list. The decision-makers employ certain strategies to manage their portfolio i.e. plenty of trading securities to maximize the profit.

In mathematical terms, the investor plan to trade N securities thus he needs to keep an N+1 length-weight list along the trading time t: $W_t = \left[w_{1,t}, w_{2,t}, \dots, w_{N+1,t}\right], t \in [1, T]$, and $|W_t|_1^1 = 1$. Note that an extra weight denotes the 'cash' term.

Denote $P_t = [1, P_t^2, P_t^3, \dots, P_t^{N+1}], t \in [1, T]$ be price dynamics of all securities and the first item denote the price of 'cash' which is fixed as 1.

Denote $B_t^i = P_t^i + fee + C_t^i, S_t^i = P_t^i - fee - C_t^i, t \in [1, T], i \in [2, N+1]$ as real buy and sell price respectively. Where fee, C_t^i are the consumption fee and impact cost. Where consumption fee is fixed for all securities while impact cost is affected by the market environment.

During the trading, all researchers will make an approximation from continuous trading time to become a discrete-time point with equal interval. Under this approximation, the price change only occurs at $t = n\hat{t}$, where $\hat{t}$ is the time interval and n is a positive integer. This is consistent with the actual situation, in the Chinese stock market the trading message is updated by every 3 seconds and around 0.33 second in the cryptocurrency market.

Therefore the investor will earn R_t from $t - \hat{t}$ to t with weight list W_t:

$$R_t = (S_t / B_{t-\hat{t}})^T W_t$$

The total profit during trading time is $\mathbf{R} = \sum_t R_t$.

Modern quantitative researcher employ AI algorithms to maximize long term returns $\mathbf{R}$ by 1) seek optimal order execution, i.e. lower B_t, higher S_t and 2) a powerful policy to adjust W_t. Note that for order execution problems, researchers have to make plenty of assumptions for mathematical modeling because high-frequency microstructure is quite complicated which made by multi-agent operation. Meanwhile, the performance of high-frequency trading highly depends on the time delay of data transfer and the computing power of the machine. That's the reason that equivalent performance is hard to achieve in real industrial applications. We believe policy management policy is easier to learn and

the experimental test is more convincing. The reason is for a relatively long term trade which $\hat{t}$ is usually set at a minute level will not only suffer less order execution impact and more predictable.

The prediction and trading decisions are usually separate in strategy based on the supervised learning model. Normally, researchers will first formulate a learning objective then extract features and learning targets from historical transaction data, for example, to predict the direction of price moving, market volatility, and so on. A supervised model like deep neuron network and gradient boosting trees is trained. The trade signals are further translated from prediction results under specific rules summarized from human knowledge. For instance, suppose we already have a pre-trained two-class classification model which predicts the probability of rising or drop, one common approach is setting a threshold to filter prediction signals then buy such security if the probability above the thresh and do nothing otherwise. The framework is shown in Figure 4.1.

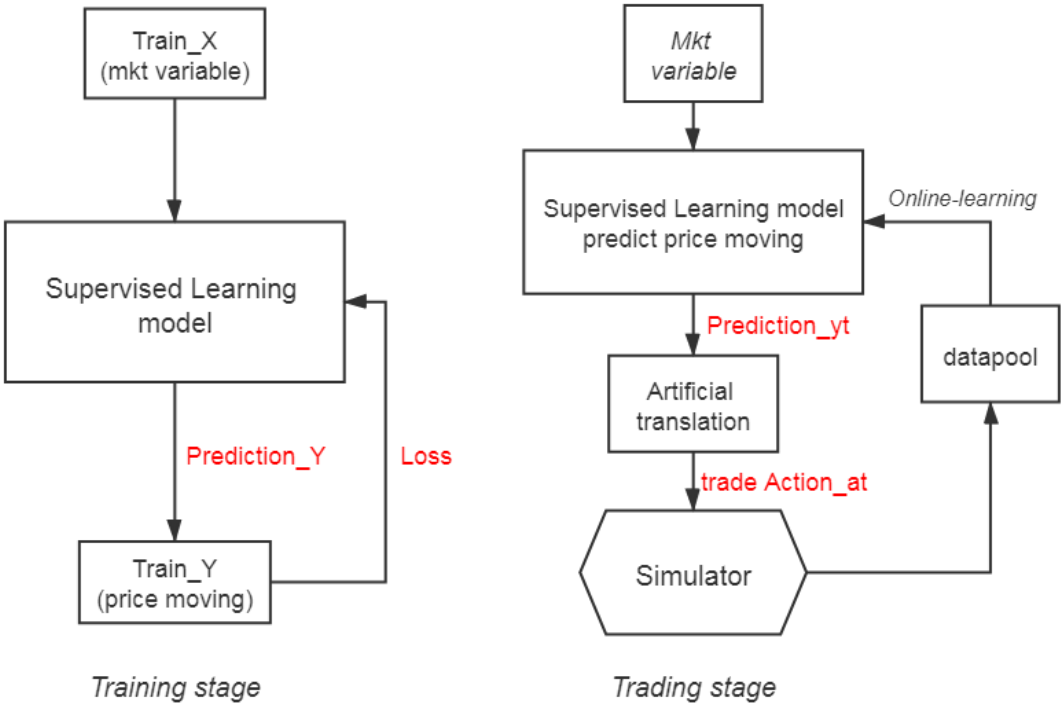

Figure 4.1 Trading by supervised learning framework. Left: training stage. Right: apply the trained model into trading.

4.1 Environment Simulator

When it comes to practical application, building a reasonable simulator like Mujoco environments are no less than an effective algorithm. The simulator is considered as mathematical modeling of actual scenario which not only provides data samples to data-driven RL agents but also incorporates an elaborate state transition model and appropriate reward function which follows the MDP process. Researchers conduct experiments on the simulator to test the performance of an RL model and the feedback further serves as guidance for algorithm modification. What's more, Simulator helps researchers find what is exactly learned by their RL agent and provides a platform for equal comparison.

In this thesis we don't follow multi-agent environments setting like[79] where plenty of investors and market makers compete against each other and the order price is precise to which bid-ask ticks. We care more about the effectiveness of weight relocating signals instead of detailed order execution, thus we construct a 'one vs market' style environment with the assumption that our decision has little effect to market side.

4.1.1 State

The state consists of three-part: Market observation, the extracted market factors, the private account and positions.

We assume a fully observable environment, and we name the first two-component as *Mkt variable* and account information as *position variable*. That is $S_t = S_{Mkt,t} \cup S_{pos,t}$.

In an order-driven market, the market observation here is the order request messages aka. Limit order books and a summary of transactions during the interval named K-line.

To be specific, there are two types of orders: Limit order (LO) and Market order (MO). For limit orders, the trader sending requests to exchange about the trading volume and at which price they intend to buy or sell, i.e. bid and ask. The orders with the same prices will be gathered in the same position on Limited order books. It is guaranteed that the deal price will no worse than the limit price, i.e. no lower than ask price and no higher than bid price. The exchange helps achieve the deal in two principles: price first and time first. That is, take ask side, for example, orders with lower price will be traded first, the higher price order can only be traded until all lower ask orders have made a deal and for limit orders with the same price, the deal is made in chronological order. The structure of the LOB is shown in Figure 4.2. For Market order, the investor only needs to tell exchange the volume they want to buy or sell, then market order will proactively 'eat' orders on the

opposite side in the LOB. The snapshot of LOB normally is sent several times per second. In our settings, LOB is updated each second.

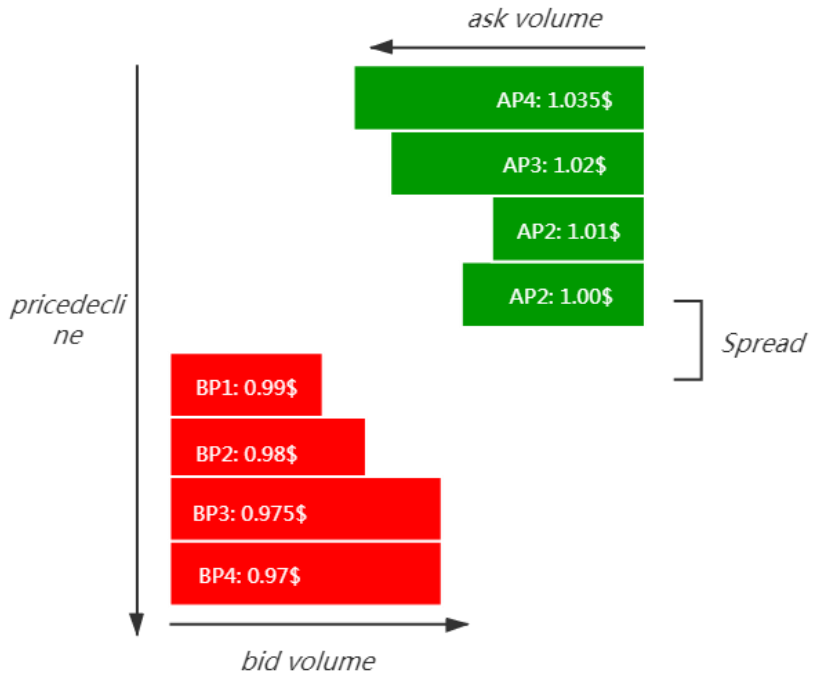

Figure 4.2 The structure of Limit order book

K-line data contains several statistic indexes calculated at the end of each time interval. The most important ones are *high, low, open, close, volume, amount* where the first two indexes denote the highest deal price, lowest deal price during the interval and *open, close* denote the price of first and last deal respectively. The total trade volumes and capital amounts during the interval are present as *volume, amount.*

Given a chosen time interval *T*, the market observation of the state S_t is the K-line indexes and snapshots of LOB during $t - T \sim t$.

The extracted market factors of S_t are calculated based on market observation, these high-order features depict market transitions in a more complex and effective way. In our settings, we employ the following factors which play a key role in supervised learning models:

The factors based on LOB snapshot: denote the price and volume at bid n level as Bid_{nprice}, Bsk_{nvol} and Ask_{nprice} for the ask side.

OFI (order flow imbalance): reflect liquidity of LOB. As we know the deal and

cancel the order will remove LOB liquidity while limit orders will add liquidity. The OFI temporal variation of such order flow at each price level. The detailed formula is shown in [89].

VOLR: it reflects the volume pressure of each side by comparing the relative size of the ask side and bid side at the first three price levels.

$$VOLR = \beta_1 \frac{Ask_{1vol}}{Bsk_{1vol}} + \beta_2 \frac{Ask_{2vol}}{Bsk_{2vol}} + \beta_3 \frac{Ask_{3vol}}{Bid_{3vol}}$$

MA(moving average) of mid-price:

$$MA_{mid,n}(t) = \frac{(Bid_{1_{price}}(\dot{t}) + Ask_{1_{price}}(\dot{t}))/2}{mean((Bid_{1_{price}}(\dot{t}) + Ask_{1_{price}}(\dot{t}))/2, \dot{t} \in [t-n, t])}$$

Factors based on K-line indexes: denote close price at time t as $Close(t)$ and time interval is δ.

EMA(Exponential Moving Average): which is the weighted average price of last *N* time spots with decay factor β:

$$EMA_N = \frac{1}{N}\sum_{k=1}^{N}\left(\frac{N-1}{N+1}\right)^k Close(t - k\delta)$$

MACD(Moving Average Convergence / Divergence)= $EMA_{N1} - EMA_{N2}$,

RSI(Relative Strength Index) which reflects the prosperity of the market in a certain period by comparing the 'strength' between seller and buyer.

Apart from well-known factors above we also created our technical factors by data mining methods[46]. In general, we employ 30 extracted market factors in Mkt variables.

For *position variables* contain: 1) current positions i.e. weight list at observation time t: $W_t = [w_{1,t}, w_{2,t}, \dots, w_{N+1,t}]$ 2) the profit P_t agent has earned and total volume V_t has been traded till now. 3) The proportion of time: t/T. The role of each position variables will be discussed in 4.2.

To sum up, the state observation provided by simulator at time t is:

$$S_t = \{[LOB(\dot{t}), \dot{t} \in [t-n, t]] \cup [Close(t), Open(t), \dots, Amount(t)] \cup [OFI, VOLR, \dots, EMA, MACD, \dots] \cup \left[W_t, V_t, P_t, \frac{t}{T}\right]\}$$

4.1.2 Action and Reward

Action:

The action taken by the simulator is the trading volume of each security, which can be converted directly from W_t. In later discussion, we fix 'cash' as the first item in W_t.Note that in our settings, investors can only send one type of order for all securities in the whole test period, that's to say whether long or short each security is pre-fixed in a single run.

When given a W_t, the action A_t is calculated by the following, which is N dimensions:

$$A_t^i = Vol_t^i = Capital * (W_t^i - W_{t-1}^i)/P_t^i$$

Where P_t^i denotes close price in the last k-line of ith item. A positive value of action indicates a 'buy' action and negative indicates 'sell'. If we set $Capital$ as current cumulative profits (also known as compound interest transaction), due to $Close$ belongs to *Mkt variable*, we could also consider $W_t - W_{t-1}$ itself as an action. For a more intuitive understanding, let's take three examples: 1) when investor bet all money on a single security, let's say the second item in position list, then the action he took should be $W_t = [0,1,0,\dots,0]$. 2) when the investor has no confidence in the current market and keeps an empty position, then the action should be $W_t = [1,0,0,\dots,0]$. 3) when the investor has equal faith in all securities, the action, in this case, should be $W_t = [1/N, 1/N, 1/N, \dots, 1/N]$.

To simulate a more realistic trading environment, we add a turnover rate $\gamma_t \propto 1/(Ask_{1vol} + Bid_{1vol})$ on each weight item with the intuition that when the volume of limit orders at first price level is sufficient, the order easy to consume, on the contrary, when the market is not active, the order hard to make a deal. Thus the weight list is trimmed by:

$$\begin{cases} W_t^i - W_{t-1}^i *= \gamma_t^i \\ W_t^0 = 1 - \sum_{i=2}^{N+1} W_t^i \end{cases}$$

Reward:

For RL application in high-frequency trading, the reward design is mainly three types: zero-profit (the agent has no pursuit of profit which is common in order execution scenario), myopic (the agent search optimal solution in one step which is similar as a greedy algorithm) and optimal (where the agent cares about global optimal). Our reward setting belongs to the second type because if and only if the action in each step is optimal, the global reward is optimal.

There are two types of reward, the first one is a timely reward which is the profit of each iteration, and the other is the evaluation of the whole trading process. In detail, given a trading action $W_t - W_{t-1}$, the timely reward is calculated by:

$$R_t = \left(\frac{P_t}{P_{t-1}}\right) W_{t-1} - |W_t - W_{t-1}| * (fee + C_t)$$

Where P_t is the close price of time t, the *fee* is fixed and C_t is the extra cost or slippage, for convenience, C_t is as a random variable sampled from a uniform distribution $C_t \sim U(0, C)$ where max cost C is a hyper-parameter.

For reward only received at *'end' state* is the bonus of larger trading volume because, in cryptocurrency, the exchange encourages more transactions by giving a lower *fee*. When the volume reaches a certain amount, there will be zero consumption fee, even negative fee (earn profit from the exchange). Thus this effect of fee discount is presented by an extra rewards at the end of trading:

$$R_{end} = R_T + \beta * V_T$$

Where β is the bonus ratio and V_T is the average position rate during the whole trading.

More Reward Engineer work is described in 4.2 and the experiment result is shown in 4.4.

4.2 RL Framework

The RL framework is shown in Figure 4.3.

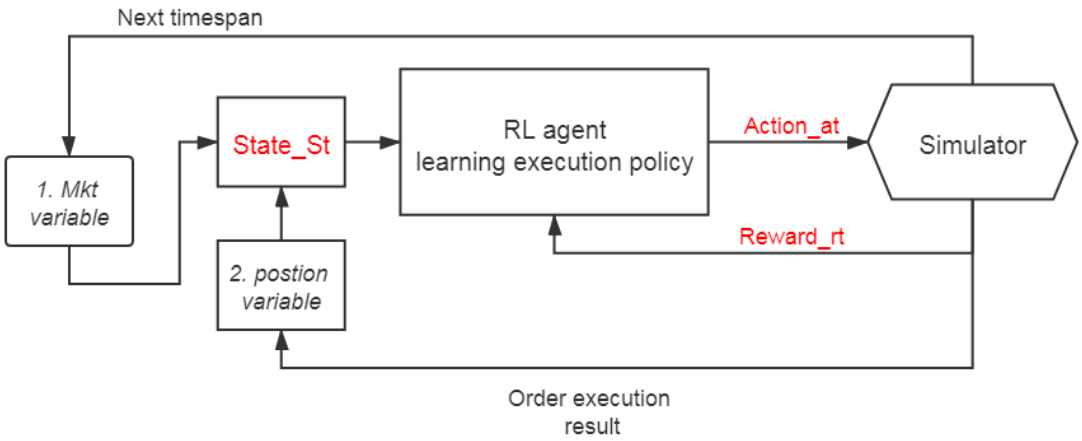

Figure 4.3 The RL trading framework. Where the RL agent is trained by modified TDL.

Modified TDL-ES

To better apply TDL in this scenario, we made some modifications to both algorithms and network structure.

The input structure i.e. state is different than Mujoco tasks, which is multi-input. We use extracted market factors only as of the first input tensor for NN, the *Mkt variable* here is a 2-D matrix with shape (N, M), the columns are M factors e.g. *MACD, RSI, OFI,* etc. and the horizontal axis denotes each N securities. The second input tensor is *Position variable* $[W_t, V_t, P_t, t/T]$ where we merge $W_t[1:]$ into the output tensor of first hidden layer and add $V_t, P_t, t/T$ as three extra unites to the input of the third layer.

For network structure, we keep Actor as a three hidden layers neural network, and the operation of each layer is modified as follows: the first two hidden layers only conduct the horizontal operation, i.e. extract more effective expressions row by row within each security factors. After this, the output of the second layer becomes normal 1-D vector, then we feed it into the third layer to mining the correlation among securities. Meanwhile, to satisfy norm restriction ($\sum_{i=1\sim N+1} W_{i,t} = 1$), a softmax activation layer is added after the previous output layer. The modified NN structure is illustrated in Figure 4.4.

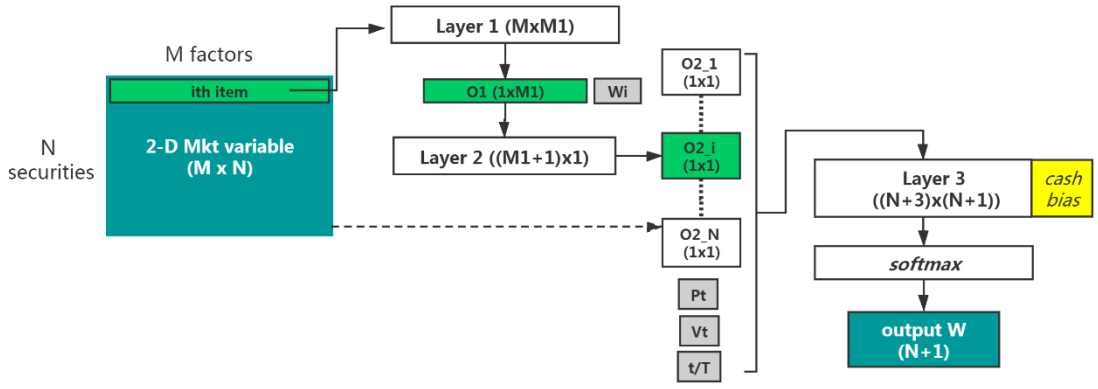

Figure 4.4 The operation process of the Actor network. The values in brackets denote input and output dimension. Note that data in the first two layers are fed row by row, and further processed vertically in the third layer. The gray square indicates second inputs. A cash bias is added in layer3 which adds its dimension to N+1.

For algorithm, firstly we change the mean target of TDL-ES proposed in 2.2.1 by adding another optimal ES offspring. Recall the proposed mean by TDL-ES:

$$\mu_{t+1}(s_t) = \mu_t(s_t) + \varepsilon \mathbb{I}\{\widehat{A_t} > 0\}(a_t - \mu_t(s_t))$$

Where a_t is offspring generated by 1,1-ES and $\widehat{A_t}$ is estimated advantage of such offspring. TDL intends to update policy towards offspring with a positive advantage. In high-frequency trading cases, the optimal action is known when the next state is given. i.e.

$$a_t{}^* = argmax_a\ earning\ rate = argmax_w\ f(P_t, P_{t+1}, w) = [0, \dots, 1, \dots 0]$$

Where we only consider price change and ignore consumption fees or costs. That is, in primitive reward design, the optimal action $a_t{}^*$ is to allocate all the money to the security with the highest yield (if all the securities prices fall, the agent should keep all money as cash). Thus, an optimal action spring is added to revise mean target:

$$\mu_{t+1}(s_t) = \mu_t(s_t) + \varepsilon_1 \mathbb{I}\{\widehat{A_t} > 0\}\big(a_t - \mu_t(s_t)\big) + \varepsilon_2 a_t{}^*$$

Where ε_2 indicates the extent of modification. Theoretically the value of ε_2 should relate to the distance between original action and optimal one to ensure a smooth update, for convenience, we choose ε_2 as 1/5 of ε_1 in experiments.

Secondly, as an inevitable problem in the quantitative trading domain, there always exists bias between train data and test data in the forms of data distribution or mapping relationship $X \to Y$. Things become even worse for a fixed state transition trajectory of RL learning process with a fixed period. The RL agent, start with an empty position and follows certain market scenarios might hard to generate unbiased experiences as well as exploration samples. To address this potential issue, we render a random initialization with a certain probability ψ. In specific, we start a run at random time point along the whole timespan and assign a random weight list to initial positions aiming to both encourage exploration and fight against bias. The experimental results show a more stable and robust policy is learned by this random-start scheme.

Reward Engineer

Reward Engineer is a kind of direction in RL research, which helps accelerate model training, improve performance, or stabilize the training process. For trading tasks, making a profit is not the only goal for inventor, risk control weight no less than profit. Some RL algorithms[33] regard a controllable risk tendency mechanism as their advantage. Normally, risk tendency includes risk- aversion, risk-neutral, and risk-seeking.

In our setting, we modify the reward function to reflect risk tendency.

We denote our primitive reward function as risk-neutral since it objectively reflects the trading profit. We propose two schemes of reward function to realize a controllable

risk tendency:

In the first scheme, we add penalty on the dispersion of positions. As a consensus in trading, investors usually reduces inventory risk by investing in different products i.e. reduce risk exposure on a single or similar items. We use standard deviation to formulate dispersion of weight list and add a penalty for excessive cases.

$$R_t = \left(\frac{P_t}{P_{t-1}}\right) W_{t-1} - |W_t - W_{t-1}| * (fee + C_t) - \eta \frac{std(W_t)}{std([1,0,\dots,0])} \quad (4\text{-}1)$$

Where the denominator $std([1,0,\dots,0])$ indicate the most extreme cases when all capital is assigned to a single security. When money is equally distributed the *std* equals 0. We call this type of reward as a dispersion control reward. When η is positive, the agent is risk- aversion or risk-seeking when η is negative.

In the second scheme, to highlight the differences among each security, we conduct nonlinear amplification for return term, thus the reward function becomes:

$$R_t = \left(\frac{P_t}{P_{t-1}}\right)^{(1+\eta)} W_{t-1} - |W_t - W_{t-1}| * (fee + C_t) \quad (4\text{-}2)$$

In this case, a positive η indicates risk-seeking policy while a negative one means risk- aversion.

Lastly, inspired by daily alpha strategy on stock market. The investor cares more about hedged returns or relative returns rather than the absolute ones. In simple calculation, a return of hedge strategy is the portfolio return minus market return. On the other hand, in sample evaluation, some RL algorithms use advantage functions other than Q value which is equivalent to using the relative reward. Hence, we modify return term from original reward function aiming to pay more attention to outperformed items:

$$R_t = \left(\frac{P_t}{P_{t-1}} - mean(\frac{P_t}{P_{t-1}})\right) W_{t-1} - |W_t - W_{t-1}| * (fee + C_t) \quad (4\text{-}3)$$

Where the $mean$ term can be seen as market return. For instance, in price drift period where the price of all securities is rising or dropping with a different degree, (4-3) helps relocate more weight on most appealing items.

4.3 Ensemble with Supervised Learning

Pure Reinforcement Learning agent might not be robust enough even fails to converge due to the low signal-to-noise ratio of trade data. Meanwhile, plenty of irregular

trading behavior caused by irrational traders and violent fluctuation brought by the emergency news will further aggravate instability issues. Thus, we further combine reinforcement learning with supervised learning models. With the guidance of prediction signals reinforcement learning policy may achieve better results.

We proposed two types of combination, the first is *Outer Link* where two methods are linked in signal level, and the other is *Inner Link* where the prediction signals degenerate into a state variable. The framework of two combinations is shown in Figure 4.5 and 4.6 respectively.

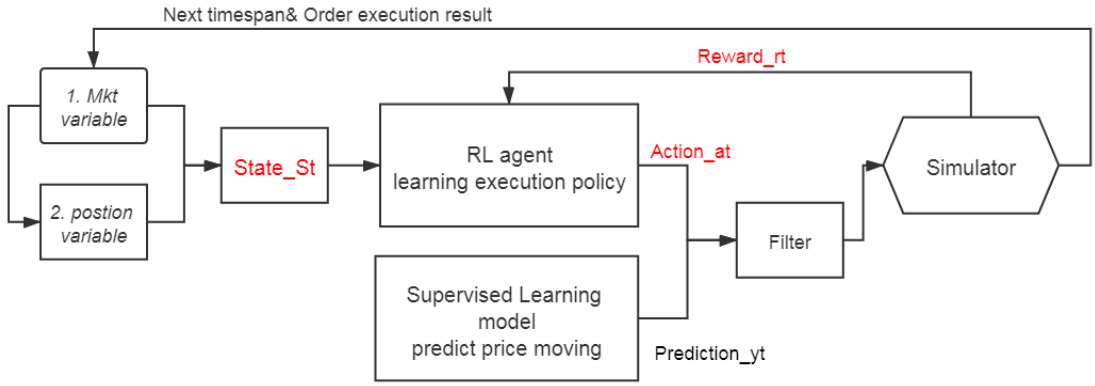

Figure 4.5 The Outer Link combination. The action signal generated by RL agent is filtered by supervised model before sending it to the simulator.

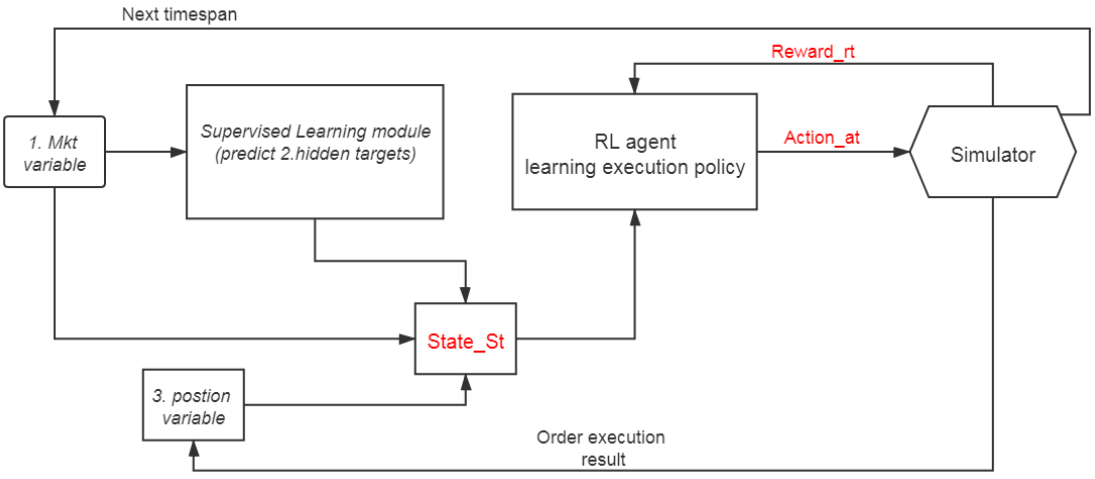

Figure 4.6 The Inner Link combination. The prediction signal generated by supervised model is added into state variable of RL agent.

For outer link, two signals are simply combined by the following rules: the securities

are labeled as ‘rise’ or ‘drop’ according to predicted probability and a selected threshold. We first set the weight of ‘drop’ securities as 0, then we: 1) reallocate residual weights to cash for risk-avoiding. 2) allocate them to securities marked as ‘rise’ in same proportion.

For inner link, we add prediction probability of each security into the output of first layer O_1, right behind the *last weight* unit (seen in Figure 4.2).

4.4 Backtest Trading

Experiment settings

This set of experiments is based on the data from Phoenix. Phoenix is a cryptocurrency exchange where traders around the world can trade between different cryptocurrencies in 24 hours a day. In this set of experiments, we use the data from eight trading pairs: *ETC/BTC, ETH/BTC, GAS/BTC, LTC/BTC, LSK/BTC, XRP/BTC, EOS/BTC* and *USDT/BTC*. For each trading pair, one sample corresponds to one market snapshot, which is captured for approximately every second. Note that we use BTC as base currency rather than RMB or dollar to get rid of the drifting impact between the cryptocurrency market and real-world currency. The training samples used in the experiments are from 90 consecutive trading days from 2018 April to August, with a total number of 60 million. The time interval T is 10 minutes means that we change the position weight of the portfolio at the end of each 10 minutes and do no operation during the interval.

We compare the following methods within the same trading period:

Supervised learning model (SL): we train a LightGBM model as a binary classification problem. The input features are extracted market factors introduced in 4.1.1. We split the first 30 days as a training set and render an online-learning scheme to update our model in weekly frequency. And test model performance in the next 60 days. We transfer prediction signals into the trading signals by first divide prediction results into ‘rise’ or ‘drop’ category according to a threshold then the cash is equally allocated to each ‘rise’ securities. The threshold is empirically set as 0.7.

TDL model (TDL): The modified TDL-ES methods described in 4.2. For detail, the number of units in each layer of policy network is 64,1,9, ε_1,ε_2 are chosen as 1 and 0.2 respectively, and we utilize primitive reward function. Corresponds to SL methods, we use the first 30 days to train RL agent and test in the next 60 days. The RL agent is retrained every weekend.

Outer Linked ensemble model (SL+TDL-outer): The outer link combination of SL and RL model. The training and testing scheme is the same as above. The reallocating approach that whether to allocate residual weight to cash or 'positive' securities are named as SL+TDL-outer-1 and SL+TDL-outer-2 respectively. The threshold of the SL model is chosen as 0.6.

Inner Linked ensemble model (SL+TDL-inner): The inner link combination of two models.

TDL model with reward design: We test two types of reward variants (4-1) and (4-2) with different risk tendencies, where we set η as -1e-4 or 1e-4 for (4-1) and 2 or -0.5 for (4-2) to realize risk-seeking and risk- aversion. The hedged reward i.e. (4-3) is also tested.

For reward function, the consumption fee is 0.001 and extra cost C_t is sampled from $U(0,0.001)$. The trading price P_T is the last price of each interval T. Note that the backtest trading period for all algorithms is 60 days thus when agents manage their portfolio every 10 minutes, there are total 8640 times turnover operations. According to the real preferential policy of consumption fee provided by the exchange, we set β in end reward function as 8640/2000=4.32, which means a 20% improvement of inventory will bring 1/10000 reduction of consumption fee.

Performance metrics

Daily rate of return (Daily PCT): We test the trading with non-compound interest, that is the trading capital in each interval is fixed as 1, the extra parts will be extracted as profit and will not be invested in the next interval. Suppose total *Pnl* in N days is Pnl_{total}, then the daily rate of return is Pnl_{total}/N.

Winning rate (WR): Suppose total times of deal is *N* and we gain profits in *n* times, so the winning rate of a given strategy is n/N. This index is equivalent to the *accuracy* of classification model.

Daily Volatility (Daily Vol): The standard deviation of daily return.

Inventory ratio (effective signal ratio): The mean ratio of inventory in each interval, i.e. $mean(\sum_t \sum_{i=2}^{N+1} W_{t,i})$. As we mentioned above, a large deal volume will bring a discount on consumption fee which is formulated in end reward. This index is equivalent to *recall* of classification tasks. Plus in other trading scenes, such as market-making and order executing, making more deals is more important than earning profits. Thus we consider the ability to provide effective signals as another aspect of ability.

4.4.1 Backtest performance

The trading metrics of each method are shown in table 4.1 below:

Table 4.1 Comparison of SL and RL variants

	Daily PCT	WR	Daily Vol	Inv ratio
SL	0.774%	**72.61%**	2.651%	39.64%
TDL	1.572%	37.29%	2.876%	92.30%
SL+TDL-outer-1	2.279%	62.36%	**1.939%**	59.18%
SL+TDL-outer-2	2.318%	40.31%	2.423%	92.30%
SL+TDL-inner	**2.593%**	37.37%	2.296%	**94.16%**

Total	**SL**	**TDL**	**SL+TDL-outer-1**	**SL+TDL-outer-2**	**SL+TDL-inner**
Pnl	3.501	6.547	6.306	8.319	**8.967**

In table 4.1, the above table shows each metric without volume bonus in end reward and the bottom table is the sum up of all Pnl.

We observe several phenomena according to this experiment: 1) According to Daily PCT, the comparative relationship is: Supervise learning-based method < Reinforcement learning method < combined SL and RL methods. And the inner combination is optimal. As we can see, even though the WR i.e. accuracy of supervised learning method is high, the traditional 'cut off' by thresh strategy leads to a poor performance where the ratio of effective signal is only 40%. 2) It is a little bit strange that the winning rate of RL agent is very low which is around 1/3, note that the WR of a random guess is 1/2, however, RL agent makes a lot of profits. We believe this is the unique policy learned by RL agent that it is more willing to earn a large profit at the risk of suffering a weak loss. In opposite, the accuracy of the supervised learning model is relatively high where for SL method with 0.7 thresholds, the WR is 70%+ and for the outer linked combination with 0.6 thresholds where the WR is determined by prediction signal is 60%+. 3) Compared with TDL with outer line methods, equipped with hard 'cut off' by supervised signal, outer line methods have a relatively higher Winning rate and lower volatility which obtain a better performance than pure RL methods. With the same Daily PCT, the first outer link method, when compared with the second approach, has lower total PNL because of the low proportion of inventory. 4) Compare the inner link with the second outer method, not only the Daily PCT of the inner link is higher but it also larger deal volume which further

brings more reward in total Pnl, In conclusion, by merge decision of supervised learning model into RL agent which turn hard ‘cut off’ to soft guidance leads to better performance.

To conduct a more intuitive comparison, we also plot the backtest trading curve and daily PCT of each method in Figure 4.7 and Figure 4.8.

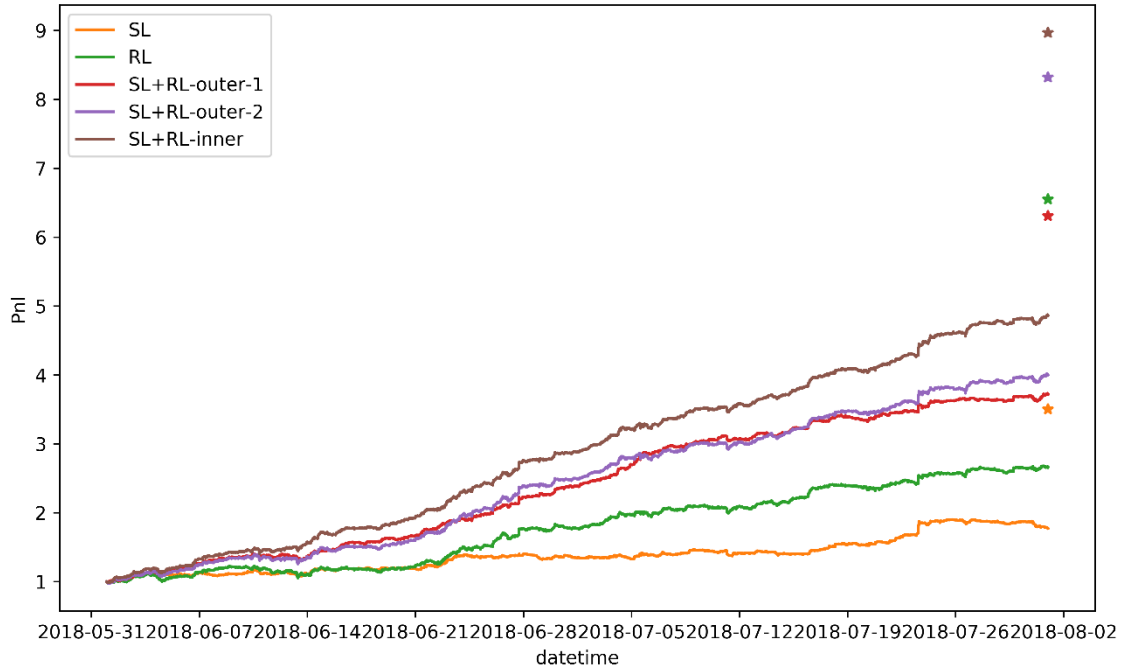

Figure 4.7 Backtest trading curve. The solid line is the cumulative reward during the test period and scatters in star shape at the end is total Pnl with volume bonus of each method.

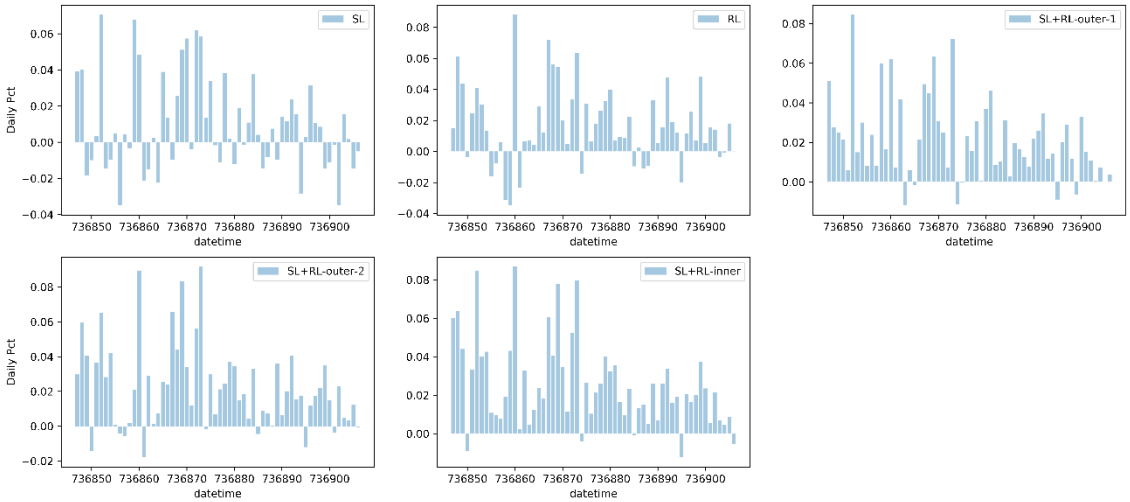

Figure 4.8 The illustration of Daily PCT. The blue bars indicate the cumulative profit earned in each single day.

4.4.2 Risk tendency with a different reward function

We also compare the performance of different reward function with controllable risk tendency, the result is shown in the table below:

Table 4.2 Comparison of reward variants

	Daily PCT	WR	Daily Vol	Inv ratio
risk- neutral	1.572%	37.29%	2.876%	92.30%
(4,1)risk-seeking	1.623%	38.24%	2.932%	92.54%
(4,1)risk- aversion	1.396%	40.62%	**2.326%**	90.63%
(4,2)risk-seeking	**1.742%**	40.36%	2.790%	93.15%
(4,2)risk- aversion	1.283%	**41.16%**	2.427%	91.32%
(4,3)hedge	0.336%	33.23%	2.675%	77.91%

The conclusions are summarized as follows: 1) the risk tendency leads no obvious improvement, but to some extent, the risk-seeking agent obtains higher profit as well as higher volatility, this conclusion in coordinate with [79] which also mentioned risk-seeking policy incur a better performance. 2) The hedged reward is not effective as expected, we guess the reason is that in formal stock market, the rank relationship among whole stocks is very regular meanwhile the market change is related to each stock which makes the rank sequence and hedged profit of each stock predictable. While in the cryptocurrency market, we can't consider the mean of the portfolio as market index and our hedging operation might break the link from market features to trading policy.

Chapter 5 Summary

In this thesis, we first proposed a new reinforcement learning algorithms for continuous control tasks named Target Distribution Learning in chapter 2, which updates the policy network in a supervised style by alternating from proposing a target distribution and update policy to match that target. We follow the classic Actor-Critic framework where the actor is modeled by Gaussian distribution. TDL can update the mean and variance separately where we employ the self-adaption technique to generate variance target and we propose three methods to set the mean target base on the ES algorithm, which is named TDL-direct, TDL-ES and TDL-ESr. Plus, we point out two instability issues of previous policy-based methods like PPO and A2C: the mixed update and gradient explosion issue when policy near deterministic and we design toy examples to illustrate these issues in chapter 1 and chapter 2 respectively.

In chapter 3, we conduct extensive experiments on Mujoco platform based on OpenAI gym environment. We try to demonstrate the performance of TDL in three aspects: 1) in 3.1, we compare the performance i.e. cumulative rewards curve on 13 control tasks. The result indicates that TDL is superior or equivalent to state-of-the-art benchmarks and at least one of three methods is optimal in every task. 2) In 3.2, we prove that our algorithm guaranteed a more efficient sample reuse. In other words, due to our supervised style of update, we can safely improve the training epoch on each sample without damaging the stability of the model. 3) In 3.3, we show that our sample-wise restriction can satisfy maximum KL divergence constraint in policy space more successfully than TRPO and PPO do, thus more efficiently guarantee a policy improvement. Plus our algorithm enjoys better repeatability and more robust to hyper-parameter.

We further apply TDL algorithm into trading domain. We start with introducing the development history of quantitative trading methods, then we point out predictability issue in the quantitative scenario and summarize two main advantage brought by AI-driven algorithms that is feature engineer and model representation in chapter 1.2. Then we further explain the potential advantage of RL framework, to some extent, reinforcement learning is the combination of intelligent prediction and intelligent strategy.

In chapter 4, we construct a comprehensive RL and SL based trading framework which includes simulator building, factor processing, and signal combination. We conduct the backtest trading in the cryptocurrency market and compare several variant methods in terms of daily return, winning rate, daily volatility, and deal volume. According to the result, our supervised model with high prediction accuracy fails to lead a good performance, and the inner link combination of SL and RL method is optimal in both high profit and large inventory ratio. Meanwhile, we design three forms of reward function to control risk tendency, the result shows that risk-seeking brings more profit. The excellent results prove the application potential of the algorithm.

For future work, on the one hand, our TDL method can be modified in off-policy setting, meanwhile, the method can be readily extended by other types of action distribution and other types of constraints in state-action space. On the other hand, for application in high-frequency trading, our portfolio management framework can further combined with the order execution system which can also be modeled by TDL.

References

[1] 陈孝良. 小样本的类人概念学习与大数据的深度强化学习[J]. 科技导报, 2016, 34(7):82-84.

[2] 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.

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

[4] Gu S , Holly E , Lillicrap T , et al. Deep Reinforcement Learning for Robotic Manipulation[J]. 2016.

[5] Hwangbo J , Lee J , Dosovitskiy A , et al. Learning agile and dynamic motor skills for legged robots[J]. Science Robotics, 2019, 4(26).

[6] Oh J , Guo X , Lee H , et al. Action-Conditional Video Prediction using Deep Networks in Atari Games[J]. 2015.

[7] Caicedo J C , Lazebnik S . Active Object Localization with Deep Reinforcement Learning[J]. 2015.

[8] Nanduri V , Das T K . A Reinforcement Learning Model to Assess Market Power Under Auction-Based Energy Pricing[J]. IEEE Transactions on Power Systems, 2007, 22(1):85-95.

[9] Cai Q , Filos-Ratsikas A , Tang P , et al. Reinforcement Mechanism Design for e-commerce[J]. 2017.

[10] Zhao J , Qiu G , Guan Z , et al. Deep Reinforcement Learning for Sponsored Search Real-time Bidding[J]. 2018.

[11] Kaur P , Goyal M , Lu J . A Proficient and Dynamic Bidding Agent for Online Auctions[C]// International Workshop on Agents & Data Mining Interaction. Springer Berlin Heidelberg, 2013.

[12] Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540):529-533.

[13] 刘全, 翟建伟, 章宗长,等. 深度强化学习综述[J]. 计算机学报, 2018(1):1-27.

[14] Rajeswaran A , Lowrey K , Todorov E , et al. Towards Generalization and Simplicity in Continuous Control[J]. 2017.

[15] Salimans T , Ho J , Chen X , et al. Evolution Strategies as a Scalable Alternative to Reinforcement Learning[J]. 2017.

[16] Mania H , Guy A , Recht B . Simple random search provides a competitive approach to reinforcement learning[J]. 2018.

[17] Ng A Y, Harada D, Russell S. Policy invariance under reward transformations: Theory and application to reward shaping[C]//ICML. 1999, 99: 278-287.Authors A . LARGE-SCALE STUDY OF CURIOSITY-DRIVEN LEARNING[J]. 2018.

[18] Pathak D , Agrawal P , Efros A A , et al. Curiosity-driven Exploration by Self-supervised Prediction[J]. 2017.

[19] Houthooft R , Chen X , Duan Y , et al. VIME: Variational Information Maximizing Exploration[J]. 2016.

[20] Mohamed S , Rezende D J . Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning[J]. Computer Science, 2015, 7(20):20-20.

[21] Williams R J . Simple statistical gradient-following algorithms for connectionist reinforcement learning[J]. Machine Learning, 1992, 8(3-4):229-256.

[22] Sutton, Richard S, Barto, et al. Introduction to Reinforcement Learning[J]. Machine Learning, 2005, 16(1):285-286.

[23] Degris T , Pilarski P M , Sutton R S . Model-Free reinforcement learning with continuous action in practice[C]// 2012 American Control Conference (ACC). IEEE, 2012.

[24] Lillicrap T P , Hunt J J , Pritzel A , et al. Continuous control with deep reinforcement learning[J]. Computer ence, 2015, 8(6):A187.

[25] Mnih V , Badia, Adrià Puigdomènech, Mirza M , et al. Asynchronous Methods for Deep Reinforcement Learning[J]. 2016.

[26] Kakade S M , Langford J . Approximately Optimal Approximate Reinforcement Learning[C]// Nineteenth International Conference on Machine Learning. Morgan Kaufmann Publishers Inc. 2002.

[27] Schulman J, Levine S, Moritz P, et al. Trust Region Policy Optimization[J]. Computer Science, 2015:1889-1897.

[28] Schulman J, Wolski F, Dhariwal P, et al. Proximal Policy Optimization Algorithms[J]. 2017.

[29] Haarnoja T , Tang H , Abbeel P , et al. Reinforcement Learning with Deep Energy-Based Policies[J]. 2017.

[30] Haarnoja T , Zhou A , Abbeel P , et al. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor[J]. 2018.

[31] Hämäläinen, Perttu, Babadi A , Ma X , et al. PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation[J]. 2018.

[32] Wang Z , Schaul T , Hessel M , et al. Dueling Network Architectures for Deep Reinforcement Learning[J]. 2015.

[33] Dabney W , Ostrovski G , Silver D , et al. Implicit Quantile Networks for Distributional Reinforcement Learning[J]. 2018.

[34] Wang Z, Bapst V, Heess N, et al. Sample Efficient Actor-Critic with Experience Replay[J]. 2016.

[35] Sharpe, William F . CAPITAL ASSET PRICES: A THEORY OF MARKET EQUILIBRIUM UNDER CONDITIONS OF RISK*[J]. The Journal of Finance, 1964, 19(3):425-442.

[36] Stephen, A, Ross. The arbitrage theory of capital asset pricing[J]. Journal of Economic Theory, 1976.

[37] Galai D , Masulis R W . The option pricing model and the risk factor of stock[J]. Journal of Financial Economics, 1976, 3(1):53-81.

[38] Fama E F , French K R . Common risk factors in the returns on stocks and bonds[J]. Journal of Financial Economics, 1993, 33(1):3-56.

[39] Bondt W F M D , Thaler R . Does the Stock Market Overreact?[J]. The Journal of Finance, 1985, 40(3):793-805.

[40] Jegadeesh, Narasimhan, Titman, Sheridan. Momentum[J]. Social Science Electronic Publishing.

[41] Hou, Kewei, Xue, Chen, Zhang, Lu. Replicating Anomalies[J]. Social Science Electronic Publishing.

[42] Jacobs, Heiko. What explains the dynamics of 100 anomalies?[J]. Journal of Banking & Finance, 2015, 57:65-85.

[43] Bloomfield R , O"Hara M , Saar G . How Noise Trading Affects Markets: An Experimental Analysis[J]. Review of Financial Studies, 2009, 22(6):2275-2302.

[44] Sanjoy, Basu. The relationship between earnings' yield, market value and return for NYSE common stocks: Further evidence[J]. Journal of Financial Economics, 1983.

[45] Chan L K C , Hamao Y , Lakonishok J . Fundamentals and Stock Returns in Japan[J]. Journal of Finance, 1991, 46.

[46] Tianping Zhang, Yuanqi Li, Yifei Jin, and Jian Li. 2020. AutoAlpha: an efficient hierarchical evolutionary algorithm for mining alpha factors in quantitative investment. Unpublished

[47] Burda, Yuri, Grosse, Roger, Salakhutdinov, Ruslan. Importance Weighted Autoencoders[J]. Computer Science, 2015.

[48] Yuanqi Li, Chuheng Zhang, Yifei Jin, Jian Li, Pingzhong Tang, 2020. AutoEnsemble: An Ensemble Model via Self-Paced Sample Reweighting and Feature Selection for Financial Market Prediction. Unpublished

[49] Andrés Arévalo, Jaime Niño, German Hernández, et al. High-Frequency Trading Strategy Based on Deep Neural Networks[C]// International Conference on Intelligent Computing. Springer International Publishing, 2016.

[50] Khaidem L , Saha S , Dey S R . Predicting the direction of stock market prices using random forest[J]. 2016.

[51] 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):1-12.

[52] Fischer T , Krauss C . Deep learning with long short-term memory networks for financial market predictions[J]. European Journal of Operational Research, 2017:S0377221717310652.

[53] J. WU, C. WANG, L. XIONG and H. SUN, "Quantitative Trading on Stock Market Based on Deep Reinforcement Learning," *2019 International Joint Conference on Neural Networks (IJCNN)*, Budapest, Hungary, 2019, pp. 1-8.

[54] Ochotorena C N , Yap C A , Dadios E , et al. Robust stock trading using fuzzy decision trees[C]// Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012 IEEE Conference on. IEEE, 2012.

[55] Nelson D M Q , Pereira A C M , Oliveira R A D . Stock market's price movement prediction with LSTM neural networks[C]// 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017.

[56] Zhang L, Aggarwal C, Qi G J. Stock price prediction via discovering multi-frequency trading patterns[C]//Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. 2017: 2141-2149.

[57] Jiang Z, Xu D, Liang J. A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem[J]. Papers, 2017.

[58] Tomas Mikolov and Kai Chen and Greg Corrado and Jeffrey Dean Efficient. 2013. Estimation of Word Representations in Vector Space[J]. arXiv preprint arXiv:1301.3781

[59] Devlin J , Chang M W , Lee K , et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[J]. 2018.

[60] Bèack T. Evolutionary Algorithms in Theory and Practice : Evolution Strategies, Evolutionary Programming, Genetic Algorithms / T. Bäck.[M]. 1998.

[61] Liu G, Zhao L, Yang F, et al. Trust region evolution strategies[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2019, 33: 4352-4359.

[62] Salimans T , Ho J , Chen X , et al. Evolution Strategies as a Scalable Alternative to Reinforcement Learning[J]. 2017.

[63] Mania H , Guy A , Recht B . Simple random search provides a competitive approach to reinforcement learning[J]. 2018.

[64] Zhao T , Hachiya H , Niu G , et al. Analysis and improvement of policy gradient estimation[J]. Neural Networks, 2012, 26(none):118-129.

[65] Mandy Grüttner, Sehnke F , Schaul T , et al. Multi-Dimensional Deep Memory Atari-Go Players for Parameter Exploring Policy Gradients[C]// Artificial Neural Networks-icann, International Conference, Thessaloniki, Greece, September, Part II. Springer-Verlag, 2010.

[66] Schulman J , Moritz P , Levine S , et al. High-Dimensional Continuous Control Using Generalized Advantage Estimation[J]. Computer ence, 2015.

[67] Hansen, Nikolaus, Ostermeier, Andreas. Completely Derandomized Self-Adaptation in Evolution Strategies[J]. Evolutionary Computation, 9(2):159-195.

[68] Hansen N . Invariance, Self-Adaptation and Correlated Mutations in Evolution Strategies[M]// Parallel Problem Solving from Nature PPSN VI. Springer Berlin Heidelberg, 2000.

[69] Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman,J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym[J]. arXiv preprintarXiv:1606.01540.

[70] Abdolmaleki A , Springenberg J T , Degrave J , et al. Relative Entropy Regularized Policy Iteration[J]. 2018.

[71] Exploring parameter space in reinforcement learning[J]. Paladyn, 2010, 1(1):14-24.

[72] Hasselt H V , Wiering M A . Reinforcement Learning in Continuous Action Spaces[C]// Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on. IEEE, 2007.

[73] Engstrom L, Ilyas A, Santurkar S, et al. Implementation Matters in Deep RL: A Case Study on PPO and TRPO[C]//International Conference on Learning Representations. 2019.

[74] Guant O , Manziuk I . Deep Reinforcement Learning for Market Making in Corporate Bonds: Beating the Curse of Dimensionality[J]. Applied Mathematical Finance, 2019, 26.

[75] Lim Y S, Gorse D. Reinforcement Learning for High-Frequency Market Making[C]//ESANN. 2018.

[76] Thomas Spooner and John Fearnley and Rahul Savani and Andreas Koukorinis. 2018. Applications of Reinforcement Learning in Automated Market-Making[J]. arXiv preprintarXiv:1804.04216

[77] Sadighian J . Deep Reinforcement Learning in Cryptocurrency Market Making[J]. 2019.

[78] Sumitra Ganesh and Nelson Vadori and Mengda Xu and Hua Zheng and Prashant Reddy and Manuela Veloso. 2019.

[79] Reinforcement Learning for Market Making in a Multi-agent Dealer Market[J]. arXiv preprintarXiv:1911.05892

[80] Das S , Magdon-Ismail M . Adapting to a Market Shock: Optimal Sequential Market-Making[C]// Conference on Advances in Neural Information Processing Systems. Curran Associates Inc. 2008.

[81] Y. Li, W. Zheng and Z. Zheng, "Deep Robust Reinforcement Learning for Practical Algorithmic Trading," in *IEEE Access*, vol. 7, pp. 108014-108022, 2019.

[82] Huang C Y. Financial trading as a game: A deep reinforcement learning approach[J]. arXiv preprint arXiv:1807.02787, 2018.

[83] Kearns M, Nevmyvaka Y. Machine learning for market microstructure and high frequency trading[J]. High Frequency Trading: New Realities for Traders, Markets, and Regulators, 2013.

[84] 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.

[85] Said E , Ayed A B H , Husson A , et al. Market Impact: A systematic study of limit orders[J]. Working Papers, 2018.

[86] Peng Wu and Marcello Rambaldi and Jean-Fran ̧ois Muzy and Emmanuel Bacry. 2019. Queue-reactive hawkes models for the order flow[J]. arXiv preprintarXiv:1901.08938.

[87] Moody J , Saffell M . Learning to trade via direct reinforcement[J]. IEEE Transactions on Neural Networks, 2001, 12(4):875-889.

[88] John Moody, Lizhong Wu, Yuansong Liao,等. Performance functions and reinforcement learning for trading systems and portfolios[J]. Journal of Forecasting, 1998, 17(5-6):441-470.

[89] Rama Cont, Arseniy Kukanov. Optimal order placement in limit order markets[J]. Social Science Electronic Publishing, 2014.

致　谢

首先要衷心感谢我的导师唐平中老师，是他带领我走进人工智能精彩世界的大门，唐老师非常尊重个人兴趣，因材施教地为我指明阶段性的方向。仍然记得刚入学时，唐老师就教导我要成为即能沉下心深入科研又能的做出实际应用成果的综合性人才，这样才具有核心竞争力，三年间，我一直为此努力。入学以来，在唐老师的带领下，我在量化投资、计算机视觉、强化学习等方面均有涉猎，尤其对基于AI 的量化策略研究有浓厚兴趣。除了科研方面的指导，唐老师也在更广的人生观层面上分享了很多经验，都使我受益终身。我还要特别感谢李建老师，李老师在这三年里给与我非常多的指导，我几乎可以算半个李老师的学生。李老师踏实、认真和切实的工作态度是我学习的榜样。这三年间，无论是机器学习的理论还是应用，尤其在很多细节之处，李老师都悉心指导，有求必应，令我获益匪浅。

感谢清华大学为我提供这么好的学习、生活环境，感谢交叉信息研究院对我的培养，让我进步和成长。在学校的三年转眼就要过去，快毕业时，开始可惜于那些没去过的角落，可惜在图书馆、教室呆的时间不够长，可惜有些食堂的窗口还没去尝试等等，我会永远铭记在清华的这段时间。

感谢实验室的学长学姐学弟学妹们，王子贺、左淞、沈蔚然、蔡庆芃、曾驭龙、刘艺成、方雯逸、肖慎柯、许娅伦、康宇衡、李君诚、张佑嘉等，感谢李建组的学长学弟比如金逸飞、曹玮、郭建波、张天平、徐进、曾梁等等，尤其特别感谢在论文上密切合作的张楚珩。他们在无论学术探讨还是生活闲谈上都给我很大帮助。

感谢 fantastic four 成员：汪勋、徐亚东、陈梦静，感谢小伙伴施宇，他们充斥着我的生活，让我觉得安心。

感谢在交叉科技共事的每个同事，以及实习公司的领导和小伙伴，感谢你们的付出和指导。另外要感谢紫荆二楼铁板炒窗口师傅和丁香园麻辣烫窗口师傅，以及FIT 楼的顺丰小哥和韵达夫妇，感谢他们给我生活带来的便利。

感谢我的家人，感谢父母的养育和奉献，感谢我女朋友董榛子，三年异国期间的陪伴和包容，让我多了一份责任和憧憬，他们是我完成学业坚实的基础。

最后，衷心感谢并祝福每个帮助过我的人。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____日　期：_____

附录 A　中文摘要

A.1　引言

A.1.1　强化学习

深度强化学习算法在近些年快速发展，主要得益于强大的深度神经网络表达能力、智能决策能力和可获得的大数据量，其在复杂的控制任务上取得了优异的表现：比如来自 DeepMind 的 Alpha GO[2]和 Alpha Zero[3]，在围棋游戏和电脑游戏击败了职业选手；比如解决机器人等领域自控问题[4]，解决计算机视觉任务[6]，以及在智能汽车和智能家居系统中也起着关键作用，还有一些研究人员将 RL 与机构设计[9]结合起来，应用于拍卖[8]和定价系统[10]。相信 RL 将在人类生活的方方面面发挥更大的作用。

强化学习的研究可以根据不同的分类方法分为不同的类别。根据更新方法，强化学习包括基于值的方法，如 DQN[12]，它根据动作估计值选择一个动作，这些方法在离散动作空间有良好的性能。基于策略梯度的方法则利用深层神经网络对策略进行建模，利用优化器根据损失梯度对网络进行更新。它可以进一步分为确定性方法和不确定性方法。这些方法在具有连续动作空间的复杂问题上表现良好。另一类方法是基于搜索和监督的，例如我们前面提到的 AlphaGo[2]。这些方法旨在利用额外的监督信息和策略搜索技术来提高学习效率[13]。

在优化目的方面，可以分解为表示、优化方法和任务设计（建模）[14]。例如，最近的一些工作表明，简单的表示，如线性[15]或 RBF 表示[14]也适用于 Mujoco 任务，一些工作侧重于奖励设计或发明不同种类的内在奖励[17]。从而使任务更容易学习或是使模型获得更好的探索能力。ES[15]和 ARS[16]使用策略搜索，但它们的工作仅适用于参数很少的表示（例如线性策略）。

大多数工作都集中在优化方法上。对于连续控制问题，最常用的是基于策略梯度的优化方法。REINFORCE[21]是最简单的策略梯度算法，其中 Q 值是通过蒙特卡罗抽样估计的。基线被用来减少方差[23]。DDPG[24]采用确定性策略梯度法，结合目标网络和噪声探索技术，解决了连续控制任务，保证了稳定性。A2C[25]是在策略网络更新中使用近似值函数的行动者-评价者方法。其大大减少了方差导致更快的学习，但也带来了不稳定性。保守策略更新[26]是为了保证策略改进的稳定性，比如 TRPO[27]和 PPO[28]等算法。尽管 SQL[29]和 SAC[30]引入了不同的优化方法，但是这项工作并没有与它们进行比较，因为它们是在引入不同策略表示的最大

熵框架下工作的。CMA-PPO[31]在 PPO 中引入了 ES 的思想，但本质上仍是一种策略梯度算法。

我们这里的工作属于优化方法，具体来说我们的工作主要是针对经典的 Actor-Critic 框架，基于确定的目标分布进行优化。在接下来的章节中，我们将以 A-C 框架为基础，首先介绍连续控制算法强化学习的基本概念和定义，然后讨论这方面的相关工作。

A.1.1.1　连续动作控制的强化学习

令s_t, a_t为当前状态和策略所采取的动作，r_t, s_{t+1}为观察到的奖励和下一个状态。给定一个轨迹$(s_0, a_0, r_0, s_1, \dots, s_k, a_k, r_k, s_{k+1} \dots)$，RL 算法的优化目标是最大化累积收益：

$$J(s, a, \theta) = E[\sum_{t=0}^{\infty} \sum_{s'} p(s_t = s' | s_0 = s, a_0 = a, \pi_\theta) \gamma^t r_t]$$

其中 π_θ 为行动策略 $\pi_\theta(s, a) = P(a|s, \theta)$，参数 θ 输出给定状态 s 下候选行动的概率，$J(s, a, \theta)$ 为累积收益，γ 为衰减率。

Actor-Critic 框架是目前最流行、最有效的 RL 框架，它结合了基于值网络的优点和策略梯度所使用的动作概率分布。在与环境交互的过程中，行动者通过策略选择动作，评价者评估每个动作的价值，两个网络协同更新，其中价值函数由真实值和估计值之间的 TD 误差更新，策略网络根据计算出的梯度（包括方向和每个动作的价值）更新。工作流程如图 1.1 所示。

对于评价网络，学习值函数来估计状态 V^π 或状态-动作对的累积收益：

$$V^\pi(s) := \mathbb{E}_{s_0 = s} \sum_{t=0} \gamma^t r_{t+1}$$

$$Q^\pi(s, a) := \mathbb{E}_{s_0 = s, a_0 = a} \sum_{t=0} \gamma^t r_{t+1}$$

这两个价值方程是相关联的:

$$Q^\pi(s, a) := \mathbb{E}_{s' \sim p(s'|s, a)} [r(s') + \gamma V^\pi(s')]$$

对于基于值的方法，如 DQN[12]、Dueling-DQN[32]、IQN[33]，它们只维护评价网络，而保留一个总是选择估计值最大的动作的确定性策略，因此这类算法求解最优价值函数实际上是为了找到最优策略。

评价者网络通过标准的监督学习方式进行更新，学习目标是：

$$y_t = r_t + \gamma Q_{\theta_q}(s_{t+1}, \pi_\theta(s_{t+1}))$$

评价者的评价网络根据最小化 TD-误差来进行更新:

$$minimize\ (y_t - Q_{\theta_q}(s_t, a_t))^2$$

另一方面。对于行动者网络（也称为策略网络）的更新，主要有两种方法。一种是基于策略的梯度，如 DDQG[24]、A2C[25]、ACER[34]。用 π_θ 表示策略，Q_{θ_q} 表示值函数。当给定一批经验数据时，梯度的计算方法如下：

$$\nabla_\theta \eta(\pi_\theta) = E_{s\sim\rho_\pi, a\sim\pi}[\nabla_\theta log\pi_\theta(a|s) Q_{\theta_q}(s,a)]$$

其中:

$$\rho_\pi(s) = E_{s_0}[\sum_{t=0}^{\infty} \gamma^t p(s_t = s|s_0, \pi)]$$

一些方法通过减去状态的价值，用优势函数$A_t(\mathrm{s,a}) = Q_{\theta_q}(s,a) - \mathrm{V(s)}$代替值函数$Q_{\theta_q}$。直观地说，导数项决定了参数空间的更新方向，值项赋予选定方向权重。

策略 π_θ 对参数 θ 的导数表示更新的方向，而值函数赋予该方向权重。通过运行 BP 算法，θ更新为：

$$\theta_{t+1} = \theta_t + \lambda\Delta\theta = \theta_t + \lambda\nabla_\theta\eta(\pi_\theta)$$

然而，步长 λ 通常较难选定，因为对于较大的 λ，算法可能会出现震荡波动，无法收敛，但较小的步长将导致太多的时间消耗。另外，在训练过程中，当网络趋于稳定时，固定步长是不合适的，有的方法采用衰减的步长或自适应的步长。然而，无论是人工设计还是根据规则生成，一个最优的步长是难以确定的。

另一个基于策略的方法将策略更新限制在信任区域内。在文献[26]中提出的保守策略迭代理论的支持下，如果保证在单次更新在策略-动作空间的变化不太大，则可以保证新策略优于旧策略。即:

$$\eta(\pi_{new}) - \eta(\pi_{old}) \le L_{\pi_{old}}(\pi_{new}) + k * \alpha^2$$
$$Where\ L_{\pi_{old}}(\pi_{new}) = \sum_s \rho_{\pi_{old}}(s) \sum_a \pi(a|s) A_{\pi_{old}}(s,a)$$

k 由超参数计算，而单步策略的更新范围由 α 控制。

为了避免人工确定步长和近似保守策略迭代，TRPO[27]在保守策略更新[26]的基础上增加硬约束，通过以下方式选择新策略分布：

$$\pi_{\theta_{t+1}} = argmax_{\theta_{t+1}} \eta(\pi_{\theta_i}) + \sum_s \rho_{\pi_{\theta_i}} \sum_a \pi(a|s) A_{\pi_{\theta_i}}(s,a)$$

$$s.t.\ D_{kl}^{max}(\theta_{old}, \theta) \leq \delta$$

其中，约束项称为信赖域，它是一个硬约束，D_{kl} 是 KL 散度，也称为相对熵，用于测量两个分布之间的距离：

$$KL(P,Q) = \int P(x) log \frac{P(x)}{Q(x)} dx$$

PPO[28]为简化计算，消除了硬约束，并使用软约束剪辑防止策略更新过大（剪辑版本）：

$$L^{CLIP}(\theta) = \mathbb{E}_t[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$$

其中$r_t(\theta) = \frac{\pi_{\theta_{new}}(a_t,s_t)}{\pi_{\theta_{old}}(a_t,s_t)}$, $\theta = maximize_\theta L^{CLIP}$.

A.1.1.2　当前算法存在的问题

在本小节，我们将指出以往方法的两个缺点，总的来说，是训练不稳定问题。为了更好地提出我们的算法，本节只介绍第一个问题，对另一个问题的讨论留到本文的主体部分。

让我们首先回顾一下之前方法的策略表示。无论是 DDPG 还是 PPO，策略都表示为由神经网络建模的高斯分布：

$$\pi_\theta(a,s) = N(a; \mu_{\theta_1}(s), \Sigma_{\theta_2}(s))$$

其中：概率分布的均值 $\mu_{\theta_1}(s)$ 和方差 $\Sigma_{\theta_2}(s)$ 在更新过程中混合更新。

“混合”一词的意思是：首先，参数的梯度是隐函数，我们无法直接看到参数空间中每个参数的更新方向；其次，每个参数（均值和方差）的更新是不可分离的，我们不知道每个参数在多大程度上发生了变化，即对动作分布的控制不是直接的。以 PPO 和 TRPO 为例：

首先，TRPO 和 PPO 通过大量的工作来约束新旧策略之间作用概率分布的变化，这产生了大量在概率空间中的计算，如 TRPO 要计算 Hessian 矩阵，PPO 要计算每个样本的新旧参数之比。其次，监督学习的成功和 RL 的不稳定性表明，监督学习方式的学习效率更高，通过监督学习更新的评价网络通常收敛速度更快、更容易学习，对于行动者-评论家网络，价值函数的参数收敛速度快，损失下降相对平稳，而对于策略网络，尽管在每个样本上分配了 KL 约束，但仍然存在 π_θ 的平均值可能被推到安全区域之外，并且仍然被其他样本向外推的情况。回顾 PPO 的更

新：

$$L^{clip} = E_t[min(\frac{\pi_{\theta_{t+1}}(a_t, s_t)}{\pi_{\theta_t}(a_t, s_t)}, clip(r_t(\theta_t), 1-\epsilon, 1+\epsilon)A_t)]$$

其中$r_t(\theta) = \frac{\pi_{\theta_{t+1}}(a_t, s_t)}{\pi_{\theta_t}(a_t, s_t)} \in [1-\epsilon, 1+\epsilon]$ ，均值更新为：

$$f \leftarrow f + \lambda \frac{A_t}{\pi_{\theta_{old}}} \frac{\partial \pi}{\partial f} = f + \lambda \frac{A_t}{\pi_{\theta_{old}}} \frac{at - f}{g^2}$$

这个公式揭示了 PPO 更新的两个缺点。首先，当更新后的平均值 $f(s_t)$ 被推到安全区域之外时，PPO 通过梯度将其从该样本向内拉回，但平均值$f(s_t)$可能会继续被其他样本向外推，如图 2 左侧第一个样本所示。更新后的均值离旧的行动均值太远，导致更新后的策略与旧策略之间存在较大的 KL 差异，进而产生不稳定。其次，当优势函数为负时，更新的平均值会被推到另一个未探测的方向，这会带来额外的不稳定性，如图 1.2 左侧最后两个样本所示。为了克服上述缺点，我们提出了一种目标学习方法，该方法通过设定学习目标，将均值和方差的更新分离开来，并在有监督的情况下进行训练。

值得一提且具有讽刺意味的是，在 ICLR 2020 最新发表的论文[73]中，作者发现，PPO 真正提升效果的地方不是所谓的新策略和旧策略之间的梯度剪辑，而是源于代码层的一些技巧，这进一步支持了我们的发现。

除此之外，我们将在第二章从理论层面指出另一个不稳定性问题，并用实验结果加以说明。

A.1.2　量化交易中的人工智能

在这一节，我们首先简要介绍量化交易的发展历史，接着列举人工智能算法在此领域的常见应用，即投资者一般用监督学习模型代替人工预测。然后讨论了强化学习的一些潜在优势。

A.1.2.1　量化交易

简单地说，量化交易是指利用先进的数学模型代替人的判断，制定策略，即在计算机的帮助下，选择能够从海量历史数据中带来超额利润的“大概率”事件。

传统的量化交易方法和理论来源于金融数学。人们普遍认为，量化投资的先驱是马科维茨，他于 1952 年建立了基于均值-方差模型的现代投资组合管理理论（PMT）。对于给定的收益率，Markowitz 模型试图形成一个风险最小的投资组合，即组合的协方差最小。用数学公式表达，当 n 个风险资产的平均收益率为 $\mu =$

$(\mu_1, \mu_2, \ldots, \mu_N)^T$，协方差矩阵为 $\Sigma = [\sigma_{i,j}]_{i=1\ to\ N, j=1\ to\ N}$ 时，Markowitz 模型求解如下优化问题：

$$Min\ \frac{1}{2} w^T \Sigma w\ \ subject\ to\ w^T \mu = \mu_p, \mathbf{1}^T w = 1,$$

其中 w 是投资组合的权重列表。后来，夏普、林特纳和莫森独立地提出了资本资产定价模型[35]。资本资产定价模型研究资产价格的均衡结构，将单个资产的预期收益率与市场组合中的风险联系起来。CAPM 认为，金融资产的预期收益基本上由一个因素决定，即市场超额收益。如果市场投资组合 M 是有效的，则任何资产的预期回报率$E(R_i)$满足：

$$E(R_i) - R_f = \beta_i \big(E(R_M) - R_f\big) = \frac{\sigma_{iM}}{\sigma_M^2} \big(E(R_M) - R_f\big),$$

其中 β_i 是资产的 beta 系数，$E(R_i) - R_f$ 是市场投资组合的预期超额收益，$E(R_M) - R_f$ 是资产的预期收益。

在此基础上，两个重要的基本理论被提出。萨缪尔森和法玛在 1965 年提出了有效市场假说。有效市场假说认为，在一个完全竞争的市场中，所有有价值的信息都及时、准确、彻底地反映在股票价格中，除非存在市场操纵，否则投资者不可能通过分析过去的价格获得高于市场平均水平的超额利润。有效市场假说是描述和理解市场的基本理论，具有重要而深远的影响。套利定价理论（APT）[36]是 Ross 于 1976 年建立的，它认为资产的预期收益受到各种风险因素的影响。APT 通过不同风险因素的线性组合对资产的回报进行建模，即：

$$R_i = E(R_i) + \beta_{i,1} F_1 + \beta_{i,2} F_2 + \cdots + \beta_{i,H} F_H + e_i,$$

式中 R_i 是资产的收益率，$E(R_i)$ 是期望值，$\beta_{i,\mathrm{k}}$，F_{k}分别代表第 i 项资产对第 k 项因素的敏感性和第 k 项共同风险因素，e_i是资产的非系统收益。

此外，还有许多模型被创建或作为量化投资其他领域的补充。类似于期权定价模型[37]、VAR（风险价值）、MM（Modigliani 和 Miller）理论等。

在众多的量化模型中，多因子模型是最常用的金融资产定价和市场预测模型之一。多因子模型根据多个特征（因素）进行资产定价或预测市场走势，每个特征（因素）反映一种风险或市场异常。采用多因素模型挖掘 alpha 收益，又称市场超额收益。Fama 和 French 首先提出了一个三因子模型[38]，他们发现 CAPM 中的 beta 不能完全解释每只股票的收益率差异，差异其实源于其他因素，例如公司的市盈率、市盈率和市值。从今天的观点来看，Fama-French 模型只是一个线性回归，所谓的 alpha 因子就是特征。事实上，现在很多基于 AI 的方法，都属于多因子模型对现代

科学技术的继承和发展。

A.1.2.2　AI 驱动的算法和数字货币市场

在当下，人工智能算法的主要作用是预测价格的变化趋势，预测的目标可以是不同的形式，如收益率、涨跌方向、排名等。预测信号被嵌入到不同市场或不同频率的策略中。正如前文所述，目前量化交易中常用的机器学习算法都是从多因子模型发展而来的。现代机器学习方法的改进主要体现在两个方面：特征工程和模型表示。

但在具体应用之前，我们首先需要讨论机器学习在金融市场中的可预测性。EMH 表明，根据市场信息，不可能获得经济利润。然而，随后的实证研究发现了一些市场异常现象，如反转[39]和动量效应[40]。市场异常是指资产价格在经济上偏离由市场效率推导出的定价模型的情况。关于更多的市场异常，我们请读者参考[41]。这些异常现象可能是由于市场参与者的心理变化或非理性等多种原因造成的，这表明我们有可能利用市场的历史，根据市场信息提取出市场变动可预测的模式。

然而，由于以下原因，利用金融市场的历史数据来预测未来并盈利是很困难的。首先，金融数据噪音大、波动性大。部分原因是存在“噪音交易者”（或非理性交易者）[42]以及其他影响市场走势的隐藏因素，如政府政策变化和突发新闻等。第二，市场的运动是由多种模式驱动的，这些模式要么源于不同交易频率的交易者（如价值投资者、技术交易者）的不同投资风格，要么源于投资者不同的心理认知。在一个近乎有效的市场中，开发一种单一的模式很难弥补冲击成本并获得利润。第三，一个好的预测模型并不一定会导致一个好的交易策略。传统上，我们通过对所有样本的准确性来评估预测模型。但是，由于各种原因：如风险、流动性和某些规定，我们无法交易所有样本（例如，同一横截面的所有股票或在所有可能的交易时间点交易）。对于交易策略，我们更关心触发交易信号的样本的准确性。现有算法已经克服或缓解了一些可预测性问题，但仍有许多核心问题有待解决。

对于人工智能驱动的特征工程，有两个主要因素促使其出现：首先，由投资者挖掘的具有实际金融意义的因子，如收益率[44]、账面市盈率[45]等，其剩余 alpha 变得有限，因子效力较低。其次，复杂的非线性人工智能模型也需要多样化特征的支持。同时，大数据和巨大的计算能力也给人工智能算法带来了便利。一些科学家通过数据挖掘自动生成特征，例如[46]使用改进的遗传算法挖掘 alpha 因子，第四范式开发了 TemporalGo 自动生成时间特征和 FeatureGo 去自动将基本特征与高级特征结合，一些科学家使用 AutoEncoder[47]以无监督的方式提取数据集中有用的信息。除了特征生成外，还有许多人工智能驱动的特征选择方法，例如许多投资者

在 boosting 树模型中使用信息增益得分，一些方法分析深层神经网络中每个特征的重要性等。自动特征工程（AutoFeatureEngineering）是 AutoML 的一个广泛分支，量化应用在这一领域中引起了越来越多的关注。

对于模型表示，典型的人工智能模型 $F_\theta(X)$，从特征空间 X（即各种交易因素）学习到目标空间 Y（即股票未来价格趋势）的映射。通常，线性回归是多因子模型的标准算法。这样的线性模型简单、稳健、易于解释。但线性组合的表达能力和拟合能力较差。近年来，随着人工智能算法的发展，各种形式的非线性 $F_\theta(X)$因其具有更强捕捉复杂模式的能力而变得流行，如 XGBoost 和 LightGBM 等梯度增强决策树[50]，MLP、LSTM 等深度学习模型[49]或深度强化学习模型。目前，一种比较成熟的交易策略是先建立预测模型，然后根据预测结果构建投资组合。在这种情况下，机器学习的目标是最大化预测精度。

但是强化学习是不同的。强化学习以利润最大化为目标，直接输出仓位权重或交易操作。即将下游的具体任务与上游的模型相结合，这种思想与自然语言处理的发展非常相似。早期的 NLP 方法通常先将词转换为特征向量，例如使用 Word2Vec[58]，然后将向量输入到 RNN 模型或其他序列处理模型中。后来 BERT[59]被提出，它直接将特定的任务应用到模型训练过程中，压倒性地击败了所有的方法。RL 算法具有以下几个优点：第一，金融市场比较混乱，信噪比较低，因此很难准确预测价格方向。然而，通过与环境的交互，直接学习因子与未来收益的关系，相对容易实现。其次，传统模型被迫在任何环境条件下做出预测，因此必须与风险控制、择时方法等相结合，而当市场不稳定或不确定时，强化学习可以做出“不操作”的决策。第三，量化交易场景非常适合 RL 框架。注意，RL 的学习目标非常类似于经济学中的长期利润，而衰减率类似于货币的贴现率，状态和行为自然与市场和贸易操作相对应。此外，强化学习能够处理监督学习模型中标签模糊不清的复杂或不确定任务，例如执行交易指令，在监督学习框架下很难建模此类任务。一般来说，强化学习可以看作是智能预测和智能策略，也就是说，RL 框架根据从数据分布到预测目标的映射，进一步学习从预测到执行的策略。在作者的知识范围内，RL 在量化交易中的应用主要集中在股票市场和加密货币市场的高频交易任务上，可以追溯到上世纪 90 年代末[87]。该应用大致可分为三个领域：第一，投资组合管理任务[51]，代理通过实时调整每个资产的仓位权重来管理单个或多个资产的交易。第二，订单执行任务[83]，代理试图以最小平均价格在时间跨度 T 内交易至少 V 交易量，最终执行价格越低越好，一般应低于 VWAP 或 TWAP（交易量或时间加权平均价格）。第三个应用是做市场景[74]，在这种情况下，RL 代理的主要目标是在一定时期 T 内交易至少 V 个交易量，以带来交易流动性，同时通过套利来寻求

利润，并保持较低的库存风险。本文旨在解决加密货币市场中多资产的组合管理问题。

此外，区块链技术比如类似 BTC、ETH 这样的加密货币是当今最热门的领域之一，本文只将其作为一种商品来看待，而不关心其背后的交易、确认机制。我们选择加密货币市场进行实验的原因在于，加密货币交易对机器学习任务更加“友好”。因为：首先，世界各地的交易者可以全天 24 小时在比如 Okex 这样的交易所进行加密货币之间的交易，也就是说，其数据量远高于每天只能交易 4 小时且只在工作日开放的股票市场。其次，研究者利用 API 既可以查询实时数据，也可以发送交易指令，完备且方便，而在我国股票市场尤其是高频交易中不允许自动交易。第三，数字货币的价格曲线波动性较大，交易机会较多，算法交易者比例较高，市场较为规则。

A.1.3　论文组织架构

论文的结构如下：在第二章中，我们首先通过实验结果指出当前方法的不稳定性问题，然后正式提出我们的目标学习算法。我们将介绍三种方法：TDL-direct 法、TDL-ES 法和 TDL-ESr 法。TDL 方法族通过在 KL 信任域内有效地设定学习目标，保证了策略的提升。最后两种算法根据进化策略的更新规则选择学习目标[60]，不同于以往在参数空间进行 ES 的方法[61]，我们采用 ES 搜索更好的动作分布。在第三章中，我们在 Mujoco 平台上对 13 个连续控制任务进行了大量充分的实验。通过将我们的算法与当今最优的方法对比表明，在某些任务中，我们的方法显著优于最优方法，而对所有任务来看，TDL 不弱于最优方法。同时，我们的方法比 TRPO 和 PPO 更能有效地在策略空间施加更新限制，而且我们的算法由于其监督学习的模式，具有更强的样本重用鲁棒性，提高了数据效率。在第四章中，我们提出了基于 TDL 的自动交易框架，主要使用 TDL 算法控制投资组合权重，此外，TDL 还可以进一步与监督学习结合。本文详细介绍了交易系统的整个流程，包括数据处理、模型训练和交易设置，并在模拟市场进行交易回测。第五章对论文进行了总结，讨论了目前的研究成果和未来可能的研究方向。

A.2　目标分布学习

A.2.1　基于梯度更新方法的不稳定问题

在 A1.1.2 中，我们已经揭示了目前基于梯度方法的不稳定问题，即均值 $\mu_{\theta_1}(\mathrm{s})$ 和方差 $\Sigma_{\theta_2}(s)$ 的混合更新，即使存在 KL 散度限制，算法也可能将动作分布推到

不安全区域。在本节，我们进一步讨论当策略接近确定性时的梯度爆炸问题。

让我们在信任域范围内重写更新方程。在一次迭代中，新策略 $\pi_\theta(a,s)=N\left(a;\mu_{\theta_{new}}(s),\Sigma_{\theta_{new}}(s)\right)$从旧策略 $\pi_{\theta_{old}}(a,s)=N\left(a;\mu_{\theta_{old}}(s),\Sigma_{\theta_{old}}(s)\right)$更新而来。在 TRPO[27]中，策略网络将在 KL 限制内最大化以下目标：

$$L(\theta)=\mathbb{E}_{s\sim\rho_{\pi_{old}}}\left[\sum_a N\left(a;\mu_{\theta_{new}}(s),\Sigma_{\theta_{new}}(s)\right)A^{\pi_{old}}(s,a)\right]$$

$$s.t.\ max_{s\epsilon S}KL(N(\mu_{\theta_{new}}(s),\Sigma_{\theta_{new}}(s)||N(\mu_{\theta_{new}}(s),\Sigma_{\theta_{new}}(s)))\ \le\delta$$

式中，$\rho_\pi(s)=\mathbb{E}_{s_0}[\sum_{t=0}\gamma^t p(s_t=s|s_0,\pi)]$，$KL(.||.)$ 表示两个概率分布的 KL 散度。根据[26]，当 d 很小时，迭代可以保证策略的提升，即$\eta(\pi_{new})>\eta(\pi_{old})$。我们可以通过 MC(Monte Carlo)采样逼近上述目标：

$$\hat{L}(\theta)=\frac{1}{T}\sum_1^T[\widehat{A_t}\frac{N(a_t|\mu_{\theta_{new}}(s_t),\Sigma_{\theta_{new}}(s_t))}{N(a_t|\mu_{\theta_{old}}(s_t),\Sigma_{\theta_{old}}(s_t))}]$$

式中，s_t，a_t 是在时间 t 时轨迹上的样本 $(s_0,a_0,r_0,s_1,\dots,s_k,a_k,r_k,s_{k+1}\dots)$，并遵循旧策略。$\widehat{A_t}$ 是对 $A_t(s_t,a_t)=Q_{\theta_{old}}(s_t,a_t)-\mathrm{V(s)}$ 的估计，即优势函数。有许多方法可以估计 $\widehat{A_t}$ ，最常用的是 GAE（广义优势估计量）[64]。

我们证明当 Σ_θ 很小时，即策略趋于确定性时，θ的目标梯度将爆炸，导致训练过程不稳定。

让我们假设一个参数与状态无关的情况，即动作分布的标准差退化为一个变量，表示为 σ。在这种情况下，$\hat{L}(\theta)$变为：

$$\hat{L}(\theta)=\frac{1}{T}\sum_1^T[\widehat{A_t}\frac{N(a_t|\mu_{\theta_{new}}(s_t),\sigma_{new})}{N(a_t|\mu_{\theta_{old}}(s_t),\sigma_{old})}]$$

则$\hat{L}(\theta)$关于 θ的梯度为:

$$\frac{\partial\hat{L}(\theta)}{\partial\theta}=\frac{1}{T}\sum_1^T\left[\widehat{L_t}(\theta)\frac{\partial logN(a_t|\mu_\theta(s_t),\sigma)}{\partial\mu_\theta(s_t)}\frac{\partial\mu_\theta(s_t)}{\partial\theta}\right]$$

注意，对数概率密度（第一分式项）的平均梯度为$(a_t-\mu_\theta(s_t))/\sigma^2$，因此：

$$\frac{\partial\hat{L}(\theta)}{\partial\theta}=\frac{1}{T}\sum_1^T\left[\widehat{L_t}(\theta)\frac{\partial\mu_\theta(s_t)}{\partial\theta}\frac{(a_t-\mu_\theta(s_t))}{\sigma^2}\right]\propto\frac{1}{\sigma^2}$$

也就是说，与θ方向相关的梯度与旧策略的标准差成反比，并且是平方阶的。

显然，随着训练的进行，当策略接近确定性（即$\sigma \to 0$）时，下一次迭代的θ梯度会很大，这将把动作分布的平均值$\mu_\theta(s_t)$推到一个远离先前确定性旧分布的地方（可能已经是最优解）。如果下一个样本是“坏”的操作，算法又会逆推策略回到最佳区域。类似于步长较大时的梯度下降法，解会在最优值附近波动。不幸的是，几乎所有 Actor-Critic 框架下基于策略梯度的算法都会遇到同样的问题，因为它们无法避免使用概率密度函数的梯度。一些工作也发现了类似的问题，如[64]，它指出 REINFORCE[21]和 PEPG[65]在梯度更新中的同质关系。

我们设计了一个小实验来说明 PPO 的不稳定性问题。考虑一个简单的场景：状态与动作无关，环境从$s_t \sim U([0,1])$随机抽样一个状态，动作的 cost(即负的 reward)是$c(a) = a^2$。虽然成本是状态独立的，但它仍然被纳入策略网络。RL 代理的目标是使单步成本最小化，即$min_\theta[\pi_\theta(a|s)c(a)]$。显然，在这个设置中，最优策略应该是确定性的：即任何状态下，始终以概率 1 选择 $a = 0$。实验结果表明，PPO 存在振荡和发散的学习过程，但 TDL 方法（见下一节）避免了梯度的计算，能够很容易地找到更好的目标，因而不存在上述不稳定性问题。请注意，我们在 PPO 中剔除了一些非必要的技巧，比如设置作用分布的方差下限以避免方差变小，或者在损失函数中添加熵调节，或者在 PPO 中设置一个小剪辑操作。这些技巧对稳定学习过程没有帮助。结果如图 2.1 所示，PPO 既没有收敛到确定性策略，也没有学习到最优的行为分布均值。

A.2.2　目标分布学习

本文提出了一种目标分布学习算法，该算法将均值和方差的更新分离开来，通过监督学习对网络进行训练。该算法在两个步骤之间交替：第一步，TDL 不直接优化 $L(\theta)$，而是提出动作分布的目标参数，其改进了原策略下的期望优势函数。第二步，训练策略网络以匹配所提出的目标参数（目标行为分布）。

在第一步中，对每个状态样本，TDL 提出其目标参数，以最大化s_t的代理目标，即：

$$L_{t,1}(\mu,\sigma) = \mathbb{E}_{a\sim N(\mu,\sigma)}[A^{\pi_{old}}(s_t,a)] \quad (2\text{-}1)$$

或者最大化利用旧的价值函数改进目标分布的概率：

$$L_{t,2}(\mu,\sigma) = \mathbb{E}_{a\sim N(\mu,\sigma)}[\mathbb{I}\{A^{\pi_{old}}(s_t,a) > 0\}]$$

L1 和 L2 都受到 KL 约束：

$$KL(N(\mu_{\theta_{new}}(s_t),\sigma_{\theta_{new}}(s_t))||N(\mu_{\theta_{old}}(s_t),\sigma_{\theta_{old}}(s_t))) \le \delta$$

第二步变成一个回归问题，训练网络使输出参数和目标参数之间的 MSE 或

MAE 最小。

TDL 独立地提出均值和方差目标，因此可以对每个参数实现不同的搜索方法。对于第一步中的策略搜索，注意，只有估计的$\widehat{A_t}$是已知的，因此目标可能有偏。在第二步中，在更新过程中存在一个平衡，即一方面探索，保持相对较大的方差以便继续探索，另一方面要准确地移动到目标处并缩小方差，我们算法中的平衡将由特定超参数控制。

让我们考虑更简单的高斯分布形式，对角高斯分布和各向同性高斯分布。对于对角高斯分布，方差$\sigma_\theta(s) \in R^n$退化为一维向量。对于各向同性高斯分布，方差$\sigma_\theta(s) \in R$是一个标量，所有维度上的方差都是相同的。给定一个轨迹$\{s_t, a_t\}$，每个动作a_t从$N(\mu_{\theta_1}(\mathrm{s}), \sigma_{\theta_2}(s))$采样，其中$\mu_{\theta_1}(s_t), \sigma_{\theta_2}(s_t)$是策略网络的输出。对每个状态动作对$(s_t, a_t)$进行(1,1)-ES[67]的操作，并提出目标均值和方差为$\hat{\mu}_t, \hat{\sigma}_t$，$\theta_1$和$\theta_2$的更新由最小化$(\hat{\mu}_t - \mu_{\theta_1}(s_t))^2$和$(\hat{\sigma}_t - \sigma_{\theta_2}(s_t))^2$得到。

基于 ES，我们提出了两种 TDL 算法，区别在于目标均值的选择，而目标方差的更新完全相同。我们首先讨论方差的更新，对于目标方差，我们使用自适应方法，即更新规则能够自适应地探索变化缓慢的方差，以防止过早收敛和违反约束。

在流行的策略表述中，行为选择的方差与状态无关。因此，$\hat{\sigma}_t(s_t) \to \hat{\sigma}_t, \sigma_{\theta_2}(s_t) \to \sigma$，方差是策略网络的一个参数。则对角高斯分布的更新规则是：

$$y_i \sim N(\mathbf{0}, \boldsymbol{I}) \quad a_i = \mu_{\theta_1}(s_i) + \bar{\sigma}_t y_i$$

$$\bar{\sigma}_{t+1} = \bar{\sigma}_t \sqrt{\frac{1}{N}\sum_{i=1}^{N} y_i{}^2 \, \mathbb{I}\{\widehat{A}_i \geq 0\})} \quad (2\text{-}2)$$

直观来看，方差轮廓将向优势样本所在的方向拉伸，同时压缩优势样本稀疏的方向。显然，这种使用所有样本均值的方差一致更新，在稳定训练过程和施加约束方面起到了积极的作用，同时也限制了算法的解空间和能力上界。

同时，如果用神经网络表示，方差也可以与状态相关。那么，各向同性高斯分布的更新变成：

$$\hat{\sigma}_{i,t+1} = \hat{\sigma}_{i,t} exp(\xi_i \mathbb{I}\{\widehat{A}_i \geq 0\})$$

对角线高斯分布变成：

$$\hat{\sigma}_{i,t+1} = \hat{\sigma}_{i,t} y_i \mathbb{I}\{A_i \geq 0\} + \hat{\sigma}_{i,t} \mathbb{I}\{\widehat{A}_i < 0\} \quad (2\text{-}3)$$

我们通过对两个候选更新对象进行几何加权组合，以平衡状态相关和状态无关两种方法的优点，得到最终更新规则：

$$\sigma_{i,t+1} = \hat{\sigma}_{i,t+1}{}^{1/(\varphi+1)} \bar{\sigma}_{t+1}{}^{\varphi/(\varphi+1)}$$

其中φ是一个平衡因子，在之后的 Mujoco 实验中，我们将φ设为 1。

A.2.2.1　TDL-ES 算法

对于目标均值的更新，我们先使用 (1,1)-ES，其中父代和子代之间具有更好适应度的样本将被设为新的均值。旧策略的动作为“父代”，采样动作是“子代”。采样动作的优势函数则为后代相对于父代的适应度函数。当给定行动采样、状态和估计的优势值$(s_t, a_t, \widehat{A_t})$，目标均值则为：

$$\mu_{t+1}(s_t) = \mu_t(s_t) + \varepsilon \mathbb{I}\{\widehat{A_t} > 0\}(a_t - \mu_t(s_t)) \quad (2\text{-}4)$$

其中，ε 是步长。直观地说，当样本是一个好的子代时，平均值将会向“好”样本移动，否则保持不变。注意，当设置 $\mu_{old} = \mu_t(s_t)$ 和 $x = a_t$，以及将适应值设为 $\widehat{A_t}$ 时，目标 $L_{t,1}(\mu, \sigma)$ 本质上与(1,1)-ES 的目标相同，即 $Q^{\pi_{old}}(s_t, a) - V^{\pi_{old}}(s_t)$ 本质上与$f(x) - f(\mu_{old})$相同，因此，可以安全地认为 TDL 的更新规则可以优化 $L_{t,1}$。

我们的算法有两个重要的特性。首先，TDL-ES 可以通有效地限制更新前后的 KL 散度在信赖域内，根据文献[27]，符合保守迭代过程。

样本(s_t, a_t)在更新前后的 KL 散度可以写成：

$$KL_t = KL[N(f_{old}, g_{old}) || N(f, g)]$$

$$= \frac{1}{2}\sum_i [2log\left|\frac{g_i}{g_{old_i}}\right| + \left((\frac{g_{old_i}}{g_i})^2 - 1\right) + (f_i - f_{old_i})^2 / {g_i}^2]$$

可以假设$\left|g_i / g_{old_i}\right| \in [1 - \epsilon, 1 + \epsilon]$其中$\epsilon \ll 1$ 并且 $f \approx \hat{u}_t$，于是我们有：

$$E_t[KL] \leq \frac{1}{2}[\varepsilon^2 n + o(\epsilon^2)]$$

其中 n 是动作空间的维数，显然我们可以通过将步长ε设置为一个较小值来安全地限制 KL。

其次，目标均值和方差的更新等价于单步随机梯度下降。重写（2-3）将y_i替换为a_i, μ_{θ_t}：

$${\hat{\sigma}_{i,t+1}}^2 = (a_t - \mu_t(s_t))^2 \mathbb{I}\{\widehat{A_i} \geq 0\} + {\hat{\sigma}_{i,t}}^2 \mathbb{I}\{\widehat{A_i} < 0\} \quad (2\text{-}5)$$

因此(2-4)、(2-5)可以认为是以某步长λ_μ和 λ_σ施加的关于$L_{t,2}$的 SGD 过程：

$$\mu_{t+1} = \mu_t(s_t) + \lambda_\mu \frac{\partial L_{t,2}(\mu, \hat{\sigma}_{i,t}(s_t))}{\partial \mu}|_{\mu = \mu_t(s_t)}$$

$$\hat{\sigma}_{i,t+1}{}^2 = \hat{\sigma}_{i,t}(s_t)^2 + \lambda_\sigma \frac{\partial L_{t,2}(\mu_t(s_t),\sigma)}{\partial \sigma}|_{\sigma=\hat{\sigma}_{i,t}(s_t)}$$

基于 TDL-ES 的单步策略更新如图 2.2 所示。

A.2.2.2　TDL-ESr 算法

由于 MDP 的固有特性，单个 RL 代理无法在并行交互过程，因此在一条轨迹上只有一个子代。所以在 TDL-ES 中，我们只根据原始分布及其唯一子代设定目标分布，而忽略了 MDP 的时间结构。考虑到相邻样本的状态观测和动作分布变化不大，我们对 TDL-ES 算法进行了修正，使其吸纳相邻样本中更多的信息，我们称之为 TDL-ESr 方法。

首先，我们将目标均值替换为相邻 2N+1 点的加权平均目标均值 $\{a_{t-N}, a_{t-N+1}, \dots, a_t, a_{t+N-1}, a_{t+N}\}$：

$$\mu_{t+1}^{target\prime} = \sum_{i=-N}^{N} \frac{\mathbb{I}(\widehat{A_{t+\iota}} > 0)\widehat{A_{t+\iota}}}{\sum_{i=-N}^{N} \mathbb{I}(\widehat{A_{t+\iota}} > 0)\widehat{A_{t+\iota}}} a_{t+i}$$

这么做主要出于两个原因：一方面，由于相邻状态的相似性，我们可以假设几个连续的动作在同一分布内，因此 (1,1)-ES 演化为(1,N)-ES，其中 N 是邻居的数目。另一方面，如果连续动作之间的差异较大，则新目标平均值的大小将很小，甚至可能接近于 0，且平均方向是随机的。相反，如果连续的动作在同一方向，则会朝此方向更新。也就是说，我们更愿意朝着连续相同的方向去更新，因为这个方向更令人放心。

其次，我们还添加了一个步长参数η来平衡新目标与旧分布的关系，通过以下公式计算最终目标平均值：

$$\mu_{t+1}^{target} = \eta\mu_t(s_t) + (1-\eta)\sum_{i=-N}^{N} \frac{\mathbb{I}(\widehat{A_{t+\iota}}>0)\widehat{A_{t+\iota}}}{\sum_{i=-N}^{N} \mathbb{I}(\widehat{A_{t+\iota}}>0)\widehat{A_{t+\iota}}} a_{t+i} \quad (2\text{-}6)$$

TDL ESr 单步更新过程如图 2.3 所示。

A.2.2.3　TDL-direct 算法

除了基于 ES，我们还提出了 TDL 直接法来直接最大化以下目标：

$$maximize_{\mu,\sigma} \sum_a N(a|\mu,\sigma)A^{\pi_{old}}(s_t,a)$$

在 KL 约束下，$KL(N(new)||N(old)) \le \delta$。给定状态样本$s_t$、动作样本$a_t$和估计优势$\widehat{A_t}$，目标平均值设定为：

$$\widehat{\mu_t} = \mu^{old}(s_t) + sign(\widehat{A_t})\, min\left(1, \frac{\sqrt{2\alpha}}{|y_t|_2}\right) y_t \sigma^{old} \quad (2\text{-}7)$$

其中$a_t = \mu^{old}(s_t) + y_t\sigma^{old}$和$\alpha$是与信任域大小相关的超参数（正实数），认为：

$$\mu_{ti}{}^2 \le (min\left(1, \frac{\sqrt{2\alpha}}{|y_t|_2}\right) y_t)^2 \le 2\alpha$$

我们假设方差的变化是单独更新的，可以假设$\sigma_{i,t+1}/\sigma_{i,t} \in [1-\epsilon, 1+\epsilon]$，其中 ϵ 是一个小值。由（2-7）更新的旧策略和新策略之间的 KL 差异为：

$$KL[N(\mu_{old}, \sigma_{old})||N(\widehat{\mu_t}, \hat{\sigma})] = \frac{1}{2}\sum_i [2log\sigma_i + \frac{1+\mu_{ti}{}^2}{\sigma_i{}^2} - 1]$$

$$\le \frac{1}{2}\sum_i \left[2\, log(1-\epsilon) + \frac{1+2\alpha}{(1-\epsilon)^2} - 1\right] = n\alpha(1+2\epsilon) + o(\epsilon^2) \approx d\alpha$$

总之，对于 TDL-ES 和 TDL 直接法，我们都有效地满足了每个样本的 KL 约束。然而，在 TRPO 或 PPO 中，它们放松了从 max 到 mean 或全局 KL 散度的约束。A.3 的实验证明了单样本约束的有效性。

A.3　基于Mujoco的实验

我们在 OpenAI gym[69]环境的 Mujoco 平台上进行了大量实验。为了展示算法解决基本问题的能力，我们设计了三个方面的实验：1）连续控制任务的累计收益。通过 13 个 mujoco 任务的实验，我们的算法在累计收益和可重复性上都优于或不弱于目前的最优算法。2）我们的算法可以安全地进行样本复用。由于算法的监督学习方式，TDL 可以提高每个样本的训练轮数而不会破坏模型的稳定性。3）对于保守迭代问题，我们的算法比 TRPO 和 PPO 算法更能满足策略空间上的最大 KL 散度的约束，从而更有效地保证策略的优化。

A.3.1　连续控制任务的表现

我们将 TDL 算法族：TDL-ES、TDL-ESr、TDL-direct 与两种著名的基于策略的方法：TRPO、PPO，以及 MPO[70]、PGPE[71]和改进的 CACLA[72]进行对比。对于 MPO，不同于 TDL，MPO 在每个样本上使用目标概率密度的估计 Q 函数，这比 TDL 中使用的状态值函数更难应用于连续动作，并且 MPO 很难保持奖励尺度不变。对于 PGPE，我们使用与 TDL（原始 PGPE 使用线性策略）相同的策略网

络来实现 PGPE，以便进行公平比较。我们观察到，在需要精确控制的任务中，PGPE 的训练过程不如 TDL 稳定。考虑到 MPO 和 PGPE 的性能相似，为了节省图形空间，我们不显示 PGPE 的结果。此外，我们还实现了一个具有相同网络结构的批量版本的 CACLA，它可以被视为不设置目标分布的 TDL-ES 简化版本。TDL-ES 通过设定目标从保守更新中获益，在渐近性能方面优于 CACLA。注意，我们自己也实现了传统的基于策略梯度的方法 DDPG 和 A2C，其在 Mujoco 的表现优于 OpenAI 提供的基准，但是它们仍然比 TDL 差很多，特别是在复杂的环境中，比如仿人环境。由于这些方法与我们的方法没有直接关系，为了进一步节省空间，我们也不会展示它们的结果，读者可以很容易地根据 OpenAI 提供的基准中进行比较。总之，各算法的比较如图 3.1 所示。

如图 3.1 所示。TDL-ESr 几乎在所有任务中都获得最高的奖励，除了：在 Ant 环境 TDL-direct 为最好；仿人站立任务上 TDL-ESr 略低于 TRPO。因此可以认为，TDL 方法总体上优于所有基准。具体来说：1）与 MPO 和改进的 CACLA 相比，TDL-ES 和 TDL-ESr 在所有任务中均优于 MPO 和改进的 CACLA，证明我们方法设定学习目标的有效性。2）与 TRPO 和 PPO 相比。我们发现它们的性能在不同的环境中差别很大。对于精确控制任务，如 Reacher 和 InvertedPendulum，它们的性能比我们差得多，因为我们可以解决 2.1 中提到的不稳定性问题。对于两个仿人任务，TRPO 的效果很好，但学习过程是波动的，如图 3.1 图例中的百分比所示。3）在训练过程中，TDL 方法波动程度最小，算法更稳定。

对于 TDL 三种模型的比较，TDL-direct 由于严格受约束而具有很好的稳定性，但更容易收敛到局部最优。由于 TDL-ES 平衡了探索和利用，因此在微调后比 TDL direct 性能更好。TDL-ESr 在三者中表现最好，特别是在复杂环境中，因为它利用了动态演化规律。

A.3.2　样本重复使用的稳定性

数据效率在训练中起着重要的作用，直接影响到训练的时间和效果。off-policy 方法通常利用称为经验重放的历史数据来训练模型以提高样本效率，而 on-policy 方法可以在每次迭代中提升样本的训练轮数，但可能产生不稳定的问题。

图 3.2 的实验比较了 TDL 和 PPO 在不同训练轮数下的样本重用性能。TDL-ESr 的性能随着训练轮数的增加而提高，即使在训练轮数为 100 的极端情况下，也仅比最优轮数的结果差一点。相反，当训练轮数大于 60 时，PPO 训练过程会急剧恶化，即 PPO 中的样本重用不会提高性能，反而使训练变得不稳定。另一个值得注意的比较是，TDL-ESr 中的样本重用导致了性能的大幅度提高（60 轮的累计回

报约为 10 轮的 2 倍），而 PPO 的提高相对较小。

注意到 PPO 是沿着剪辑后的代理目标，用策略梯度来更新策略网络的，一方面，正如我们在 A1.1.2 中提到的，均值和方差的梯度是混合的，我们不能显式地表示动作分布的更新方向，另一方面，当重复优化代理目标时，将截取、屏蔽更多的样本，相应状态样本上的动作分布可能会偏离，这可能导致不稳定的更新，如图 1.2 所示。然而，在 TDL 算法下的模型被反复训练以匹配已知稳定的目标分布，从而可以安全地提高样本重用率。

A.3.3　约束 KL 散度

依赖于 CPI（保守策略迭代）的算法，如 TDL、TRPO 和 PPO，需要在状态行为空间中对策略更新进行约束。一个最常用的约束是 KL 散度。我们想测试这些算法对 KL 约束的限制有多好。我们遵循 TRPO 中的意图，在每次迭代中记录所有样本的最大 KL 散度，在实验中，我们首先在每次迭代中采样 2048 个（步数）轨迹，然后计算每个状态动作样本的 KL 散度，最后记录最大值。结果如图 3.3 所示。

实验结果表明：1）TDL 算法的最大 KL 散度受我们设置的阈值（即红色和绿色曲线都低于我们设置的水平上限）的有效限制，而 TRPO 算法的最大 KL 散度在所有任务中都超过了预先设置的限制，在有些任务中，甚至比边界高出 2 个数量级（100 倍）。2） 我们的算法特别是 TDL 直接法的最大 KL 散度远小于 TRPO 和 PPO（其中 KL 阈值设置甚至高于 TRPO），这说明了 TDL 在抑制 KL 散度和保持稳定的能力。基于上述观察，我们可以安全地得出结论：TDL 可以更有效地限制状态行为空间中策略的变化，从而保证策略迭代的更为保守。此外，我们的实验为[73]中的发现提供了更多的支持。

A.4　量化交易的应用

投资组合管理问题是在一系列标的之间进行资本再分配的决策问题，是一个动态调整权重的过程。决策者采用一定的策略来管理其投资组合，目的是使收益最大化。

用数学表达，投资者计划交易 N 种资产，资产的权重分配在每时刻上用一个 N+1 长度的权重向量表示：$W_t = \left[w_{1,t}, w_{2,t}, \dots, w_{N+1,t}\right], t \in [1,T]$，并且$|W_t|_1^1 = 1$。注意，额外多出首项表示“现金”项。

令$P_t = [1, P_t^2, P_t^3, \dots, P_t^{N+1}], t \in [1,T]$为所有资产的价格，第一项固定为 1，表示“现金”价格不变。

分别令$B_t^i = P_t^i + fee + C_t^i, S_t^i = P_t^i - fee - C_t^i, t \in [1,T], i \in [2, N+1]$为实际

的买卖价格。其中fee, C_t^i是交易费用和冲击成本。所有标的的交易费用是固定的，而冲击成本受市场环境的影响。

在交易过程中，研究者们会将连续交易时间近似为等间隔的离散时间点。在此近似下，价格变化仅发生在$t = n\hat{t}$时刻，其中$\hat{t}$是时间间隔，n 是正整数。这与实际情况相符，在中国股市中或加密货币市场，交易信息一般每 3 秒和 0.33 秒推送一次。

令投资者将从$t - \hat{t}$到t（以W_t为持仓权重）赚取的收益为R_t：

$$R_t = (S_t / B_{t-\hat{t}})^T W_t$$

则全部交易时间内的总收益为：$\mathbf{R} = \sum_t R_t$。

现代量化研究者采用人工智能算法，以 1）寻求最优的订单执行，即更低的买入价、更高的卖出价和 2）学习一个强大的策略来调整持仓权重。注意，对于订单执行问题，由于多智能体博弈所产生的高频微观结构非常复杂，研究者不得不对数学模型进行大量的假设，同时，高频交易的性能在很大程度上取决于数据传输的时延和机器的计算能力。这使得实际的工业应用很难达到与模拟实验同等表现的原因。我们相信组合管理类问题更容易学习，实验测试也更与实际情况相符。原因在于，仓位控制是相对长期的信号，$\hat{t}$通常设定在分钟级别，受到订单执行影响较小，而且更具可预测性。

基于监督学习模型的预测和交易决策通常是分开的。通常情况下，研究者首先制定训练目标，然后从历史交易数据中提取特征和学习目标，如预测价格走势、市场波动等。一般使用深度神经网络或基于决策树类的模型进行预测，再根据人类知识总结出的特定规则，将预测结果进一步转化为交易信号。例如，假设我们已经有了一个预先训练好的两分类模型预测价格上升或下降的概率，那么，一种常见的转化方法是设置一个阈值来过滤预测信号，在概率高于阈值时购买，否则不进行操作。监督学习框架如图 4.1 所示。

A.4.1　模拟环境

在实际应用中，构建一个合理的环境模拟器是非常重要的。该模拟器相当于对实际应用场景的数学建模，它不仅为数据驱动的 RL 代理提供数据样本，还包含具体的状态转换模型和遵循 MDP 过程的奖励函数。研究者在模拟器上进行实验，以测试 RL 模型的性能，并进一步将反馈作为算法改进的指导。此外，模拟器还能帮助研究者找到 RL 算法真正学到的东西，并提供一个平等比较的平台。

在本文中，我们不采用多代理环境，比如[79]，在这种环境中，大量投资者和做市商相互竞争，订单价格精确到买卖双方的价格。我们更关心权重分配信号的有

效性，而不是详细的订单执行，因此在假设我们的决策对市场方影响不大的情况下，构建了一个“一 vs 市场”的环境。

A.4.1.1　状态建模

状态包括三个部分：市场观测、提取的市场因子和持仓账户信息。

我们假设一个完全可观察的环境，将前两个分量统称为市场变量，将帐户信息命名为持仓变量。也就是$S_t = S_{Mkt,t} \cup S_{pos,t}$。

在订单驱动的市场中，这里的市场观察是订单请求消息，具体来说，有两种类型的指令：限价订单（LO）和市场订单（MO）。对于限价指令，交易者向交易所发送关于交易量和交易价格的请求。相同价格的订单将在订单簿上以相同的位置收集。交易价格保证不劣于限价，即不低于卖出限价，不高于买入限价。交易的促成主要有两个原则：价格优先和时间优先。也就是说，以卖出方为例，价格较低的订单将首先进行交易，价格较高的订单只能在所有较低的订单都成交之后才进行交易，对于价格相同的限价订单，则按挂单时间顺序进行交易。LOB 的结构如图 4.2 所示。对于市价单，投资者只需告诉交易所想买或卖的数量，市价单会主动“吃掉”订单簿中对手方的订单，保证完成交易量。订单簿的快照通常每秒发送几次，在我们的设置中为每秒更新一次。

K 线数据包含在每个时间间隔结束时刻的几个统计指标。最重要的是最高价、最低价、开盘价、收盘价、成交量、成交额，前两个指标分别表示区间内最高成交价格、最低成交价格，开盘、收盘价分别表示间隔内第一笔和最后一笔成交价格。区间的成交量和成交额以数量、金额表示。给定一个时间间隔 T，状态S_t的市场观测是$t-T\sim t$期间 K 线指数和 LOB 快照。

市场因子是基于市场观测来计算的，这些高阶特征能更有效地刻画了市场状态。在我们的环境中，主要使用以下因子，其在监督学习模型中起着关键作用：

基于 LOB 快照的因子：令买方 n 档的价格和数量为Bid_{nprice}，Bsk_{nvol}，卖方同理。

OFI（订单流不平衡）：反映 LOB 的流动性。我们知道，交易和取消订单将降低订单的流动性，而限价单会增加流动性。各价格水平下订单流的 OFI 随时间变化。具体公式见[89]。

VOLR：通过比较前三个价格档位下买卖双方的相对规模来反映双方盘口压力。

$$VOLR = \beta_1 \frac{Ask_{1vol}}{Bsk_{1vol}} + \beta_2 \frac{Ask_{2vol}}{Bsk_{2vol}} + \beta_3 \frac{Ask_{3vol}}{Bid_{3vol}}$$

中间价 MA（移动平均线）：

$$MA_{mid,n}(t)=\frac{(Bid_{1_{price}}(t)+Ask_{1_{price}}(t))/2}{mean((Bid_{1_{price}}(t)+Ask_{1_{price}}(t))/2,t\in[t-n,t])}$$

基于 K 线指数的因子：令 t 时刻的收盘价为$Close(t)$，时间间隔为δ。

EMA（指数移动平均）：最近 N 个时间点的加权平均价格，衰减因子β：

$$EMA_N=\frac{1}{N}\sum_{k=1}^{N}\left(\frac{N-1}{N+1}\right)^k Close(t-k\delta),$$

MACD（移动平均收敛/发散）= $EMA_{N1}-EMA_{N2}$，

相对强度指数（RSI），通过比较买卖双方的“强度”来反映一定时期内市场的繁荣程度。

除了上述常见因子外，我们还通过数据挖掘方法创建了一些技术因子[46]。总的来说，我们在 Mkt 变量中共使用 30 个量价因子。

对于仓位变量，包括：1）当前头寸，即观察时间 t 时的权重表：$W_t=[w_{1,t},w_{2,t},\dots,w_{N+1,t}]$ 2）截至目前，用户赚取的利润P_t和总交易量V_t。3）时间比例：t/T。各仓位变量的作用将在 A4.2 中讨论。

综上所述，模拟器在时间 t 提供的状态观测为：

$$S_t=\{[LOB(i),i\in[t-n,t]]\cup[Close(t),Open(t),\dots,Amount(t)]\cup[OFI,VOLR,\dots,EMA,MACD,\dots]\cup\left[W_t,V_t,P_t,\frac{t}{T}\right]\}$$

A.4.1.2　动作和奖励

动作：

模拟器所输入的动作是每个标的的交易量，可以直接从W_t转换。在后面的讨论中，我们将“现金”作为W_t的第一项。注意，在我们的设置中，投资者在整个测试期间只能发送一种类型的交易订单，也就是说，每种标的是多头还是空头都是预先确定的，不能混合。

当给定W_t时，动作A_t为：

$$A_t^i=Vol_t^i=Capital*(W_t^i-W_{t-1}^i)/P_t^i$$

式中，P_t^i表示第 i 个资产最新 k 线的收盘价。动作的正值表示“买入”动作，负值表示“卖出”。如果我们将资本设为当期累计利润（又称复利交易），由于平仓实际上属于市场变量，我们也可以将W_t-W_{t-1}本身视为一种行为。为了更直观的理解，让我们举三个例子：1）当投资者把所有的钱都押在单一资产上时，假设为第二项，那么他所采取的行动应该是$W_t=[0,1,0,\dots,0]$。2）当投资者对当前市场

没有信心，并保持空头头寸时，动作应为$W_t = [1,0,0,\dots,0]$。3）当投资者对所有证券具有同等信心时，在这种情况下，动作应为$W_t = [1/N, 1/N, 1/N, \dots, 1/N]$。

为了模拟一个更为现实的交易环境，我们在每个权重项上加上了一个成交率$\gamma_t \propto 1/(Ask_{1vol} + Bid_{1vol})$，直觉来说，当市场上挂单数量足够时，该订单容易成交，反之，当市场不活跃时，该订单很难成交。因此，权重列表被修剪为：

$$\begin{cases} W_t^i - W_{t-1}^i *= \gamma_t^i \\ W_t^0 = 1 - \sum_{i=2}^{N+1} W_t^i \end{cases}$$

奖励：

对于 RL 在高频交易中的应用，奖励（收益）设计主要有三种类型：零利润（在订单执行场景中，代理不追求利润）、短视（代理一步搜索最优解，类似贪婪算法）和最优（代理关心全局最优）。我们的奖励设置属于第二种类型，因为当且仅当每个步骤中的操作是最优时，全局奖励才是最优的。

有两种报酬，一种是及时报酬，即每次迭代的利润，另一种是对整个交易过程的评价。具体来说，给定一个交易行为$W_t - W_{t-1}$，及时报酬的计算方法如下：

$$R_t = \left(\frac{P_t}{P_{t-1}}\right) W_{t-1} - |W_t - W_{t-1}| * (fee + C_t)$$

P_t是 t 时刻的价格，手续费是固定的，C_t是冲击成本或滑点，为方便起见，C_t是从均匀分布$C_t \sim U(0, C)$中抽样的随机变量，其中最大成本 C 是一个超参数。

对于全局奖励，为交易量相关的额外奖励，只在交易结束时返回。因为在加密货币领域，交易所通过提供更低的费用来鼓励更多的交易，所以手续费随交易量递减，当交易量达到一定数量时，甚至产生零甚至负手续费（从交易中赚取利润）。因此，这种手续费折扣是通过在交易结束时的额外奖励来体现的：

$$R_{end} = R_T + \beta * V_T$$

其中β为红利比率，V_T为整个交易期间的平均持仓率。

A4.2 描述了更多的奖励函数的设计，A4.4 给出了不同奖励函数实验对比结果。

A.4.2　RL 框架

我们提出的 RL 框架见图 4.3.

改进的 TDL-ES：

为了更好地应用 TDL，我们在算法和网络结构上做了一些修改。

模型输入不同于 Mujoco 任务，是多输入。我们只使用提取的市场因子作为神经网络的第一个输入张量，为一个形如(N,M)的二维矩阵，列是 M 个因子，例如

MACD、RSI、OFI 等，横轴表示每个 N 个资产。第二个输入张量是持仓变量$[W_t, V_t, P_t, t/T]$，我们将$W_t[1:]$合并到第一个隐藏层的输出张量中，并将$V_t, P_t, t/T$作为三个额外的变量添加到第三层的输入中。

对于网络结构，我们将 Actor 保持为三层神经网络，每层的操作修改如下：前两层只进行水平卷积操作，即在每个因子内逐行提取特征。之后，第二层的输出成为标准的一维向量，然后将其输入第三层，挖掘标的之间的关联。同时，为了满足约束($\sum_{i=1\sim N+1} W_{i,t} = 1$)，在输出层后增加一个 *softmax* 激活层。修改后的神经网络结构如图 4.4 所示。

在算法上，我们加入最优的遗传子代来改变 A2.2.1 中提出的 TDL-ES 的均值目标。回顾 TDL-ES 的目标均值：

$$\mu_{t+1}(s_t) = \mu_t(s_t) + \varepsilon \mathbb{I}\{\widehat{A_t} > 0\}(a_t - \mu_t(s_t))$$

其中a_t是由 1,1-ES 产生的后代，$\widehat{A_t}$是此后代的优势函数值。对于高频交易，当下一个状态给定时，最优行为是已知的。即

$$a_t{}^* = argmax_a \, earning \, rate = argmax_w \, f(P_t, P_{t+1}, w) = [0, \dots, 1, \dots 0]$$

我们只考虑价格变化而忽略消费费用或成本。也就是说，在原始的报酬设计中，最优的行为$a_t{}^*$是将所有的钱分配给收益率最高的资产（如果所有的资产价格都下跌了，代理人应该把所有的钱都保存为现金）。因此，增加了一个最优决策来修正平均目标：

$$\mu_{t+1}(s_t) = \mu_t(s_t) + \varepsilon_1 \mathbb{I}\{\widehat{A_t} > 0\}\big(a_t - \mu_t(s_t)\big) + \varepsilon_2 a_t{}^*$$

式中，ε_2表示修改程度。从理论上讲，ε_2的值应该与初始作用与最优作用之间的距离有关，以保证更新的顺利进行，为了方便起见，我们在实验中选择了ε_2作为ε_1的 1/5。

其次，作为量化交易领域的一个不可避免的问题，训练数据和测试数据之间总是在数据分布和映射关系$X \to Y$上存在偏差，而对于固定时间周期的 RL 学习过程，其状态转移轨迹固定，情况变得更糟。RL 代理，从空仓开始，遵循给定的市场轨迹，可能很难产生公正的经验以及有效的探测样本。为了解决这个问题，我们给出了一个具有一定概率 Ψ 的随机初始化。具体来说，我们在整个时间范围内的随机时间点开始算法，并分配一个随机的初始权重列表，目的是鼓励探索和消除偏差。实验结果表明，该随机启动方案具有较好的稳定性和鲁棒性。

奖励函数设计:

奖励函数设计是 RL 研究的一个方向，它有助于加速模型训练、提高奖励或稳定训练过程。对于交易任务，获利不是唯一目标，风险控制也同样重要。一些 RL

算法[33]将可控风险趋势机制视为优势。风险倾向一般包括风险规避、风险中性和风险寻求。

我们通过修改奖励函数以反映风险趋势。原始报酬函数表示为风险中性，因为它客观地反映了交易利润。下面，我们提出两种实现风险可控趋势的报酬函数：

在第一个设计中，我们增加了对仓位分散的惩罚。作为交易中的共识，投资者通常通过投资不同的产品来降低仓位风险，即降低单个或类似项目的风险暴露。我们用标准差来制定权重列表的离散度，并增加对离散度的惩罚。

$$R_t = \left(\frac{P_t}{P_{t-1}}\right) W_{t-1} - |W_t - W_{t-1}| * (fee + C_t) - \eta \frac{std(W_t)}{std([1,0,\dots,0])} \quad (4\text{-}1)$$

其中分母 $std([1,0,\dots,0])$ 表示将所有资本分配给单一证券时的最极端情况。当钱平均分配时，标准差等于 0。我们称这种奖励为仓位控制奖励。当 η 取正或负，分别表示风险规避和风险寻求。

对于第二个设计，为了突出每种证券之间的差异，我们对回报期进行非线性放缩，因此回报函数变成：

$$R_t = \left(\frac{P_t}{P_{t-1}}\right)^{(1+\eta)} W_{t-1} - |W_t - W_{t-1}| * (fee + C_t) \quad (4\text{-}2)$$

在这种情况下，η取正表示寻求风险，取负表示规避风险。

最后，受股票市场阿尔法策略的启发。投资者更关心对冲收益或相对收益，而不是绝对收益。在股票市场，对冲策略的回报率是投资组合回报率减去市场回报率。另一方面，在样本评价中，一些 RL 算法也使用的是优势函数而不是绝对 Q 值。因此，我们修改了收益项，加重对有超额收益的资产的关注：

$$R_t = \left(\frac{P_t}{P_{t-1}} - mean(\frac{P_t}{P_{t-1}})\right) W_{t-1} - |W_t - W_{t-1}| * (fee + C_t) \quad (4\text{-}3)$$

其中所有标的的平均收益可以被视为市场回报。例如，在价格漂移期，所有证券的价格都有不同程度的上升或下降，（4-3）有助于将更多的权重转移到上涨更多的资产上。

A.4.3　与监督学习模型集成

由于交易数据的信噪比较低，纯强化学习算法可能不够稳定，甚至无法收敛。同时，大量非理性交易者造成的不规范交易行为和突发新闻带来的剧烈波动，将进一步加剧不稳定问题。因此，我们进一步将强化学习与监督学习模型相结合。在预测信号的指导下，强化学习策略可以取得更好的效果。

我们提出了两种类型的组合：第一种，两种方法在信号水平上集成的外部链接

法，另一种是设置预测信号退化为状态变量的内部链接法。两种组合的框架分别如图 4.5 和 4.6 所示。

对于外部链接法，两个信号通过以下规则进行简单组合：根据预测的概率和阈值，将证券标记为“上升”或“下降”。我们首先将“下降”证券的权重设为 0，然后分别测试：1）将剩余权重重新分配给现金，以规避风险。2） 将它们按相同比例分配给标记为“上涨”的证券。

对于内部链接法，我们将每个标的的预测概率都添加到第一层O_1的输出中即最后一个权重单元后面（如图 4.4 所示）。

A.4.4　交易回测

实验设置

以下实验是基于 Phoenix 交易所的数据。Phoenix 是一家加密货币交易所，全世界的交易员可以 24 小时在不同的加密货币之间进行交易。在这组实验中，我们使用了 8 种币币 BTC 交易对的数据：ETC/BTC、ETH/BTC、GAS/BTC、LTC/BTC、LSK/BTC、XRP/BTC、EOS/BTC 和 USDT/BTC。对于每个交易对，一个样本对应一个市场快照，该快照大约每秒捕获一次。请注意，我们使用 BTC 作为基础货币，而不是人民币或美元，以消除加密货币市场和现实世界货币之间的漂移影响。实验使用的训练样本为 2018 年 4 月至 8 月的连续 90 个交易日，总数为 6000 万。时间间隔 T 为 10 分钟意味着我们在每 10 分钟结束时改变投资组合的仓位权重，并且在下一个时间间隔内不进行操作。

我们在同一交易区间比较下列方法：

监督学习模型（SL）：使用 LightGBM 训练二分类问题。输入特征是 A4.1.1 中引入的市场因子。我们将前 30 天作为训练集，并在测试集上在线学习，以周频更新我们的模型。在后 60 天内测试模型的表现。按如下规则转换交易信号：先根据阈值将预测结果划分为“上升”或“下降”类别，然后将现金平均分配给每个“上升”证券。根据经验，阈值设定为 0.7。

TDL 模型（TDL）：A4.2 中描述的改进的 TDL-ES 方法。具体来说，策略网络的每层单元数分别为 64,1,9，ε_1,ε_2取 1 和 0.2，并使用原始奖励函数。对应于 SL 方法，我们同样使用前 30 天训练 RL 代理，并在接下来的 60 天内进行测试。RL 算法每周末都重新训练。

外链接集成模型（SL+TDL-Outer）：SL 和 RL 模型的外连接组合。训练-测试设置同上。根据权重分配的不同，将剩余权重分配给现金或预测为“正”的资产的方法分别称为 SL+TDL-outer-1 和 SL+TDL-outer-2。选择 SL 模型的阈值为 0.6。

内链接集成模型（SL+TDL-Inner）：两个模型的内链接组合。

测试不同奖励机制的 TDL 模型：对两种不同风险倾向的报酬变量（4-1）和（4-2）进行测试，其中（4-1）的η设为-1e-4 或 1e-4，而（4-2）的η设为 2 或-0.5，实现了风险寻求和风险规避。对冲收益即（4-3）也进行了测试。

对于奖励函数，手续费为 0.001，额外成本C_t取自 $U(0,0.001)$。交易价格P_T是每间隔 T 的最后一个价格。注意，所有算法的测试交易时长都是 60 天，因此当代理在每 10 分钟内管理其投资组合时，总共有 8640 次换手操作。根据交易所实际提供的消费费用优惠政策，我们将终端奖励函数 β 设为 8640/2000=4.32，即库存提高 20%，消费费用降低 1/10000。

性能指标

日收益率（Daily PCT）：我们进行非复利的交易测试，即每个区间的交易资金固定为 1，盈利或亏损的部分进行回收或扣除，不再投入下一区间。假设 N 天的总 Pnl 为Pnl_{total}，则日收益率为Pnl_{total}/N。

胜率（WR）：假设交易总次数为 N，我们有 n 次获利，于是策略的胜率为 n/N，该指标相当于分类模型的准确率。

日波动率（Daily Vol）：日收益的标准差。

持仓比率（有效信号比率）：各区间存货的平均比率，即$mean(\sum_t \sum_{i=2}^{N+1} W_{t,i})$。如前所述，大量交易可以产生手续费折扣，这是在最终奖励中反馈的。这个指标相当于分类任务的召回率。另外，在其他交易场景中，比如做市和执行指令，实现更多的交易量比赚取利润更重要。因此，我们认为提供有效信号的能力也是算法的一个重要能力。

A.4.4.1　回测表现

各方法的回测指标如表 4.1 所示。在表 4.1 中，上部分显示没有最终交易量奖励的各指标值，下表是所有 Pnl 的总和。

根据实验结果，观察到的几种现象：1）根据每日 PCT，比较关系为：监督学习法<强化学习法<SL 和 RL 相结合。且内链接是最优的。尽管监督学习方法的 WR（即准确率）很高，但传统的“阈值切分”策略在有效信号率仅为 40%的情况下导致收益很低。2）有些奇怪的是，RL 代理的胜率很低，大约仅为 1/3，注意随机猜测的 WR 是 1/2。但是 RL 代理却能获得很大的利润。我们认为，这是 RL 代理所学到的交易策略，即它更愿意在冒着微弱亏损的风险去赚取高额利润。相反，对于阈值为 0.7 的 SL 方法，WR 为 70%+，对于阈值为 0.6 的外链接组合，WR 为 60%+，监督学习模型的准确率相对较高。3）与监督学习方法相比，外链接方法有较高的

收益率和较低的波动率，具有更好的性能。在同样的日收益率指标下，第一外链法与第二外链法相比，由于库存比例较低，总 pnl 更低。4）与第二外链法相比，内链法的日均收益更高，交易量也更大，故进一步提高了 Pnl 的总收益，总之，将监督学习模型的决策合并到 RL-agent 中，使硬“切分”变为软“切分”，可以获得更好的性能。

为了进行更直观的比较，我们还在图 4.7 和图 4.8 中展示每种方法的回测曲线和每日收益情况。

A.4.4.2　基于不同风险倾向的奖励函数

我们进一步比较了不同报酬函数在风险可控趋势下的表现，结果如表 4.2 所示。结论如下：1）风险倾向没有明显的改善收益，但在一定程度上，风险寻求者获得了更高的利润和更高的波动性，这一结论与文献[79]一致。2） 对冲收益没有达到预期的效果，原因是在正式的股票市场中，整个股票之间的秩关系是非常规则的，同时市场的变化与每一个股票相关，使得每一个股票的秩序列和超额收益是可预测的。而在加密货币市场中，我们不能将投资组合的均值作为市场指数，我们的对冲操作可能会打破市场特征与交易策略之间的联系。

个人简历、在学期间发表的学术论文与研究成果

个人简历

1995 年 04 月 04 日出生于河南省洛阳市。

2013 年 9 月考入华中科技大学人工智能与自动化系自动化专业，2017 年 7 月本科毕业并获得工学学士学位。

2017 年 9 月免试进入清华大学交叉信息研究院攻读硕士学位至今。

发表的学术论文

[1] Zhang C , Li Y , Li J . Policy Search by Target Distribution Learning for Continuous Control[J]. AAAI 2020 oral.