From Deterministic to Stochastic: An Interpretable Stochastic Model-free Reinforcement Learning Framework for Portfolio Optimization

Zitao Song[1†], Yining Wang[1†], Pin Qian[2], Sifan Song[1], Frans Coenen[3], Zhengyong Jiang[1*] and Jionglong Su[4*]

[1]Department of Mathematical Sciences, Xi'an Jiaotong-Liverpool University, Suzhou, China.
[2]Department of Computer Sciences, Xi'an Jiaotong-Liverpool University, Suzhou, China.
[3]Department of Computer Sciences, University of Liverpool, Liverpool, United Kingdom.
[4]School of AI and Advanced Computing, XJTLU Entrepreneur College (Taicang), Xi'an Jiaotong-Liverpool University, Suzhou, China.

*Corresponding author(s). E-mail(s): Zhengyong.Jiang@xjtlu.edu.cn; Jionglong.Su@xjtlu.edu.cn;
†These authors contributed equally to this work.

Abstract

As a fundamental problem in algorithmic trading, portfolio optimization aims to maximize the cumulative return by continuously investing in various financial derivatives within a given time period. Recent years have witnessed the transformation from traditional machine learning trading algorithms to reinforcement learning algorithms due to their superior nature of sequential decision making. However, the exponential growth of the imperfect and noisy financial data that is supposedly leveraged by the deterministic strategy in reinforcement learning, makes it increasingly challenging for one to continuously obtain a profitable portfolio. Thus, in this work, we first reconstruct several deterministic and stochastic reinforcement algorithms as benchmarks. On this

basis, we introduce a risk-aware reward function to balance the risk and return. Importantly, we propose a novel interpretable stochastic reinforcement learning framework which tailors a stochastic policy parameterized by Gaussian Mixtures and a distributional critic realized by quantiles for the problem of portfolio optimization. In our experiment, the proposed algorithm demonstrates its superior performance on U.S. market stocks with a 63.1% annual rate of return while at the same time reducing the market value max drawdown by 10% when back-testing during the stock market crash around March 2020.

Keywords: Portfolio Management, Reinforcement Learning, Deep Learning, Quantitative Finance

1 Introduction

With the aim of maximizing return, portfolio optimization is a decision-making process that continuously allocates funds into various financial derivatives [1]. A key challenge for this is carefully balancing the multidimensional information and sometimes conflicting objectives of various decision processes in a noisy financial environment. Thus, many trading algorithms are expected to operate on this fine granularity. Traditionally, many machine learning and deep learning methods have been used to predict future price trends and fluctuations [2–4]. Nevertheless, one of the inherent difficulties of these price-prediction-based algorithms is to forecast future stock behavior with a high accuracy level. In fact, given the Efficient Market Hypothesis [5], it is nearly impossible for any trader to hypothetically outperform the market and consistently produce risk-adjusted excess returns (alpha).

Lately, deep Reinforcement Learning (RL) has attracted much attention due to its remarkable achievements in playing video games [6] and board games [7]. In RL, an agent's current behavior is closely related to its future rewards through multiple interactions with its environment. Such behavior allows the agent to gradually adopt an action that can maximize rewards and minimize penalties without predicting future states. This learning process is natural in biological life forms, and it has also been shown to be highly effective for artificial agents [8]. As a matter of fact, incorporating deep neural networks into the reward-penalties learning process gives the deep RL an inherent edge in many different applications.

There have been many successful attempts to implement model-free deep RL algorithms on algorithmic trading problems. This includes the value-based RL and policy-based RL algorithms. By discretizing market actions, Lucarelli and Borrotti [9] propose a value-based RL which applies recent adaptations of Q-learning, e.g., Deep Q-Network [6], Double-DQN [10], and Dueling-DQN [11], to portfolio selection problems. Although discretizing market action is feasible, with the growing number of assets as the input and higher dimensional

action as the output, it becomes increasingly difficult for the neural network to handle.

In order to accommodate more assets, Jiang et al. [12] introduce an RL framework based on the actor-critic Deterministic Policy Gradient Algorithm [13, 14], a technique that is combined with value-based RL and policy-based RL, that can continuously output actions through the policy function approximated by a neural network. However, states in [12] still depend on historical stock prices with only three features, i.e., highest, lowest, and close prices of stocks. This is a relatively simplified assumption since the stock market operates far beyond the scope of these three features and rather independently of past performance. Indeed, the external environment, including the global economy and companies themselves, has a significant impact on the stock market. Moreover, the market strategy in [12] is deterministic, thus its trading agent is highly conservative and lacks the ability to explore and gain alpha returns.

In addressing these problems, our contributions are four-fold. First, we benchmark several classic RL algorithms, Deep Deterministic Policy Gradient (DDPG) [14], Twin-Delayed DDPG (TD3) [15], and Soft Actor-Critic (SAC) [16, 17], in the continuous portfolio optimization action space. Second, to imitate the uncertainty in the real financial market, we propose a novel state-of-the-art stochastic reinforcement learning framework inspired by Soft Actor-Critic (SAC) and Quantile-Regression DQN (QR-DQN) [18, 19], namely **S**tochastic **P**olicy with **D**istributional **Q**-Network (SPDQ) for the dynamic management of the stock market portfolio. Importantly, we create a novel structure containing a stochastic policy, modeled by Gaussian Mixtures, and a distributional critic modeled by quantile numbers. Third, we enrich the state space by adding additional qualitative financial factors. Additionally, we reformulate the one-step reward by adding a risk term to the simple return. Fourth, we provide the interpretation of the model strategy, and an ablation study for different hyperparameters to better serve the diverse input states as well as to assess the robustness of our proposed algorithm.

The rest of this paper is organized as follows. Section 2 provides a comprehensive review of previous model-free reinforcement algorithms and their applications in portfolio optimization. Section 3 gives a mathematical definition of the portfolio management problem. Section 4 first presents a basic preliminary of reinforcement learning and introduces the relevant methodologies, including both deterministic and stochastic algorithms to address our problem. Section 5 details experimental procedures and results corresponding to the proposed algorithms. In Section 6, the conclusions for this research are given.

2 Related Works

2.1 Stated-of-the-Art RL Algorithms

Previous works have used Deep Deterministic Policy Gradient (DDPG) [14] and Twin Delayed Deep Deterministic Policy Gradient (TD3) [15] to generate

deterministic continuous action space. On the contrary, Soft Actor-Critic [16, 17] learns a stochastic action by maximum entropy reinforcement learning, including a temperature hyperparameter used to control the importance of return and entropy. Apart from stochastic action space, C51 [18] and QR-DQN [19] learn a value distribution and highlight the ways in which the value distribution impacts learning in the approximate setting. Given the uncertainty of the financial market, the built-in stochastic settings in SAC [16, 17] have an edge over the Gaussian Noise added to deterministic DDPG [14] and TD3 [15] policy when it comes to exploration. Consequently, unlike fixed rewards in video games, estimating a value distribution of the cumulative market return, rather than averaging its randomness, becomes increasingly important.

2.2 DRL Applications in Stock Trading

Current mainstream RL uses an accumulated discounted reward as the objective function. Among the stock trading research on this discounted reward settings, Liang et al. [20] propose two adapted versions of policy-based RL algorithms based on Proximal Policy Optimization (PPO) [21] and Policy Gradient (PG) for portfolio management on China's stock market. Lucarelli and Borrotti [9] implement the Deep Q-Network, Double DQN, and Dueling DQN, which all are value-based RL algorithms. For cryptocurrency portfolios, Jiang et al. [12] apply Deterministic Policy Gradient (DPG), containing both policy network and value network to solve portfolio optimization problems. Wang et al. [22] use a hierarchy structure containing a high-level RL with an Entropy Bonus to control the portfolio weights and a low-level RL to control selling price and quantities within one day. Fang et al. [23] implement an Oracle by distilling actions trained from perfect future stock information (Policy Distillation) [24] to guide the agent making decisions under imperfect previous stock information. For model-based approaches, Yu et al. [25] incorporate an Infused Prediction Module (IPM) into the original actor-critic style DDPG algorithm so the transition states can be predicted by IPM. All the proposed RL algorithms claim to be profitable and outperform classical algorithms in terms of the Sharpe value and geometric mean return. Importantly, the input states of these RL algorithms are limited to the open, close, high, and low price vectors.

Risk-aware DRL in Stock Trading

Apart from considering the expected accumulated discounted reward as the objective function, there is another research track in the Reinforcement Learning community that uses Conditional Value at Risk (CVaR/VaR) as the objective function which emphasizes the AI safety and risk awareness. Theoretically, Chow et al. [26] propose the CVaR MDP in which the standard risk-neutral expectation is replaced by a risk-sensitive Conditional-Value-at-Risk (CVaR) objective. Stanko and Macek [27] introduce CVaR Q-learning, a sampling version of CVaR Value Iteration [26] based on the distributional

policy improvement algorithm. In the financial market, modern portfolio theory (MPT), or mean-variance analysis [28], Value-at-Risk (VaR) [29] and Conditional-Value-at-Risk (CVaR) [30] are all widely used in risk management models to reduce the maximum possible loss of a financial product under price fluctuations. However, very little research connects these risk models with reinforcement learning and successfully applies it to the financial market.

Function Approximator in Stock Trading

Other literature of stock trading focuses on designing the customized topology of the neural network for financial features. Initially, many researchers use customized neural networks on stock price prediction tasks. For instance, Chen et al. [31] incorporate a graphical convolutional network based on quantitative data, and Ding et al. [32] embed business events according to knowledge graph information to predict stock prices. Later, inspired by the prediction tasks [31, 32], AlphaStock [33] creates a Transformer-based Cross-Asset Attention Network (CAAN) that uses multiple stock features to approximate the functions in the RL algorithms.

3 Problem Statement

In portfolio optimization, we would like to continuously allocating capitals into a number of financial assets with the aim of maximizing the cumulative return. For an automatic trading agent, the process of obtaining daily returns through enhancing or reducing portfolio positions can be seen as a finite Markov Decision Process. This section provides a mathematical setting of the portfolio optimization problem and its connection to Reinforcement Learning.

3.1 Assumptions

In this work, we only consider back-test tradings where the trading agent has no information about the future stock market. The trading agent is assumed to return at a timestamp in the stock market history and carries out paper trading from then onward. To meet the requirement of back-test tradings, we make two assumptions in our experiment:

1. Zero Slippage: The market assets are high in liquidity so that each transaction can be completed immediately after an order is placed.
2. Zero Market Impact: The transaction made by the trading robot is insignificant so it has no influence on the market.

In a realistic trading environment, these two assumptions are valid under the circumstance of the high trading volume in the stock market.

3.2 Mathematical Formalism

To formulate our portfolio model, we modify the settings in Online Portfolio Selection (OLPS) [34]. The portfolio consists of one cash asset and m

stock assets. The trading time is equally divided into periods of length T and length T equals to one day in this paper. Since it is assumed in the back-test experiments that at the beginning period of $t+1$, assets can be immediately traded at the opening price of the period of $t+1$, we are allowed to use the closing price v of period t to complete the transaction. More specifically, for a portfolio vector $\mathbf{w}_t = [w_{0,t}, w_{1,t}, ..., w_{m,t}]^T$, where the first element is the weight of the cash and the other i^{th} element represents the proportion of total capital invested in the i^{th} stock at period t. We derive its price relative vector $\mathbf{v}_t = [v_{0,t}, v_{1,t}, ..., v_{m,t}]^T$ based on the ratio of t^{th} closing price to the last closing price for the i^{th} asset. Based on $\mathbf{w}_t$ and $\mathbf{v}_t$, the final cumulative wealth after n periods is $p_f = p_0 \Pi_{t=1}^{n} \mathbf{w}_t^T \mathbf{v}_t$ where p_0 is the initial investment, $\mathbf{w}_1 = [1, 0, ..., 0]^T$ and $\Sigma_{i=0}^{m} w_{i,t} = 1$. The t^{th}-step exponential growth rate r_t is given by $r_t = \log(\mathbf{w}_t^T \mathbf{v}_t)$.

Since the transaction cost is indispensable, OLPS [34] adopts the proportional transaction model [35, 36], i.e., the incurred transaction cost is proportional to the wealth transfer when reallocating $\mathbf{w}_t$. Specifically, it introduces a transaction cost factor [36] μ_t, which is the ratio of total wealth after reallocating to wealth before reallocating, and the one-step exponential growth rate r_t can be rewritten as $r_t = \log(\mu_t \mathbf{w}_t^T \mathbf{v}_t)$, where $\frac{1-\gamma_s}{1+\gamma_b} \leq \mu_t \leq 1$, γ_s and γ_b are the commission fees of selling and buying stocks. When $\gamma_s = \gamma_b = \gamma$, Moody et al. [37] give an approximation to μ_t, i.e.,

$$\mu_t = 1 - \gamma \sum_{i=1}^{m} |w'_{i,t} - w_{i,t}| \tag{1}$$

where $w'_{i,t} = \frac{w_{i,t-1} \cdot v_{i,t}}{\Sigma_{j=0}^{m} w_{j,t-1} \cdot v_{j,t}}$ represents the adjusted portfolio weights due to the change in the stock price at time t.

In our work, we adopt the exponential growth rate r_t with OLPS's transaction cost, and use (1) to approximate it. Importantly, to complete our final one-step reward, we introduce an additional risk term and reformulate r_t as

$$r_t = \log(\mu_t \mathbf{w}_t^T \mathbf{v}_t) - \beta \operatorname{Var}(r) \tag{2}$$

where β is a reward-risk adjust hyperparameter and $\operatorname{Var}(r)$ is computed through the variance of all previous $r_{1:t}$.

4 Methodology

In this section, we first give a short description of the basic concepts in reinforcement learning, including value functions and loss functions, that are fundamental to subsequent algorithms proposed in Section 4.1. We then detail the novel architecture and description of the proposed reinforcement learning algorithm in Section 4.2.

4.1 Reinforcement Learning: A Short Description of Main Concepts

As demonstrated in [38], reinforcement learning and control problems usually include an agent that acts in a stochastic environment by sequentially selecting actions over a sequence of time steps to maximize a cumulative reward. Generally, these problems can be formalized as discrete time stochastic Markov Decision Processes where an agent interacts with its surrounding environment in the following way: given a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where

- $\mathcal{S}$ is a (finite) set of Markov states $s \in \mathcal{S}$.
- $\mathcal{A}$ is a (finite) set of actions $a \in \mathcal{A}$.
- $\mathcal{P}$ is dynamics (model-free) or an explicit transition model (model-based) for each action. For an explicit transition model satisfying the Markov property, it can be specified as $\mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a)$.
- $\mathcal{R}$ is an expected reward function under policy π and defined as $\mathcal{R}(s_t = s, a_t = a) \triangleq \mathbb{E}_\pi[r_t \mid s_t = s, a_t = a]$.
- $\gamma \in [0, 1]$ is a future discount factor.

For one single episode, the agent starts in a given state $s_0 \in \mathcal{S}$. At each time step t, it chooses an action $a_t \in \mathcal{A}$ based on a policy π and receives an immediate one-step reward r_t. It then keeps updating until it reaches a terminal state. All in all, our ultimate goal is to control an optimized policy π that can generate an optimal return at each state s. More details for the related definitions can be found in [38].

4.2 Proposed RL Framework: From Deterministic to Stochastic

In this section, we first construct the deterministic actor-critic settings in Section 4.2.1. This is followed by a detailed description of how Stochastic Policy with Distributional Q-network (*SPDQ*) is implemented in Section 4.2.2.

4.2.1 Deterministic Framework

Adapted from Q-learning, rather than globally maximizing Q, Silver et al. [13] utilize Deterministic Policy Gradient (DPG) to obtain the maximum return in a continuous action space through iteratively moving in the gradient direction of Q. In practice, for a deterministic policy μ_θ, its policy parameters θ^{k+1} are learned by gradient ascent $\nabla_\theta Q^{\mu_k}(s, \pi_{\mu_{\theta_k}}(s))$. Similar to the actor-critic style algorithm, the critic in the DPG algorithm is learned by minimizing the Bellman error. Importantly, DPG [13] lays the theoretical foundation for Deep DPG [14].

In DDPG, Lillicrap et al. [14] incorporate deep neural network function approximators into DPG. In other words, for target value function (network) $Q^{w'}(s, a)$ and learned value function (network) $Q^w(s, a)$, DDPG introduces a method to slowly update the target network from a parameterized network

rather than directly copying weights w to the targets. Practically, the weights of target network are slowly tracked by trained networks: $w' = \tau w + (1-\tau)w'$ with $\tau \ll 1$. In this way, the target value functions can only update slowly, greatly enhancing the stability of learning. Additionally, in order to consistently train the critic without divergence, DDPG requires another target policy function $\mu_{\theta'}$ that is also slowly updated by learned policy function μ_θ in the same manner of target value functions. Another contribution of DDPG is that it introduces a Gaussian Noise Process $\mathcal{N}$ added to the continuous action spaces to encourage exploration. Generally, acting based on a deterministic policy may not ensure adequate exploration and may result in sub-optimal solutions, especially in a highly volatile financial environment.

4.2.2 Stochastic Framework

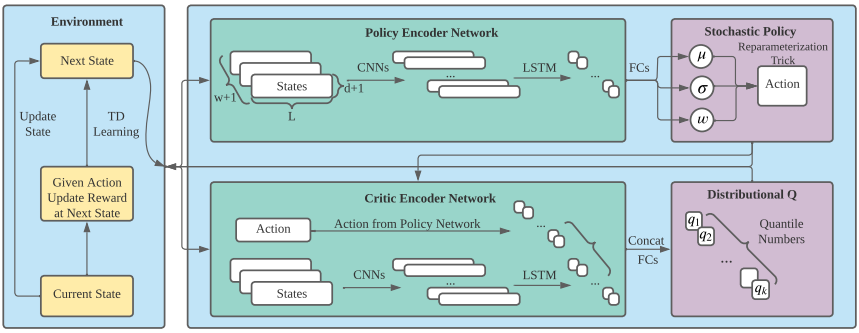


Fig. 1 Proposed Stochastic RL Framework. Here, in the policy network, the coupled states $\mathbf{s}_t \in \mathbb{R}^{l\times(m+1)\times(d+1)}$ are considered as input states to be fed into a policy encoder network (upper green block). Later, the encoded state vector is dropped into the FC layers (upper purple block) to generate means, standard deviation and mixture weights of the output action $\mathbf{a}_t \in \mathbb{R}^{d+1}$. The sampled actions $\mathbf{a}_t$ are realized by performing a reparameterization trick. Subsequently, the new generated action $\mathbf{a}_t$ is simultaneously fed into the critic encoder network (lower green block) and the financial environment (left blue block), with which it can interact, to obtain reward $\mathbf{r}_t$ and generate a new state $\mathbf{s}_{t+1}$. In the critic network, the encoded layer (lower green block) input by $\mathbf{s_t}$ and $\mathbf{a}_t$ is then fed into the FC layers (lower purple block) to generate the quantile numbers of the value distribution $\mathbf{Q}_t$. Finally, after estimating $\mathbf{Q_t}$ and $\mathbf{Q_{t+1}}$, value distribution is learned by using temporal difference.

Formally, we model the portfolio optimization problem with a trading cost as one Markov Decision Process $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma)$ without considering the transition probabilities. In practice, the time horizon for this MDP is set to be the total holding time of the portfolio until the portfolio value p_f reaches zero. At the beginning time of t, the trading agent generates a new portfolio weight and reallocates money to particular financial assets according to that weight. Here, we define the coupled states at time t as $s_t := \{\mathbf{X}_t, \mathbf{W}_t\} \in \mathcal{S}$, where $\mathbf{X}_t$ is the historical stock features and $\mathbf{W}_t$ is the historical weights. In other words, we

consider the previous weights of the portfolio to be also a part of state and concatenate them with the previous states along the dimension of feature. For the non-cash assets, the j^{th} feature $\mathbf{X}_{t,i,j}$ for assets i at time t are built by a look-back time window with length l, i.e., $\mathbf{X}_{t,i,j} = \{x_{t-l,i,j}, x_{t-l+1,i,j}, \ldots, x_{t-1,i,j}\}$, in which $x_{t-1,i,j}$ represents the basic information of feature j of asset i at time of $t-1$. For cash, $\mathbf{X}_{t,0,j}$ is made up of unit vectors and collectively gives the same shape as $\mathbf{X}_{t,i,j}$. In this setting, we have $m+1$ assets (counting cash) and $d+1$ features (previous assets weights as a new feature). This gives us a coupled state $\mathbf{s}_t \in \mathbb{R}^{l\times(m+1)\times(d+1)}$. At the beginning holding period t, based on the inputted $\mathbf{s}_t$, the trading agent will generate a new continuous action defined as $\mathbf{a}_t := \mathbf{w}_t \in \mathcal{A}$ to redistribute the fund among the assets. Here $\mathbf{w}_t \in \mathbb{R}^{m+1}$ and satisfies $\sum_m \mathbf{w}_t = 1$. For each state-action pair $(\mathbf{s}_t, \mathbf{a}_t)$ at holding period t, its reward $r_t \in \mathcal{R}$ satisfies (2).

Maximum Entropy Reinforcement Learning

Instead of using the standard cumulative returns as the reinforcement learning objective, our goal is to learn a stochastic policy $\pi(a_t|s_t)$ that maximizes the new entropy objective $J(\theta)$, i.e., $J(\theta) = \mathbb{E}_{s\backsim\rho^{\pi_\theta}, a\backsim\pi_\theta}[r(s,a) + \alpha\mathcal{H}(\pi_\theta(\cdot|s))]$, where ρ^{π_θ} is the marginal state distribution, $\mathcal{H}(\cdot)$ represents an entropy function which is calculated as $\mathcal{H}(\pi_\theta(\cdot|s)) = \mathbb{E}_{a\backsim\pi_\theta}[-\log(\pi_\theta(a|s))]$, and α stands for the temperature parameter that weighs the importance of the reward against the entropy term.

To model the diverse modality of our portfolio policy under different states, we suppose that the output of our policy network $\Theta(s_t) := \{\mu_\theta(s_t), \sigma_\theta(s_t)\}$, follows a Mixture Model with K multivariate gaussian components $(N_i(\mu_i, \Sigma_i), i = 1, 2, ..., K)$. Based on the formulation, we sample an action $a'_t \in \mathcal{A}'$ from this policy network by performing the reparameterization trick. Thus, the probability density function of $\mathcal{A}'$ is given by $p_{\mathcal{A}'}(a') = \sum_{i=1}^{K} \omega_i N_i(a';\ \mu_i, \Sigma_i)$, where $\sum_{i=1}^{K} \omega_i = 1$.

Subsequently, we introduce a map $f : \mathcal{A}' \to \mathcal{A}$ to map the original random variable $\mathcal{A}'$ to a simplex region that satisfies the properties in $\mathcal{A}$. The function f is written as

$$f(a'_i, \tau) = \frac{\exp(a'_i/\tau)}{\sum_{j=1}^{h} \exp(a'_j/\tau) + \delta}, \tag{3}$$

where $\tau \in (0, \infty)$ is the temperature parameter that controls the weight distribution in different assets, and $\delta \approx 10^{-9}$ is a small number to ensure that the map f is invertible.

Consequently, the density function of $\mathcal{A}$ after transformation is represented by $p_{\mathcal{A}}(a) = p_{\mathcal{A}'}(a')|\det J_f(a', \tau)|^{-1}$, where $J_f(\cdot, \tau)$ is the Jacobian of $f(\cdot, \tau)$. Finally, the log-likelihood of action $\pi(a_t|s_t)$ (entropy term) can be expressed as

$$\begin{aligned}\log \pi_\theta(a_t|s_t) &= \log p_{\mathcal{A}'}(a'_t) - \log|\det J_f(a'_t, \tau)| \\ &\gg \log p_{\mathcal{A}'}(a'_t) + h\log(\tau) - \sum_i^h \log(a_t^i)\end{aligned} \tag{4}$$

where a_t^i represents the weight of the i^{th} asset, the inequality part comes together from the Matrix Determinant Lemma [39] and proper scaling. Thus, it gives us a lower bound of $\log \pi_\theta(a_t|s_t)$ to simplify minimizing the log-likelihood itself. Detailed proof of this inequality is shown in Appendix A.

Finally, the policy parameters θ can be learned by minimizing the following equation from [17], i.e.,

$$L_\pi(\theta) = \mathbb{E}_{s_t \backsim \mathcal{D}, a_t \backsim \pi_\theta}[\alpha \log(\pi_\theta(a_t|s_t)) - Q^w(s_t, a_t)] \tag{5}$$

where $\mathcal{D}$ stands for the experience replay buffer. Importantly, in (4), we derive a lower bound for $\log \pi_\theta(a_t|s_t)$. Thus, to avoid gradient explosion during training, we introduce the lower bound to Eq. (5) and only minimize the lower bound of $L_\pi(\theta)$. By extending the DDPG style policy gradient [14], we can approximate the gradient of policy network (5) with $\nabla_\theta L_\pi(\theta) = (\nabla_{a_t} \alpha \log(\pi_\theta(a_t|s_t)) - \nabla_{a_t} Q(s_t, a_t))\nabla_\theta g_\theta(s_t;\ \epsilon_t)$, where $g_\theta = f \circ \Theta$ and a_t is evaluated at $g_\theta(s_t;\ \epsilon_t)$.

Distributional Value Function

For a policy π, instead of considering the cumulative return observed at each time t (i.e., the sum of discounted rewards G_t observed from one trajectory of states following the policy π) as a constant, we model it as a distribution to reflect the uncertainty of return in the real financial markets and denote it as Z_t^0, where $Z_t^0 = \sum_{i=0}^T \gamma^i R_i$ and R_i represents a random variable of i^{th} step reward. Thus the reward distribution, together with the entropy term, is written as $Z_t^\pi = \sum_{i=0}^T \gamma^i R_i + \alpha \mathcal{H}(\pi)$. Based on this, the action-value function required in (5) is rewritten as

$$Q^\pi(s, a) = \mathbb{E}_{s_i \backsim \mathcal{D}, a_i \backsim \pi}[\sum_{i=t}^T \gamma^{i-t} R(s_i, a_i) + \alpha \mathcal{H}(\pi(\cdot|s_i))].$$

In practice, our approximation to this value distribution aims to model the quantile numbers of the target distribution and we call it a quantile distribution. Accordingly, the output of the critic network is a vector of length N that represents N quantiles and its associated discrete cumulative probabilities are $q_i = \frac{i}{N}$ for $i = 1, \ldots, N$ and $q_0 = 0$.

Formally, let $w : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^N$ be the parametric model of our critic network. A probability quantile distribution $Z^w : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathbb{R})$ maps each state action pair (s, a) to a uniform probability distribution supported in $\{w_i(s, a)\}$. We write $Z^w(s, a) := \frac{1}{N} \sum_{i=1}^N \delta(w_i(s, a))$, where $\delta(z)$ represents the Dirac function at $z \in \mathbb{R}$.

Based on the one-step temporal difference learning, we train our critic network using Quantile Huber Regression [19] that minimizes the distance

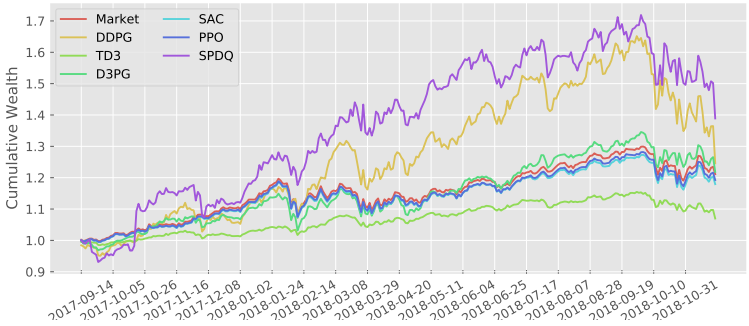


Fig. 2 The Cumulative Wealth in U.S. market on the validation set for different models

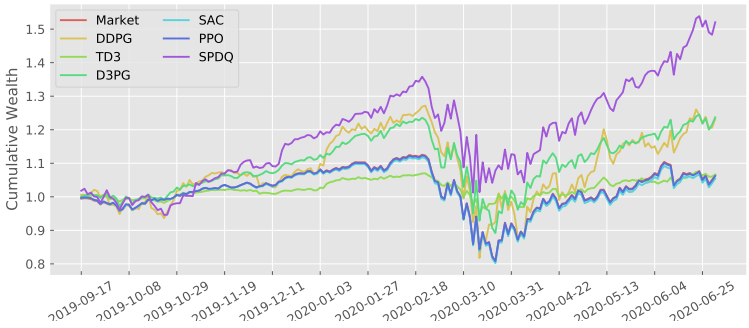


Fig. 3 The Cumulative Wealth in U.S. market on the test set for different models

between the target distribution and the learned distribution. The Quantile Huber Regression Loss in our problem is expressed as

$$L_Z(w) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\backsim\mathcal{D}}[\sum_{i=1}^{N} |q_i - \delta(u_i < 0)|\mathcal{L}_k(u_i)] \tag{6}$$

where $u_i = r(s_t, a_t) + \gamma(w_i'(s_{t+1}, \theta'(s_{t+1})) - \alpha \log(\pi_{\theta'}(\theta'(s_{t+1})|s_{t+1}))) - w_i(s_t, a_t)$, $\mathcal{D}$ is the experience replay buffer, and $\mathcal{L}_k$ is the huber loss.

Stochastic Policy with Distributional Q-network (SPDQ)

At each time step, for any coupled state $\mathbf{s}_t = \{\mathbf{X}_t, \mathbf{W}_t\}$ of both policy network and critic network, where $\mathbf{X}_t \in \mathbb{R}^{l\times(m+1)\times d}$ is the historical stock features and $\mathbf{W}_t \in \mathbb{R}^{l\times(m+1)}$ is the previous assets weights, SPDQ encodes the coupled state by letting the dimension of features in $\mathbf{s}_t$ be the channel dimension and feeding $\mathbf{s}_t$ into the *Conv2D* layers with the *same* padding scheme on time dimension and the *valid* padding scheme on assets dimension. Subsequently,

SPDQ merges the 22 assets dimensions into one single dimension while maintaining the length of time horizon simultaneously. Its outputs are followed by an LSTM Layer acting temporally to resolve the complexity between the long-range time horizon. To output a stochastic policy, the encoded state vector is directly fed into the Fully Connected Layer to generate means, standard deviations and mixture weights of the output action. For the critic network, the encoded state vector is concatenated together with the predicted action from the policy network, and fed into the FC Layers to output quantile numbers. The overall SPDQ reinforcement learning framework is shown in Figure 1.

Practically, we initialize two neural networks for both the policy and critic networks: θ as the learned policy network, θ' as the target policy network, w as the learned critic network, and w' as the target critic network. Given an input state $\mathbf{s}_t$, we first use the policy network θ to generate a new portfolio weight $\mathbf{a}_t$, and use it to interact with the financial environment, obtaining a new reward $\mathbf{r}_t$ and a new state $\mathbf{s}_{t+1}$, and forming a one-step trajectory $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1})$. Subsequently, we store the trajectory in a Prioritized Experience Replay Buffer [40] and do not start training until the number of samples in the replay buffer reaches the batch learning size. During training, for sampled one step trajectory $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1})$, we update the policy parameters by maximizing the entropy in (5) and update critic parameter by minimizing the temporal difference in (6). For the temperature hyperparameter α to control the importance of entropy term, we update it by minimizing the temperature loss in (7). The loss of α is derived in [17], namely

$$L(\alpha) = \mathbb{E}_{(s_t,a_t)\backsim\mathcal{D}}[-\alpha \log \pi_\theta(a_t|s_t) - \alpha\mathcal{H}_0] \tag{7}$$

where $\mathcal{H}_0$ is the minimum value of entropy.

After updating the gradient of the learned network, the weights of the target network are slowly adjusted by the trained networks in a similar manner as DDPG. Importantly, to reduce per-update error caused by noisy input, we choose to update the policy network and its target network less frequently than the critic network. The detailed algorithm for SPDQ is summarized in Algorithm 1.

5 Experiment

In this section, we first introduce our data processing techniques and performance metrics. Next, we benchmark the deterministic algorithms of DDPG [14] and TD3 [15], and the stochastic algorithms of Distributional Deterministic Policy Gradient (D3PG) [19], Proximal Policy Optimization (PPO) [21], and SAC [17], in the provided U.S. stock market. Subsequently, we evaluate the proposed stochastic algorithms with the listed baseline algorithms. Finally, we display the interpretation of the model strategy, and investigate the impact of different hyperparameter choices, using an ablation study.

Algorithm 1 SPDQ for Portfolio Optimization

input: batch size M, iterations T, target entropy $\mathcal{H}_0$, discount factor γ.
Initialize policy network parameters θ, value network parameters w, and α.
Initialize target network parameters $\theta' \leftarrow \theta$, $w' \leftarrow w$
while not converge **do**
  Observe state s_i and sample action $a'_i \sim \pi_\theta(\cdot|s_i)$
  Map a'_i to simplex region a_i ▷ Eq. (3)
  Relax a_i's log-likelihood $l_i \propto \log \pi_\theta(a_i|s_i)$ ▷ Eq. (4)
  Observe next state s'_i, reward r_i and done signal d_i
  Store $(s_i, a_i, l_i, r_i, s'_i, d_i)$ in replay buffer $\mathcal{D}$
  if `curr_buffer_size` $\geq M$ **then**
    for t from 1 to T **do**
      Sample M trajectories (s, a, l, r, s', d)
      Construct target distribution: $y = r + \gamma(Z^{w'}(s, \pi_{\theta'}(s')) - \alpha \cdot l)$
      Update w, θ, α by SGD using $\Delta_w, \Delta_\theta, \Delta_\alpha$, where:
      $\Delta_w = \frac{1}{M}\sum_i \nabla_w \mathbb{E}[|q - \delta(u_i < 0)|\mathcal{L}_k(u_i)]$, $u = (Z^w(s, a) - y)$
      $\Delta_\theta = \frac{1}{M}\sum_i \nabla_\theta \pi_\theta(s_i)\nabla_a[\log \pi_\theta(a|s_i) - \mathbb{E}[Z^w(s_i, a)]]|_{a\sim\pi_\theta(s_i)}$
      $\Delta_\alpha = \frac{1}{M}\sum_i \nabla_\alpha[-\alpha(l_i + \mathcal{H}_0)]$
      Update target network parameters w' and θ'
      $w' \leftarrow \rho w' + (1 - \rho)w$
      if t mod `policy_update` $== 0$ **then**
        $\theta' \leftarrow \rho\theta' + (1 - \rho)\theta$
      end if
    end for
  end if
end while

5.1 Dataset setting and Preprosessing

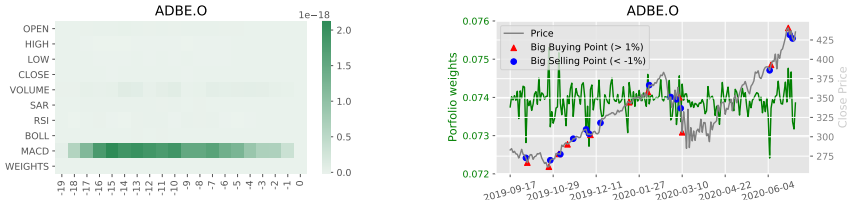
The U.S. stock market data used in our experiments are obtained from Wind[1]. The time range of the data is from January 2005 to December 2020. This long time interval covers several well-known market events, such as the crash of 2008-2009 caused by the subprime mortgage crisis [41] and the 'meltdown' in 2020 caused by COVID-19 [42], which diversifies the market states and enables our trading agent to learn from real-world data fluctuations. Each collected stock contains nine different features ranging from the fundamental indexes like OPEN, CLOSE, LOW, HIGH to the technical indexes like BOLL and MACD. Concretely, 22 stocks[2] are chosen from S&P500 in the top 50 of the index's component with large volumes, so our trading algorithms would not influence the market price. Detailed information related to stock names and feature names can be found in Supplementary Table B1 and B2. In addition, we introduce one cash asset as a risk-free option for the trading agent. Moreover, the period of stock data used in the experiments is given in Table 1.

[1] https://www.wind.com.cn/
[2] In our work, we focus on these 22 stocks for ease of explanation. This framework is also applicable to other portfolios.

Importantly, each feature is normalized by the first feature in the look-back window and scaled by a positive factor c.

Table 1 Period of stock data used in the experiments.

	The U.S. Market	Duration (Days)
Training	2005/01/03-2017/06/29	3144
Validating	2017/06/29-2019/07/01	503
Testing	2019/07/01-2020/12/08	365



(a) Integrated Gradients of the input states (b) Execution details

Fig. 4 A case study of the Adobe after training 50 episodes

5.2 Performance Metrics

We use the following performance metrics to evaluate our algorithms:

- *Annual Rate of Return (ARR)* [43] is the annual average return rate, it is defined as
$$ARR = \frac{p_f - p_0}{p_0} \times \frac{T_{year}}{T_{all}}$$
where p_f is the final portfolio value, p_0 is the initial portfolio value, T_{year} represents the total number of trading days within one year, and T_{all} is the total number of trading days.
- *Annualized Volatility (AVOL)* [43] is the annual average volatility to reflect the average risk of a strategy in a year. It is defined as
$$AVOL = \text{Var}\left[\frac{p_t - p_0}{p_0}\right] \times \sqrt{\frac{T_{year}}{T_{all}}}$$
where p_t is the portfolio value at each step.
- *Annualized Sharpe Ratio (ASR)* [43] is the risk-adjusted annual return based on APR and $AVOL$. It is defined as
$$ASR = \frac{ARR}{AVOL}.$$

- *Maximum DrawDown (MDD)* [43] is the maximum loss from a climax to a dip of a portfolio, before a new climax is formed. It reflects the risk of the investment. It is defined as

$$MDD = \max_{t\in(0,T)} \left\{ \frac{\max_{t'\in(0,t)}\{p_{t'}\} - p_t}{\max_{t'\in(0,t)}\{p_{t'}\}} \right\}.$$

- *Downside Deviation Ratio (DDR)* [43] is the risk-adjusted annual return divided by the Downside Deviation which represents the potential loss that may arise from risk as measured against a Minimum Acceptable Return (MAR) such as bank interest. It is defined as

$$DDR = \frac{ARR}{\sqrt{\mathbb{E}[\min\{r_t - MAR, 0\}^2]}}.$$

5.3 Results

Experiment settings

Each algorithm, including benchmarks, is trained in our experiment by interacting with an artificial financial environment for 100 episodes. Each episode randomly chooses a 500 time steps length consecutive holding period within the defined training period in Table 1. After training over one episode, our algorithm is then validated on a validation set with 300 time steps to assess its generalization ability. Our algorithm is implemented by Tensorflow on Python and trained through two RTX 2080 Ti Graphic Cards. The results of the stochastic algorithms are aggregated over an average of five replicates to ensure reliability.

In all experiments, we use a replay buffer with size 5000 and only consider behavior policies that are parameterized by Gaussian mixtures. For all the experimented algorithms we initialize the learning rate for the actor to be 5×10^{-4}. For the critic, we initialize the learning rate to be 5×10^{-3}. We use the exponential decay with a rate of 0.5 for both actor and critic. To optimize α, its learning rate corresponds to 1×10^{-3} and the decay rate equals 0.9. Furthermore, we use a batch size of 64 for all the algorithms. The remaining hyperparameters, including look-back window size, τ, risk control factor β, etc., are fine-tuned on the proposed validation set.

Overall performance

We compare the proposed stochastic reinforcement learning algorithm, Stochastic Policy with Distributional Q-Network (SPDQ), with two popular deterministic algorithms (DDPG and TD3), three classic stochastic algorithms (SAC, D3PG, and PPO), and the standard market. In our experiments, the market value is calculated by consistently holding a uniformly weighted portfolio among these 22 stocks. Figures 2 and 3 give the cumulative wealth of the portfolio versus trading days in the U.S market on the validation and test sets.

Table 2 Performance comparison between different deterministic and stochastic DRL algorithms in the U.S. market. The best results for each metric are highlighted in bold.

Algo. Category	DRL Algorithms	ARR	AVOL	ASR	MDD %	DDR
Deterministic	Market	0.091	0.071	1.28	28.2	0.872
	DDPG [14]	0.287	0.112	2.56	35.7	1.806
	TD3 [15]	0.083	**0.030**	2.72	**11.5**	1.747
Stochastic	PPO [21]	0.082	0.071	1.16	28.1	0.782
	SAC (SP) [17]	0.072	0.070	1.03	28.3	0.689
	D3PG (DQ) [19]	0.303	0.093	3.25	27.8	2.663
	SPDQ	**0.631**	0.164	**3.86**	25.5	**4.379**

Specifically, from the plot of market value in Figure 3, we observe this tested period is indeed the crashed period during COVID-19 [42]. It is straightforward that the deterministic algorithm DDPG experiences the most significant decline in March, although it outperforms the market. Surprisingly, another deterministic algorithm, TD3, has a poorer performance and falls far behind DDPG in both validation and test sets. Notably, among the tested stochastic algorithms, D3PG, which has a distributed critic, is superior to the market while PPO and SAC, which have a stochastic actor, nearly share the same cumulative wealth with the market. Importantly, it is worth pointing out that the proposed SPDQ consistently beats the market and has the fastest recovery rate after the 'meltdown' in March. Consequently, by comparing SPDQ with SAC, which only uses a stochastic policy, and D3PG, which only uses a distributional Q-function, we observe that the stochastic policy and the distributional Q-function jointly contribute to the final performance.

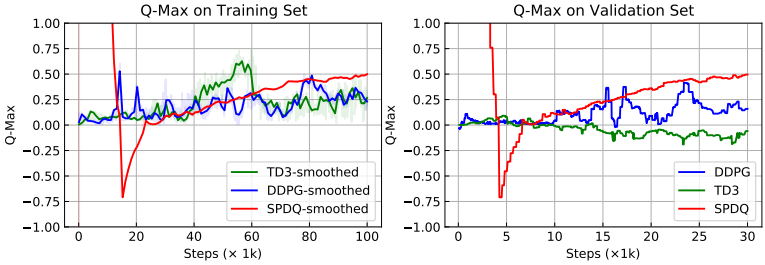
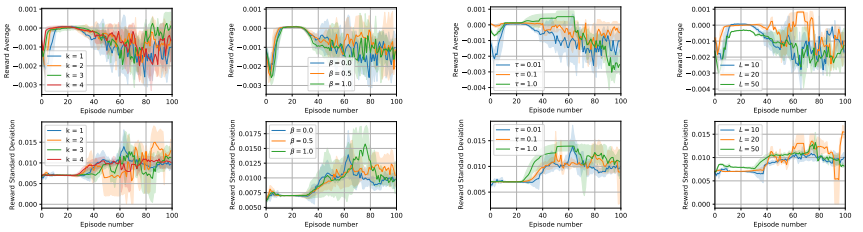


Fig. 5 Learning curves for Q-Max. Q-Max for TD3 and DDPG is calculated by maximizing Q-value in each batch. Q-max for SPDQ is calculated by maximizing the average of all the quantiles in each batch.

Additionally, we evaluate the metrics of different algorithms in Table 2. From the table, we can observe that SPDQ has the best record on three risk-adjusted metrics. Specifically, SPDQ gives 63.1% ARR, up from D3PG 108% ranked in second. It also outperforms the market value in MDD by 10% (25.5% versus 28.2%). Interestingly, TD3 has the lowest *AVOL* and *MDD*. However, it comes at the expense of gaining cumulative returns. Specifically, it

underperforms while competing with the market in ARR (8.3% versus 9.1%). Consequently, we conclude that the proposed stochastic framework attempts to maximize the cumulative returns at the cost of slightly increasing the volatility.



(a) Mixture Numbers (b) Beta (c) Tau (d) Window Length

Fig. 6 Reward learning curves on validation sets for different parameters

Learning analysis

The learning curves on training and validation sets are given in Figure 5. Intuitively, we observe that SPDQ has a better convergence property for the Q-Max that approximates the cumulative return on both training and validation sets. It also has a good generalization ability on the validation set. Moreover, SPDQ begins to level off after about 80 thousand training steps. On the contrary, the deterministic algorithms DDPG and TD3 fluctuate a lot during training, although the overall trend is increasing. Importantly, there is a gap between TD3's training and validation curves, indicating that it may suffer from poor performance regarding unseen data. In summary, SPDQ evidently outperforms TD3 and DDPG on both the training and validation sets, which is consistent with the performance results on the test set.

Trading Strategy Interpretation

Here, we attempt to investigate the action patterns for different strategies, i.e., how the distribution of each asset changes over time. We discover that for the tested deterministic algorithms, especially DDPG, the weights directly converge to several assets within ten episodes, after which no further big changes were observed. In other words, the weights of each asset will fluctuate above or below a fixed mean that is invariant to time. The proposed stochastic algorithm, however, behaves more diversely than the deterministic ones. We observe the change of the trading strategy of the proposed SPDQ contains mainly three steps. It first uniformly distributes weights into 22 assets, then focuses on the several assets by putting more weights on them. Finally, after training for extended episodes, it converges to the assets that are assigned more weights to previously. Notably, the critical reason for the proposed stochastic algorithm to perform better is that our trading agent excels at selecting

profitable long-term assets portfolio, and it chooses to consistently hold them instead of selling and buying shares frequently at every time step.

In addition, we create attribution maps of the input financial features and interpret the long/short actions by using the gradient-based methods in Integrated Gradient (IG) [44] and Alphastock [33], which help us quantify and visualize the critical features valued the most by our trained model. Specifically, we aggregate the values derived from the Integrated Gradients of the inputted states s_t during all of the test time, and visualize them using a heatmap. Concretely, we pick ADBE.O, which shares the highest weights after training over 50 episodes, as a case study in Figure 4(a). Among the inputted nine features, MACD has the highest score (positive gradients) during the last 15 to 10 days. Since the objective function of the policy is to the value function, positive gradients of MACD indicate that if a stock's MACD keeps increasing in the last 15 to 10 days, the value function will also increase the next day. Consequently, our model considers MACD as a signal of future growth of the stock price and thus puts more weights on this asset. Figure 4(b) details how the proposed model executes orders. The selling or buying point is highlighted if the turnover rate is larger than one percent. We observe that its weights fluctuate around 0.074 which is nearly two times bigger than the average weights 0.043. This finding suggests that our proposed algorithm will attach more weights to those profitable assets instead of investing in all the assets averagely.

5.4 Ablation Study

Table 3 Ablation on the effect of Gaussian Mixtures in the U.S. market. The best results for each metric are highlighted in bold.

	ARR	AVOL	ASR	MDD %	DDR
$k = 1$	0.181	**0.071**	2.530	**11.93**	1.735
$k = 2$	0.004	0.077	0.049	28.69	0.037
$k = 3$	**0.631**	0.164	**3.86**	25.46	**4.379**
$k = 4$	0.422	0.121	3.483	12.28	3.715
Market	0.083	0.154	1.170	28.17	0.093

In our ablation study, we study the impact of changing the mixture numbers on the model's final performance. As Table 3 demonstrates, when the policy is parameterized by a unimodal Gaussian ($k = 1$), SPDQ slightly outperforms the market on *ASR* and has the most stable return since it reaches the lowest *AVOL* among the other three options. When we start to increase its modality ($k > 1$), we find it may not lead to better performance and even gives worse results when k equals four. Interestingly, a bimodal Gaussian parameterization leads to a deficit portfolio and the greatest drawdown. However, a trimodal Gaussian parameterization produces a strategy that smoothly balances the return and risk.

Furthermore, Figure 6 provides a comprehensive ablation study of the impact of the mixture numbers, reward-risk adjust factor β, temperature factor τ in (3), and Length of look-back window on the validation reward average and standard deviation. According to Figure 6(a), we verify that the trimodal Gaussian parameterization (green curve) has relatively higher average rewards on the validation set while at the same time possessing a lower rewards standard deviation. Additionally, in Figure 6(b), when the reward-risk adjust factor β equals 0.5, the rewards standard deviation is lower than that of adding no risk control until the training episodes are over 60. Nevertheless, the standard deviation becomes even higher if β keeps increasing to 1. Moreover, the original Softmax activation function with temperature $\tau = 1$ has a smooth growing start of the average rewards in Figure 6(c). However, it falls dramatically after training over 60 episodes. On the contrary, $\tau = 0.1$ generates a more stable training process. Figure 6(d) demonstrates that a longer length of look-back window will not necessarily lead to better performance on the validation set. Instead, without compromising and taking too much risk, $L = 20$ gives the highest average rewards.

6 Conclusions

In this paper, we research on the continuous portfolio optimization with trading costs via deep reinforcement learning. We benchmark several classic deterministic and stochastic reinforcement learning algorithms on our artificial financial environment. Next, we propose a novel interpretable stochastic reinforcement learning framework for the portfolio optimization problem. Concretely, we build a stochastic policy parameterized by Gaussian Mixtures and a distributional critic realized by quantile numbers to interact with the noisy financial market. Finally, the extensive experiments demonstrate that our proposed stochastic algorithm outperforms its deterministic counterparts in terms of controlling risk and gaining profit in the U.S. stock market.

In the future, this research can be extended to the following aspects. First, incorporating Conditional-Value-at-Risk (CVaR) to the existing reinforcement learning framework and applying it to the actual financial market is a promising research direction since CVaR has a superior ability over the mean-variance settings to safeguard a decision-maker from risky movements. Second, investigating the customized exploration functions for the trading agents in reinforcement learning is very important and has the potential to outperform the strategy of exploring blindly based on the Gaussian distributions.

Declarations

Competing Interests

The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] Haugen, R.A.: Modern Investment Theory. Prentice Hall, Hoboken, New Jersey (1986)

[2] Heaton, J.B., Polson, N.G., Witte, J.H.: Deep learning for finance: deep portfolios. Applied Stochastic Models in Business and Industry **33**(1), 3–12 (2016)

[3] Niaki, S.T.A., Hoseinzade, S.: Forecasting s&p 500 index using artificial neural networks and design of experiments. Journal of Industrial Engineering International **9**(1), 1–9 (2013)

[4] Freitas, F.D., Souza, A.F.D., de Almeida, A.R.: Prediction-based portfolio optimization model using neural networks. Neurocomputing **72**(10), 2155–2170 (2009)

[5] Fama, E.: Efficient capital markets: A review of theory and empirical work. Journal of Finance **25**(2), 383–417 (1970). https://doi.org/10.2307/2325486

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236

[7] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Driessche, G.V.D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Marc Lanctot, e.a.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)

[8] Neftci, E.O., Averbeck, B.B.: Reinforcement learning in artificial and biological systems. Nature Machine Intelligence **1** (2019). https://doi.org/10.1038/s42256-019-0025-4

[9] Lucarelli, G., Borrotti, M.: A deep q-learning portfolio management framework for the cryptocurrency market. Neural Computing and Applications **32**, 17229–17244 (2020). https://doi.org/10.1007/s00521-020-05359-8

[10] Hasselt, H.v., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. AAAI'16, pp. 2094–2100. AAAI Press, Phoenix, Arizona (2016)

[11] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., Freitas, N.: Dueling network architectures for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1995–2003 (2016). PMLR

[12] Jiang, Z., Xu, D., Liang, J.: A deep reinforcement learning framework for the financial portfolio management problem. arXiv preprint arXiv:1706.10059 (2017)

[13] Silver, D., Lever, G., Heess, N., Thomas Degris, D.W., Riedmiller, M.: Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on Machine Learning (ICML-14), pp. 387–395 (2014)

[14] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (2019)

[15] Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. CoRR **abs/1802.09477** (2018) https://arxiv.org/abs/1802.09477

[16] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning, pp. 1861–1870 (2018). PMLR

[17] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al.: Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905 (2018)

[18] Bellemare, M.G., Dabney, W., Munos, R.: A distributional perspective on reinforcement learning. CoRR **abs/1707.06887** (2017) https://arxiv.org/abs/1707.06887

[19] Dabney, W., Rowland, M., Bellemare, M.G., Munos, R.: Distributional reinforcement learning with quantile regression. CoRR **abs/1710.10044** (2017) https://arxiv.org/abs/1710.10044

[20] Liang, Z., Chen, H., Zhu, J., Jiang, K., Li, Y.: Adversarial deep reinforcement learning in portfolio management. arXiv preprint arXiv:1808.09940 (2018)

[21] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017) https://arxiv.org/abs/1707.06347

[22] Wang, R., Wei, H., An, B., Feng, Z., Yao, J.: Deep stock trading: A hierarchical reinforcement learning framework for portfolio optimization and order execution. arXiv preprint arXiv:2012.12620 (2021)

[23] Fang, Y., Ren, K., Liu, W., Zhou, D., Zhang, W., Bian, J., Yu, Y., Liu, T.-Y.: Universal trading for order execution with oracle policy distillation. arXiv preprint arXiv:2103.10860 (2021)

[24] Rusu, A.A., Colmenarejo, S.G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., Hadsell, R.: Policy distillation. arXiv preprint arXiv:1511.06295 (2015)

[25] Yu, P., Lee, J.S., Kulyatin, I., Shi, Z., Dasgupta, S.: Model-based deep reinforcement learning for dynamic portfolio optimization. CoRR **abs/1901.08740** (2019) https://arxiv.org/abs/1901.08740

[26] Chow, Y., Tamar, A., Mannor, S., Pavone, M.: Risk-sensitive and robust decision-making: a cvar optimization approach. arXiv preprint arXiv:1506.02188 (2015)

[27] Stanko, S., Macek, K.: Risk-averse distributional reinforcement learning: A cvar optimization approach. In: IJCCI, pp. 412–423 (2019)

[28] Markowitz, H.M.: Portfolio Selection. Yale university press, London (1968)

[29] Longerstaey, J., Spencer, M.: Riskmetricstm—technical document. Morgan Guaranty Trust Company of New York: New York **51**, 54 (1996)

[30] Rockafellar, R.T., Uryasev, S., *et al.*: Optimization of conditional value-at-risk. Journal of risk **2**, 21–42 (2000)

[31] Chen, Y., Wei, Z., Huang, X.: Incorporating corporation relationship via graph convolutional neural networks for stock price prediction. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management. CIKM '18, pp. 1655–1658. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3269206.3269269. https://doi.org/10.1145/3269206.3269269

[32] Ding, X., Zhang, Y., Liu, T., Duan, J.: Knowledge-driven event embedding for stock prediction. In: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pp. 2133–2142. The COLING 2016 Organizing Committee, Osaka, Japan (2016). https://www.aclweb.org/anthology/C16-1201

[33] Wang, J., Zhang, Y., Tang, K., Wu, J., Xiong, Z.: Alphastock: A buying-winners-and-selling-losers investment strategy using interpretable deep

reinforcement attention networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1900–1908 (2019)

[34] Li, B., Hoi, S.C.H.: Online portfolio selection: A survey. ACM Comput. Surv. **46**(3) (2014). https://doi.org/10.1145/2512962

[35] Blum, A., Kalai, A.: Universal portfolios with and without transaction costs. Machine Learning **35**, 193–205 (1999)

[36] Györfi, L., Vajda, I.: Growth optimal investment with transaction costs. In: Freund, Y., Györfi, L., Turán, G., Zeugmann, T. (eds.) Algorithmic Learning Theory, pp. 108–122. Springer, Berlin, Heidelberg (2008)

[37] Moody, J., Wu, L., Liao, Y., Saffell, M.: Performance functions and reinforcement learning for trading systems and portfolios. Journal of Forecasting **17**, 441–470 (1998)

[38] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)

[39] Harville, D.A.: Matrix algebra from a statistician's perspective. Taylor & Francis (1998)

[40] Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015)

[41] Duca, J.V.: Subprime mortgage crisis. Federal Reserve History **23** (2013)

[42] Organization, W.H., et al.: Naming the coronavirus disease (covid-19) and the virus that causes it. Brazilian Journal of Implantology and Health Sciences **2**(3) (2020)

[43] Financial Terms Dictionary. Investopedia. https://www.investopedia.com/financial-term-dictionary-4769738

[44] Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. ICML'17, pp. 3319–3328. JMLR.org, Sydney, NSW, Australia (2017)

Appendix A Mathematical Details

A.1 Computing the determinant of the Jacobin Matrix

For $f : \mathbb{R}^h \to \mathbb{R}^h$ from the main manuscript (3), where h is the dimension of actions, we let $a = f(x)$, and the Jacobin of this function is:

$$J_f(x,\tau) = \frac{1}{\tau}\begin{pmatrix} a_1 - a_1^2 & -a_1a_2 & \cdots & -a_1a_h \\ -a_2a_1 & a_2 - a_2^2 & \cdots & -a_2a_h \\ \vdots & \vdots & \ddots & \vdots \\ -a_ha_1 & -a_hz_2 & \cdots & a_h - a_h^2 \end{pmatrix},$$

if we define $v = (a_1, a_2, \cdots, a_h)^T$ and $D = diag(a)$, then we have:

$$\begin{aligned} \det(J_f(x,\tau)) &= \det(\frac{1}{\tau}(D - vv^T)) \\ &= (\frac{1}{\tau})^h \cdot (1 - v^T D^{-1} v) \cdot \det D \quad \text{by the Matrix Determinant Lemma} \\ &= (\frac{1}{\tau})^h \cdot (1 - \sum_{i=1}^{h} a_i) \cdot \prod_{i=1}^{h} a_i \quad \text{by the property of matrix } D \\ &= (\frac{1}{\tau})^h \cdot \delta \cdot \prod_{i=1}^{h} a_i \quad \text{by } \sum_{i=1}^{h} a_i \approx 1 \end{aligned}$$

A.2 Computing the lower bound of the log probability

According to the formula of the transformation of random variables, we have $p_{\mathcal{A}}(a) = p_{\mathcal{A}'}(a')|\det J_f(a',\tau)|^{-1}$, if we let $p_{\mathcal{A}}(a) := \pi_\theta(a_t|s_t)$, then its log-likelihood can be written as:

$$\begin{aligned} \log \pi_\theta(a_t|s_t) &= \log p_{\mathcal{A}'}(a_t') - \log|\det J_f(a_t',\tau)| \\ &= \log p_{\mathcal{A}'}(a_t') + h\log(\tau) - \log(1 - \sum_{i}^{h} a_t^i) - \sum_{i}^{h} \log(a_t^i) \\ &= \log p_{\mathcal{A}'}(a_t') + h\log(\tau) - \log(\delta) - \sum_{i}^{h} \log(a_t^i) \\ &\gg \log p_{\mathcal{A}'}(a_t') + h\log(\tau) - \sum_{i}^{h} \log(a_t^i) \end{aligned}$$

Therefore, the lower bound of the transformed log likelihood on a simplex region is $\log p_{\mathcal{A}'}(a_t') + h\log(\tau) - \sum_i^h \log(a_t^i)$.

Appendix B Supplementary Tables

Supplementary Table B1 Abbreviations and Full names of the used 22 stocks in the U.S. market

Abbreviation	Full Name
CMCSA.O	COMCAST CORP.
ADBE.O	ADOBE INC.
GOOGL.O	ALPHABET INC.
AAPL.O	APPLE INC.
BRK_B.N	BERKSHIRE HATHAWAY INC.
T.N	AT&T INC.
PG.N	PROCTER & GAMBLE CO
XOM.N	EXXON MOBIL CORP.
DIS.N	WALT DISNEY CO
UNH.N	UNITEDHEALTH GROUP INC.
JPM.N	JPMORGAN CHASE & CO
CSCO.O	CISCO SYSTEMS, INC.
HD.N	HOME DEPOT INC.
AMZN.O	AMAZON COM INC.
CRM.N	SALESFORCE.COM, INC.
JNJ.N	JOHNSON & JOHNSON
KO.N	COCA COLA CO
NFLX.O	NETFLIX INC.
VZ.N	VERIZON COMMUNICATIONS INC.
MSFT.O	MICROSOFT CORP.
BAC.N	BANK OF AMERICA CORP.
ABT.N	ABBOTT LABOTORIES

Supplementary Table B2 Abbreviations and explanations of the used nine features

Abbreviation	Explanation
OPEN	The first quotation price of one stock after the opening time of a trading day.
CLOSE	The last transacted price of one stock before the closing time of a trading day.
HIGH	The highest transacted price of one stock in a trading day.
LOW	The lowest transacted price of one stock in a trading day.
VOLUME	The transacted volume of one stock in a trading day.
MACD	A momentum indicator which is used to track the trade trends of the stock price.
SAR	An indicator to identify suitable exit and entry points by highlighting the direction that an asset is moving.
RSI	A momentum indicator to measure the speed and magnitude of a changing stock price.
BOLL	An indicator to characterize the prices and volatility.