Expert Systems With Applications 198 (2022) 116807

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Risk-sensitive policies for portfolio management

Mingfu Wang, Hyejin Ku *

Department of Mathematics and Statistics, York University, 4700 Keele Street, Toronto, M3J 1P3, Canada

ARTICLE INFO

Keywords:
Portfolio management
Portfolio optimization
Reinforcement learning
Worst-case scenario
Hierarchical DDPG
Distributional DDPG

ABSTRACT

The decision-making on portfolio investment is fundamental in the financial market, but getting the optimal strategy is challenging due to high uncertainty and massive noise in the market. Deep Deterministic Policy Gradient (DDPG), proposed by Lillicrap et al. (2015), is a deep Reinforcement Learning (RL) algorithm that made remarkable achievements in the financial perspective. Although the applications of RL in financial trading are well-developed, it is surprising that most of the literature ignores the possible risk of rare occurrences of catastrophic events and the effect of the worst-case scenarios on trading decisions. In this paper, we first develop a novel deep RL algorithm, called Hierarchical DDPG, that combines the classical DDPG algorithm and the Hierarchical RL structure to control the risk of portfolio investment. Second, we adapt the distributional DDPG method for portfolio management problems, which aims to maximize the *α-percentile* expectation based on the distribution of future returns. A real world dataset is used to validate the performance of our proposed models. The experimental results show that our proposed models outperform the market and classical DDPG, and moreover, both approaches provide effective methods of constructing a risk-sensitive policy to protect investors from suffering a huge loss.

1. Introduction

Portfolio management is a decision-making process that allocates investment funds to gain maximum profit and relatively lower risk based on individuals' goals, risk preferences, and investment horizons. The foundation of modern portfolio theory can be traced back to the pioneering work of Markowitz (1952), in which his Mean–Variance analysis is a representative methodology in the framework of return–risk trade-off analysis. The original Mean–Variance theory is developed in a mathematical skeleton that constructs a portfolio that maximizes the expected return for a given degree of risk. The disadvantage of the original Mean–variance model is that when a portfolio has high return and volatility, investors might give up the strategy of high returns to remain at low risk. Moreover, there is much noise and uncertainty in the financial market, which leads to inaccurate values of the mean and variance.

Over the last few decades, many studies investigate the application of RL algorithms to financial market trading, and try to predict the price movements or trends by using historical market data. The advantage of RL learning technology in portfolio optimization is that the algorithm can thoroughly learn and extract useful information from market history, without any advanced knowledge and experience in financial markets, and without making any assumptions about the models. The application of RL in portfolio management problems starts from 1998. Moody and Saffell (2001), Moody, Wu, Liao, and Saffell (1998) first propose a recurrent RL algorithm for portfolio optimization problem and construct assets allocation systems. Both of these studies aim to maximize the differential Sharpe ratio, that is, to maximize risk-adjusted returns by considering transaction costs. The disadvantage of using the differential Sharp ratios is that it penalizes returns exceeding a certain value and takes more weight on recent returns. In addition, the differential Sharpe ratio cannot distinguish the potential growth trend of the portfolio. In another attempt by Almahdi and Yang (2017), they extend the recurrent reinforcement learning approach using an adjusted objective function and seek an optimal weight portfolio strategy under the expected maximum drawdown risk measure. However, these existing models have fixed the number of shares for trading. In reality, when the buying or selling signal occurs in the market, it is necessary to determine how many shares to buy or sell. Trading a fixed number of stocks in each transaction does not reflect the real market situation and affects the total profits. With the development of deep RL, deep RL has demonstrated the capability to learn complex policies from many types of environments.

Deep Q-network (DQN) is one of the most popular methods in deep RL. As the approximation of the Q-value function, the neural network can be applied to approximate the reward by taking actions and

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: morgan08@yorku.ca (M. Wang), hku@mathstat.yorku.ca (H. Ku).

https://doi.org/10.1016/j.eswa.2022.116807
Received 22 July 2021; Received in revised form 28 February 2022; Accepted 28 February 2022
Available online 12 March 2022

pursuing policies from a given state. Bertoluzzo and Corazza (2012), Chen and Gao (2019), and Park, Sim, and Choi (2020) apply DQN to portfolio management problems and make remarkable achievements. The advantage of DQN is that it does not require the labeled data that suffer from the constraints and bias of data. It can automatically adapt to the changes in the underlying data distribution, thereby it is a suitable method for the dynamic of the financial market. However, their actions are limited to the discrete action space while the actions are continuous in portfolio management problems. To overcome this issue, DDPG is proposed by Google DeepMind (Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, & Wierstra, 2015), a type of actor–critic based DRL algorithm that supports the continuous action space encountered in portfolio optimization problems. Jiang and Liang (2017) and Xiong, Liu, Zhong, Yang, and Walid (2018) present innovative approaches based on DDPG to solve the trading problem of the optimal trading position at each transaction in stock market and cryptocurrency market. The experimental results of their evaluation take into consideration transaction costs and prove the effectiveness of the algorithms in portfolio management. The advantage of DDPG is that it can deal with the problem of high-dimensional continuous action space well, and its purpose is to learn a policy function directly, instead of approaching the Q-value function.

Hierarchical Reinforcement Learning (HRL) is a promising method that expands the traditional reinforcement learning methods by decomposing the elaborate and intricate problems into sub-problems and effectively solving each sub-problem. The HRL method has some advantages, such as it is easier to be trained, and solving each sub-problem individually will improve its reusability, which will accelerate the learning process. HRL has been devoted to learning these difficult tasks for a long time, the multi-layer strategies are trained to make decisions and control at a higher level of temporal and behavior abstraction (Barto & Mahadevan, 2003; Dayan, 2002; Dietterich, 1998; Nachum, Gu, Lee, & Levine, 2018). In general, by having a hierarchy of policies, only the lower-level policies execute actions to the environment, and the higher-level policies are trained to distribute the sub-tasks to the lower-level policies. Although the applications of deep RL algorithms in the financial market are well studied, most of the previous works only consider maximizing the total profit, and surprisingly, they ignore the impact of possible disasters.

Obtaining risk-sensitive policies in portfolio management problems has been studied for making optimal investment decisions depending on risk preference. Traditional stochastic control algorithms were first used to solve portfolio optimization problems in 1995. A conventional approach utilizes diffusion process models to obtain the optimal trading policies that maximize the expected utility of consumption and/or terminal wealth. As shown by Hansen and Sargent (1995), a recursive formulation of risk sensitivity preserves the tractability of risk-sensitive control theory and produces decision rules with time-invariant risk adjustments for infinite-horizon control problems. The solution of the risk-sensitive control problem is identical to the solution of a particular type of robust control problem. Hansen and Sargent (2001) describe the links between the risk-sensitive control problem and the robust control problem. Risk-sensitive control theory makes decision rules more risk-responsive by including a risk-sensitive constraint parameter to the objective of the decision-maker, which provides equivalent solutions for the multiplier robust control problem. Bielecki and Pliska (1999) propose a novel factor model in which underlying economic variables, defined as ergodic Gaussian processes, affect the mean returns of individual assets. They adopt risk-sensitive control theory to formulate objective function with a risk-sensitive parameter. The approach differs from the traditional stochastic control approach in that the investor's risk aversion is expressed clearly rather than implicitly through a utility function. Solving the dynamic programming equation to maximize the portfolio's long run growth rate adjusted by asymptotic volatility yields the best risk-sensitive policies. Fleming and Sheu (2000, 2002) extend a more general model that is proposed by Bielecki and Pliska (1999). They associate risk-sensitive control problems with the portfolio management problem, and reformulate as infinite time horizon risk-sensitive control problems. They eliminate the assumption that the individual asset and economic factors have independent noise and analyze the portfolio management problem without any constraints. These studies mentioned above have applied risk-sensitive control theory to the dynamic portfolio problem, in which the obtained optimal trading strategies are associated with a risk-sensitive parameter. In our proposed RL algorithms, the ideal risk-sensitive trading strategies are also determined by the risk constraint and risk tolerance. In contrast to the traditional risk-sensitive control theory, our proposed RL algorithms are applicable to systems with continuous state space and entail randomization or noise with any arbitrary distributions, with no prior assumptions required. Furthermore, while the portfolio management problem is a straightforward application of stochastic control theory, it is rarely applied in reality. While there are various probable factors for this lack of application, the computational tractability and statistical challenges connected with model parameter estimation appear to be the most important. In comparison to the traditional risk-sensitive control paradigm, our proposed RL algorithms provide a more practical and tractable way to achieve optimal asset allocation decisions.

In this paper, we aim to construct policies with risk awareness to protect the investor under the worst-case scenarios. Inspired by Dietterich (1998), Nachum et al. (2018), we propose a novel RL algorithm, called Hierarchical DDPG, which combines the classical DDPG algorithm and the Hierarchical structure for portfolio management problems. The original higher-level policy of HRL performs at an abstraction layer and distributes sub-tasks to the lower-level policy, which corresponds directly to the target that the lower-level policy attempts to reach. In our proposed Hierarchical DDPG, the higher-level policy adjusts the lower-level policy's actions to reduce the portfolio risk and operates in the environment. We employ parametric Conditional Value-at-Risk (CVaR) as a metric that measures the portfolio risk. HRL is extended by adding the portfolio risk indicator, so that the agent can implement different trading strategies for different scenarios. More precisely, the lower-level policy of Hierarchical DDPG can be interpreted as a *worker*, aiming to maximize the total profit of the portfolio when the portfolio risk is lower than the CVaR constraints. The higher-level policy of Hierarchical DDPG can be interpreted as a *manager*, whose purpose is to reduce the portfolio risk immediately based on the *worker*'s action when portfolio risk exceeds the investor's tolerance. On the other hand, most existing RL algorithms cannot learn risk-sensitive policies because they only consider maximizing the average and do not penalize the effects of rare occurrences of catastrophic events. Motivated by Barth-Maron, Hoffman, Budden, Dabney, Horgan, Tb, Muldal, Heess, and Lillicrap (2018), Tang, Zhang, and Salakhutdinov (2020), we propose the distributional DDPG model for portfolio management problems with the purpose of seeking a risk-sensitive policy that can map the same state to different actions according to risk preference. We construct the α-*percentile* expectation as our measure, which represents the expected return under the distribution of the α-*percentile* at the bottom of future return. The risk-sensitive policies can be obtained by maximizing the α-*percentile* expectation based on different values of risk parameter α. When α is small, the agent focuses on maximizing the performance of the worst-case scenario.

The main goal of this paper is to construct a risk-sensitive policy to protect investors who may suffer a huge loss due to a financial crisis or rare disaster events. In pursuing this goal, we have made the following contributions. First of all, we design the Hierarchical DDPG algorithm to learn the solution of the portfolio management problem. When the portfolio risk is below the CVaR constraints, the Hierarchical DDPG agent aims to maximize the total profit. But when the portfolio risk exceeds the CVaR constraints, the priority of the Hierarchical DDPG agent is to reduce the portfolio risk immediately, instead of maximizing the total profit. Second, we propose the distributional DDPG method for solving the portfolio optimization problem based on the uncertainty

of future returns. According to the investor's risk preference, the distributional DDPG algorithm can learn a risk-averse policy that yields different actions depending on risk parameters, which is more robust than the other RL algorithms.

The proposed approaches are then validated by a real-world dataset from the U.S stock market.[1] It is well known that the U.S. stock market crashed during the Coronavirus pandemic in 2020, which was one of the most dramatic stock market crashes in history. The circuit breaker mechanism was triggered three times in a month, S&P500 plunged 1019 points, an equivalent of roughly 29%. This provides a good example for verifying our algorithms. The three different comprehensive performance metrics are employed to assess the portfolio performance from different perspectives. Our experimental results show that Hierarchical DDPG is superior in portfolio management to the classical DDPG method because it can significantly reduce or avoid a loss caused by the occurrences of catastrophic events. Also, the results demonstrate that the distributional DDPG agent can provide a risk-averse policy depending on the risk parameter, and the *α-percentile* expectation is well-suited as the criterion of the distributional DDPG, which provides a good distributional critic that can be learned. Via the experimental study, we verify that two proposed algorithms provide promising results, and our approaches are an effective way to protect the investor who may suffer a huge loss due to a financial crisis or rare disaster events.

This paper is organized as follows. Section 2 briefly reviews the related work in the area of portfolio management using RL. Section 3 formulates the portfolio allocation problem. Section 4 introduces the classical DDPG algorithm. Section 5 introduces our proposed novel models. The core innovation of this paper, Hierarchical DDPG and Distributional DDPG for the portfolio management problem are presented in this section. Section 6 displays the experiment results for classical DDPG, Hierarchical DDPG, and Distributional DDPG; and analyzes the obtained results. The final conclusion is presented in Section 7.

2. Related work

In recent decades, portfolio optimization problems in financial trading have attracted much attention. As a primary approach in the field of Artificial Intelligence (AI),[2] deep RL is one of the most popular portfolio management methods in the financial market due to its outstanding performance compared to expert traders. Deep RL has originally been used in applications of video games (Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, Ostrovski, et al., 2015) and chess games (Silver, Hubert, Schrittwieser, Antonoglou, Lai, Guez, Lanctot, Sifre, Kumaran, Graepel, et al., 2018). Ormoneit and Glynn (2002) propose a kernel-based RL method to conquer the issue of instability in RL. Their method aims at learning within the framework of average-cost and applying this method to portfolio management problems. Nevmyvaka, Feng, and Kearns (2006) propose a novel RL algorithm for optimizing transaction execution in the modern financial market by using NASDAQ market high-frequency datasets. Most traders in the real world are dealing with large-scale diversified investment portfolios, but due to time constraints, they cannot deal with individual stock and millisecond data, which makes it necessary to use automatic trading agents. Their experiment results of real-world data on three NASDAQ stocks demonstrate that RL can indeed result in significant improvements.

Deep RL that combines deep learning and RL algorithms can divide into three groups: policy gradient, value-based, and actor–critic. Policy gradient algorithms learn directly the stochastic policy function that maps a state to the probability of each action in action space. Value-based algorithms approximate the Q-value function that represents the expected accumulated rewards by given a state on taking an action and pursuing a policy. The observed information is analyzed through the neural network and output Q-values of each action, then the value-based algorithms rely on the reward function to influence the output of neural networks by backpropagation. The Temporal difference (TD) learning method plays a key role in the actor–critic algorithm that combines the value-based method and the policy-based method. The policy-based network plays as an actor who outputs an action, while the value-based network acts as a critic that appraises the action estimated by the actor-network and generates the TD errors to update the actor and critic network.

DQN is one of the value-based deep learning methods, which updates the Q-value through a neural network instead of updating the Q-table to maximize the cumulative rewards. In the absence of a deterministic strategy, the algorithm will select the action that provides the highest Q-value, and then the Q-value will be updated continuously until it converges to the best action. Bertoluzzo and Corazza (2012) apply DQN to portfolio management problems. The action space is defined as buying, selling or holding. To compare the performance of DQN and Kernel-based RL algorithm, real-world data from three Italian stocks are used to test and validate the performance. The experiment results show the DQN algorithm performs better than the Kernel-based RL algorithm. Chen and Gao (2019) combine the DQN and Deep Recurrent Q-network for portfolio optimization problems and construct a daily stock trading system that can automatically decide to make transactions on each trading day. The Standard & Poor's 500 Index ETF is used to evaluate their trading system, and its daily prices are defined as the state of reinforcement learning in the trading environment. Jeong and Kim (2019) propose an automated system that can predict the number of shares of each transaction by adding a deep neural network regressor to DQN. In addition, they adopt a transfer learning technique to pre-train neural networks when financial data is insufficiently large. The experiment results reveal that the total profit is significantly increased by forecasting the number of shares. Pendharkar and Cusatis (2018) design an on-policy SARSA and off-policy Q-learning for the purpose of asset allocation, train the RL agent with discrete action space, which can maximize the return of portfolio or differential Sharp ratio, and compare it with other RL methods in financial markets. Gao, Gao, Hu, Jiang, and Su (2020) propose a novel DQN framework, which is expressly designed for managing a multi-asset portfolio and allows DQN agents to optimize their trading strategies by interacting with the real financial market. Park et al. (2020) derive a novel portfolio trading strategy for multi-asset management in the practical action space and devise a transformation function that maps the infeasible action to similar feasible actions.

Deep Deterministic Policy Gradient (DDPG), proposed by Lillicrap et al. (2015), is one of the actor–critic algorithms that support continuous action space. Compared to DQN, the merit of DDPG is that it can handle high-dimensional continuous action problems well, and it directly outputs the optimal action instead of the Q-value. Jiang and Liang (2017) implement the DDPG algorithm that adopts a convolutional neural network to solve the asset allocation problem in the cryptocurrency market. They optimize the investment portfolio by weighting all stocks, and make it suitable for continuous-time actions to solve the discrete action space problem. A back-test experiment is applied in the cryptocurrency market, and their experimental results achieve positive results compared to another three RL portfolio management algorithms. Liang, Chen, Zhu, Jiang, and Li (2018) extend DDPG by using a deep residual network and propose an adversarial training method that improves the performance of deep RL. It has been tested on the Chinese stock market that illustrates this approach can significantly improve the training efficiency, average daily earnings and Sharp ratio. Xiong et al. (2018) explore the potential of training DDPG

[1] Real data is collected from Yahoo Finance.

[2] In its 12th annual Global Alternative Fund and Investor Survey, November 2018, Ernst & Young (EY) reports that more than 40% of hedge fund managers admit that they refer to AI to develop strategies to enhance performance for making greater profits in their investment process.

Table 1
Notations for the trading system.

Symbols	Explanations for the notation
$v^c_{i,t}$	Closing price of the ith asset in the tth trading period
$v^h_{i,t}$	Highest price of the ith asset in the tth trading period
$v^l_{i,t}$	Lowest price of the ith asset in the tth trading period
$v^o_{i,t}$	Opening price of the ith asset in the tth trading period
$v^v_{i,t}$	Volume of the ith asset in the tth trading period
$v_{i,t}$	Prices and volume of the ith asset in the tth trading period
$V_{i,t}$	Market information of the ith asset in the tth trading period
w_t	Portfolio weight at the beginning of period $t+1$
w'_t	Portfolio weight at the end of period t before execution
p_t	Portfolio value at the beginning of period $t+1$
p'_t	Portfolio value at the end of the period t before execution
y_t	Relative price change in the tth trading period
r_t	Rate of return at the end of period t
$\hat{\mu}_t$	Mean value of returns for each asset at the end of period t
μ_t	Mean value of portfolio return at the end of period t
$\hat{\Sigma}_t$	Variance–covariance matrix of returns for each asset at the end of period t
$\mathcal{R}_t$	Variance of portfolio return at the end of period t
c	Commission rate for buying and selling
C_t	Trading cost rate for the tth trading period
n	Number of risky assets
m	Window size

agents to obtain the optimal trading strategy in the stock market. They construct a portfolio that consists of 30 stocks, and the trading environment has been created by adopting the daily prices of each stock. Compared to the traditional minimum-variance method, the DDPG algorithm has gained higher benefits, which proves the effectiveness of the algorithm. Wu and Li (2020) construct Gate Deterministic Policy Gradient (GDPG) by adding the Gate Recurrent Unit into DDPG to extract financial features from the time-series stock market data. The performance of their proposed GDPG method is verified by comparing the experimental results, which shows that the GDPG method gains a higher return than the traditional DRL and it can spawn a more stable performance even in the turbulent financial market.

Despite a tremendous amount of research and high-quality results in the area of portfolio management by RL, there is very little literature that takes into account the portfolio risk, especially under the worst-case scenarios, in constructing trading strategies. Every financial product has its own risk and reward characteristics. The ultimate goal of investors is to choose the best portfolio with the highest return, and keep the portfolio risk below a certain degree. Thus, it is a significant and important task for the RL agent to construct a risk-sensitive or risk-averse policy for the investors who may suffer a huge loss caused by rare events.

3. Portfolio allocation problem

Portfolio optimization requires continuous reallocation of an investment fund into different assets. Our trading agent does this allocation periodically. The trading environment is formulated as follows. For the convenience of readers, we provide Table 1 that includes all symbols.

3.1. Problem formulation

The individual asset consists of the opening, highest, lowest, closing prices, and volume for each trading period. We denote by $v^c_{i,t}$ the closing price of the ith asset in the tth trading period. Similarly, $v^h_{i,t}$, $v^l_{i,t}$, $v^o_{i,t}$ denote the highest, lowest, and opening prices of the ith asset in the tth period, respectively. Denote by $v^v_{i,t}$ the volume of the ith asset in the tth period. For the tth trading period, the prices and volume of each individual asset can be expressed as

$$v_{i,t} = \left[v^o_{i,t}, v^h_{i,t}, v^l_{i,t}, v^c_{i,t}, v^v_{i,t}\right], \tag{3.1}$$

and the information the agent can observe on the ith stock at timestep t is written as

$$V_{i,t} = \begin{bmatrix} v^o_{i,t}, & v^h_{i,t}, & v^l_{i,t}, & v^c_{i,t}, & v^v_{i,t} \\ v^o_{i,t-1}, & v^h_{i,t-1}, & v^l_{i,t-1}, & v^c_{i,t-1}, & v^v_{i,t-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ v^o_{i,t-m+1}, & v^h_{i,t-m+1}, & v^l_{i,t-m+1}, & v^c_{i,t-m+1}, & v^v_{i,t-m+1} \end{bmatrix}, \tag{3.2}$$

where m is the window size.

For continuous markets, the relative price change in the tth trading period is defined as the element wise division of $v^c_{i,t}$ by $v^c_{i,t-1}$:

$$y_t = \frac{v^c_t}{v^c_{t-1}} = \left[1, \frac{v^c_{1,t}}{v^c_{1,t-1}}, \frac{v^c_{2,t}}{v^c_{2,t-1}}, \dots, \frac{v^c_{n,t}}{v^c_{n,t-1}}\right]^T, \tag{3.3}$$

where n is the number of stocks in the portfolio. Note that the first element of y_t represents the relative price of cash, therefore, it is always 1. We can use this relative price change vector to calculate the portfolio value in a period. The portfolio weight vector is defined as

$$w_t = [w_{0,t}, w_{1,t}, w_{2,t}, \dots, w_{n,t}], \tag{3.4}$$

where $w_{i,t}$ is the fraction of investment on stock i at the beginning of period $t+1$ with the initial portfolio weight vector $w_0 = [1, 0, 0, \dots, 0]$, and $\sum_{i=0}^{n} w_{i,t} = 1$ with each $w_{i,t} \geq 0$. Note that the initial value of the weight vector w_0 indicates that all the investment capital is in the riskless asset at the beginning. Assuming p_{t-1} is the portfolio value at the beginning of period t, ignoring transaction costs, the portfolio value at the end of period t can be calculated as

$$p_t = p_{t-1} y_t \cdot w_{t-1}, \tag{3.5}$$

where w_{t-1} is the portfolio weight vector at the beginning of period t and its ith element $w_{i,t-1}$ is the proportion of stock i in the portfolio after a reallocation of capital.

The rate of return at the end of period t can be calculated as

$$r_t = \frac{p_t}{p_{t-1}} - 1 = y_t \cdot w_{t-1} - 1, \tag{3.6}$$

and the corresponding logarithmic rate of return is given by

$$\log(r_t) = \log(\frac{p_t}{p_{t-1}} - 1) = \log(y_t \cdot w_{t-1} - 1). \tag{3.7}$$

The mean value and variance of portfolio return can be formulated as

$$\mu_t = w_t \cdot \hat{\mu}_t, \tag{3.8}$$

$$\mathcal{R}_t = w_t^T \hat{\Sigma}_t w_t, \tag{3.9}$$

where $\hat{\mu}_t$ and $\hat{\Sigma}_t$ are the mean value and the variance–covariance matrix of returns for each asset. Note that the $\hat{\mu}_t$ and $\hat{\Sigma}_t$ are estimated every step based on the observation and window size. If there is no transaction cost, the final portfolio value will evolve as follows

$$p_T = p_0 \prod_{i=1}^{T} (1 + r_i) = p_0 \prod_{t=1}^{T} y_t \cdot w_{t-1}, \tag{3.10}$$

where p_0 is the initial investment amount.

3.2. Transaction costs

In the real world, buying or selling assets incurs a transaction cost, usually in the form of commission fee. Assuming a constant commission rate, we can recalculate the final portfolio value. At the beginning of period t, the portfolio's action vector is w_{t-1}. Due to the price changes of assets in the market, the portfolio weight vector transforms to w'_t at the end of period t:

$$w'_t = \frac{y_t \odot w_{t-1}}{y_t \cdot w_{t-1}}, \tag{3.11}$$

where $\odot$ is element-wise multiplication. The mission of the agent is to reallocate portfolio weights from w'_t to w_t by buying or selling relevant

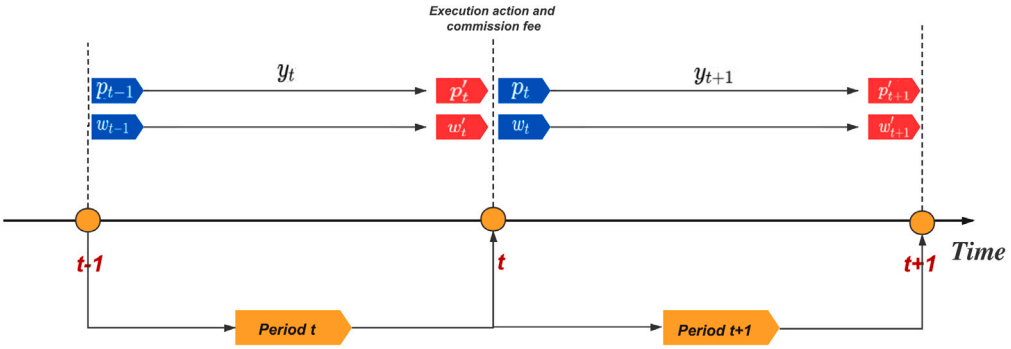

Fig. 1. The structure of our trading system.

assets. Paying all commission fees, this reallocation action shrinks the portfolio value. If we set a constant commission rate $c \in [0, 1)$ for buying and selling, then the trading cost rate of each period C_t can be approximated as (Hegde, Kumar, & Singh, 2018; Jiang & Liang, 2017):

$$C_t = c \sum_{i=1}^{n} \left| w'_{i,t} - w_{i,t} \right|, \tag{3.12}$$

where $C_t \in [0, 1)$. Assuming all buying and selling trades are executed at the end of a day, the portfolio value (3.5) at the end of day t evolves

$$p_t = (1 - C_t) p'_t, \tag{3.13}$$

where p'_t represents the portfolio value at the end of period t before execution, that is, $p'_t = p_{t-1} y_t \cdot w_{t-1}$.

Therefore, the rate of return (3.6) can be rewritten as:

$$r_t = \frac{p_t}{p_{t-1}} - 1 = (1 - C_t) y_t \cdot w_{t-1} - 1. \tag{3.14}$$

Hence, the final portfolio value can be expressed as

$$p_T = p_0 \prod_{t=1}^{T} (1 - C_t) y_t \cdot w_{t-1}. \tag{3.15}$$

Fig. 1 demonstrates the dynamic relationships among portfolio values and weight vectors on the time axis.

3.3. Assumption and restrictions

To simulate real-world market trades, we make several assumptions to formulate the problem. First of all, the actions are only executed at the end of a period. Second, we assume that the opening price is equal to the closing price of the previous day. After-sales market transactions are not allowed. Third, short selling is not allowed in our trading environment. Finally, we also assume the market is sufficiently liquid such that any transactions can be executed immediately with minimal market impact.

4. The classical DDPG algorithm

Portfolio management is a financial decision-making task, which aims at boosting the total profits or returns and lowering the risk via asset allocation. The asset allocation process can be constantly changed, so we require an off-policy agent using a DRL algorithm that maps a high dimensional state space to a high dimensional continuous action space. DDPG is an actor–critic based deep RL algorithm proposed in Lillicrap et al. (2015). It uses a neural network as a Q-function approximator and proposes a replay buffer to improve convergence to the optimal policy, because the proposed replay buffer resolves the problem that the learned action function is relatively unstable.

The classical DDPG algorithm has been developed by a Markov decision process, which consists of a state space $\mathcal{S}$, action space $\mathcal{A}$, an initial state distribution $p(s_0)$, transition dynamics $p(s_{t+1}|s_t, a_t)$, and reward function $r(s_t, a_t)$. The DDPG algorithm includes four neural networks: the actor network, the critic network and their respective target networks. In the initial stage, we randomly initialize each network and reset the replay buffer, and the DDPG agent aims to learn from interaction with the environment. At the beginning of the training process, the current state, next state, action and the immediate reward from the environment are stored in a replay buffer, then DDPG assembles a mini-batch from the replay buffer and feeds it to both the actor, critic, and their target networks. For each sample mini-batched from the replay buffer, the target actor network produces a target action according to the next state; the target Q-value is generated by the target critic network associated with the next states and target action. The target Q-values of the current actions and states are calculated from the immediate rewards and the discounted Q-values for the next states via the Bellman equation. The critic network is updated by minimizing the TD-error, calculated as the difference between target Q-value and actual Q-value, the actor network is then trained by adopting the policy gradient for the critic network. Finally, the target network weights are updated using a soft updates strategy from actor and critic networks. A soft update strategy includes smoothly mixing the regular network weights with target network weights. The structure of classical DDPG is shown in Fig. 2.

For the portfolio allocation problem, at each trading time t, we assume the DDPG agent only observes the market information of OHLCV data. With such an assumption, the observation s_t can be expressed as:

$$s_t = \left[V_{1,t}, V_{2,t}, \ldots, V_{n,t} \right], \tag{4.1}$$

where $V_{i,t}$ is defined by (3.2). We take the portfolio weight vector as an action, so action vector a_t is equal to weight vector w_{t-1}, where w_{t-1} denotes the portfolio weight vector at the beginning of period t. The DDPG agent aims to maximize the total profit, which is equivalent to maximizing the logarithmic return. Therefore, the reward function $r(s_t, a_t)$ taking into account the transaction cost is defined as

$$r(s_t, a_t) = \log \frac{p_t}{p_{t-1}} = \log((1 - C_t) y_t \cdot w_{t-1}). \tag{4.2}$$

Thus, we have the immediate reward at each timestep that avoids the sparsity of the reward problem. At each timestep t, the agent takes an action a_t based on the current observation s_t, and receives a reward $r(s_t, a_t)$. The total discounted future rewards until timestep T is given by

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i), \tag{4.3}$$

where the discount factor $\gamma \in [0, 1]$. The objective of reinforcement learning is to learn a policy by maximizing the expected discounted future rewards given the current state

$$J = \mathbb{E}\left[R_t \middle| s_t \right]. \tag{4.4}$$

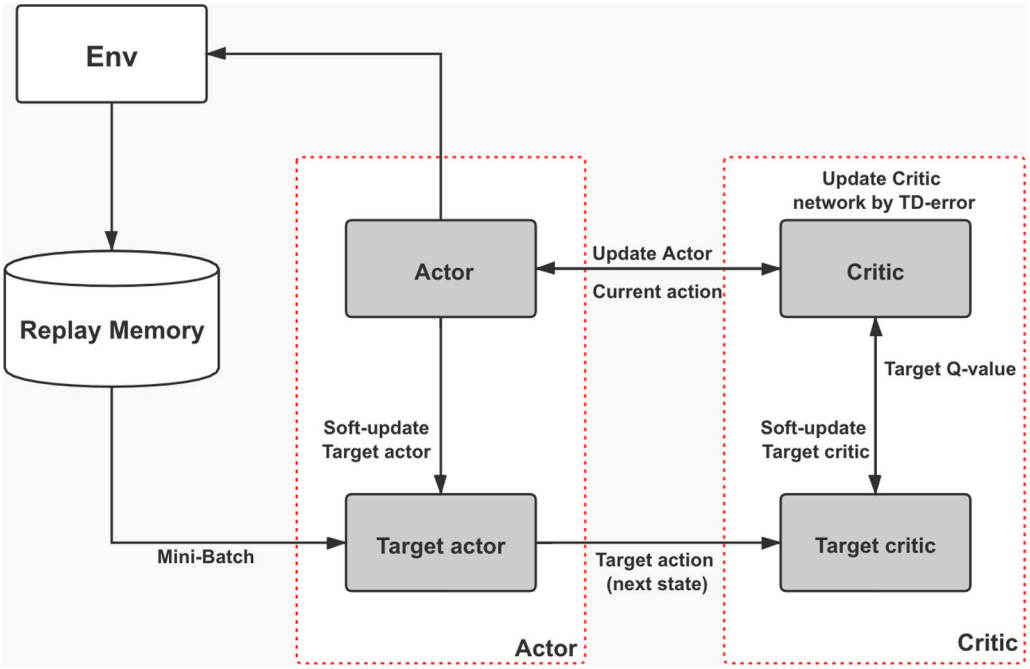

Fig. 2. The structure of classical DDPG.

By the Bellman equation, it allows us to compute the Q-value by recursion:

$$Q^{\mu}(s_t, a_t) = \mathbb{E}\Big[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))\Big]. \tag{4.5}$$

The parametrized actor function $\mu(s|\theta^{\mu})$ specifies the current policy by deterministically mapping states to a specific action $\mu : \mathcal{S} \rightarrow \mathcal{A}$. The critic network $Q(s, a)$ is updated by minimizing a squared TD-error below:

$$L = \frac{1}{N}\sum_{i}\Big[y_i - Q(s_i, a_i|\theta^{Q})\Big]^2, \tag{4.6}$$

where $y_i = r(s_i, a_i) + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$ and N is the number of transitions mini-batched from replay buffer.

Note that y_i is calculated by a separate target network which is softly updated, $Q'(\cdot)$ and $\mu'(\cdot)$ represent the target critic and actor network with the parameter $\theta^{Q'}$ and $\theta^{\mu'}$, respectively. The actor is updated by the following gradient of J with respect to the parameter θ^{μ} based on the policy gradient theory from Silver, Lever, Heess, Degris, Wierstra, and Riedmiller (2014)

$$\begin{aligned}\nabla_{\theta^{\mu}} J &= \mathbb{E}\Big[\nabla_{\theta^{\mu}} Q(s, a\Big|\theta^{Q})\Big|_{s=s_t, a=\mu(s_t|\theta^{Q})}\Big] \\ &= \mathbb{E}\Big[\nabla_a Q(s, a\Big|\theta^{Q})\Big|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})\Big|_{s=s_t}\Big].\end{aligned} \tag{4.7}$$

5. Proposed approaches

5.1. The distributional DDPG model

Although most deep RL aims to optimize the decision-making rule in terms of the expected future discounted rewards, the agent sometimes for some specific purpose aims to seek a big win on rare occasions or avoid a rare likelihood of suffering a huge loss. To reduce the effects of rare bad events, the distributional RL is proposed by Dearden, Friedman, and Russell (1998) and Engel, Mannor, and Meir (2005), which aims to adopt a Gaussian distribution to approximate the distribution of future returns and model the uncertainty under this approximate distribution. Distributional RL has the benefit of considering the risks that may exist when future returns are stochastic since the observable state cannot capture the intrinsic randomness of the environment. In addition, if there is a high variance or heavy tail in return distribution, the strategy of maximizing average return may lead to over-estimation of the expected future reward. Motivated by Barth-Maron et al. (2018), Tang et al. (2020), and Bellemare, Dabney, and Munos (2017), we apply the distributional RL algorithm to the portfolio management problem, namely Distributional DDPG, which constructs a risk-sensitive policy to reduce the effects of disaster events or potential losses (see Fig. 3).

The standard Markov decision process (MDP) consists of a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ represent the state space and action space respectively, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ denotes the reward function, γ denotes the discount factor, and $\mathcal{P}$ is the transition probability density of moving the current state into the next state. The action space $\mathcal{A}$ is assumed to be continuous. Suppose that R is a random variable for future return, $p(R|s, a)$ is the probability distribution of future returns, which is given the current state s and action a. The *α-percentile* expectation[3] that represents the expected return under the bottom *α-percentile* of the distribution over returns is employed as the criterion of distributional RL. The objective function can be formulated as:

$$J_{\alpha} = \mathbb{E}[R\Big|R \leq percentile(\alpha), s], \tag{5.1}$$

where $\alpha \in [0, 1]$ is a risk parameter. When $\alpha \rightarrow 0$, the strategy will concentrate on doing well in the worst case, while when $\alpha \rightarrow 1$, the strategy aims to perform well in the average performance. We combine the distributional RL with classical DDPG so that the critic learns to model the distribution over the expected total discounted rewards. Our proposed method combines the distributional RL and DDPG, which enables critics to simulate the distribution of future returns. As long as a good distribution can be learned by critics, then the actor network is updated by backpropagating the gradient back through the critic network.

Distributional DDPG includes an actor–critic network structure of the DDPG algorithm, and contains a distribution of future return $Z(s, a)$ that is a mapping from state–action pairs to distributions over returns. The distributional Bellman equation points out the distribution of

[3] We note that some literature call this CVaR, but we do not use it here to avoid confusion.

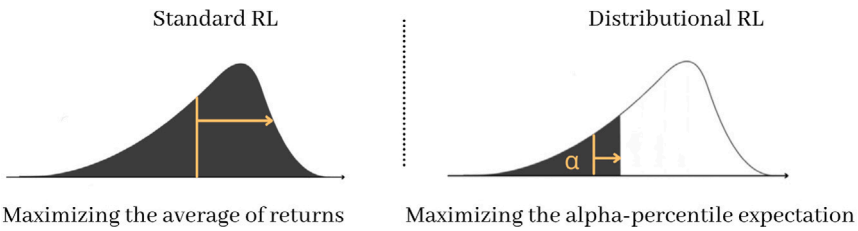

Fig. 3. Distributional RL.

$Z(s,a)$ is evaluated by three associated random variables: the reward r, the next state–action (s',a'), and its return $Z(s',a')$. The nature of distributional return $Z(s,a)$ is described by a following recursive equation:

$$Z(s,a) \overset{D}{=} r(s,a) + \gamma Z(s',a'), \tag{5.2}$$

where $U \overset{D}{=} V$ indicates that the random variable U has the same distribution pattern as V. In here, $Z(s,a)$ represents the inherent stochasticity of the interaction between the agent and the environment. Then, the transition operator P^{μ} is defined as:

$$P^{\mu} Z(s,a) \overset{D}{=} Z(s',a'), \quad s' \sim p(\cdot|s,a), a' \sim \mu(\cdot|s), \tag{5.3}$$

and the distributional Bellman operator $\mathcal{T}^{\mu}$ is given by

$$\mathcal{T}^{\mu} Z(s,a) \overset{D}{=} r(s,a) + \gamma P^{\mu} Z(s,a). \tag{5.4}$$

This implies that the two sides of a distributional equation relate to the distribution of two independent random variables, and it can be used to train the distributional reinforcement learning in many areas of research. Similar to the expected Bellman operator, the distributional Bellman operator can be proved to converge to the true return distribution. The convergence theory of the distributional Bellman operator has been proven by a contraction lemma, which needs to evaluate the distance between two return distributions (Rowland, Bellemare, Dabney, Munos, & Teh, 2018).

The Wasserstein metric is the main tool to measure the distance between cumulative distribution functions, proposed by Bickel and Freedman (1981). Different from the Kullback–Leibler (KL) divergence, the Wasserstein metric is a true probability that takes into account the probability of distances between various outcome events, which leads to the Wasserstein metric being well-suited for the field that exists an underlying similarity. For $p < \infty$, the pth Wasserstein distance between two probability distributions F_U and F_V is defined as (Olkin & Pukelsheim, 1982):

$$W_p(U,V) = \left(\int_0^1 \left| F_U^{-1}(s) - F_V^{-1}(s) \right|^p ds \right)^{1/p}, \tag{5.5}$$

where F^{-1} is the inverse cumulative distribution function (CDF). Assuming that $U \sim \mathcal{N}(\mu_1, C_1)$ and $V \sim \mathcal{N}(\mu_2, C_2)$, the 2-Wasserstein distance simplifies to:

$$W_2(U,V) = |\mu_1 - \mu_2|^2 + C_1 + C_2 - 2(C_1 C_2)^{\frac{1}{2}}. \tag{5.6}$$

As in Tang et al. (2020), we model $Z(s,a)$ as a Gaussian distribution, which provides a closed-form of the *α-percentile* expectation.[4] The output of the critic network can be expressed as the estimated mean and variance of future returns $Z(s,a)$ with weights θ^{Q}:

$$f_{critic}(s,a,\alpha|\theta^{Q}) \rightarrow \{\hat{Q}(s,a,\alpha), \hat{\mathcal{V}}(s,a,\alpha)\}, \tag{5.7}$$

where $f_{critic}(s,a,\alpha|\theta^{Q})$ denotes the critic network with input state s, action a, and risk parameter α. We adopt Convolutional Neural Network (CNN) for the critic network. Three hidden convolution layers with *Relu* activation function are added following the input layer. Then, we modify the output layer that predicts the estimated value of mean and variance of the future returns. The *Softplus* activation function is applied to predict the variance of the future returns, which keeps the variance always positive. The convergence proofs[5] for the critic network are given in Appendix A.

With the benefit of the critic's structure, the estimated $\hat{Q}(s,a,\alpha)$ and $\hat{\mathcal{V}}(s,a,\alpha)$ are applied to calculate the *α-percentile* expectation in closed-form. Let $\Gamma^{\mu}(s,a,\alpha)$ denotes the *α-percentile* expectation in given state s, executing action a, and following policy μ hereafter, the closed-form of *α-percentile* expectation is formulated as:

$$\Gamma^{\mu}(s,a,\alpha) = \mathbb{E}[R|R \le percentile(\alpha), s, a] = \hat{Q}(s,a,\alpha) - \frac{\varphi(\alpha)}{\Phi(\alpha)} \sqrt{\hat{\mathcal{V}}(s,a,\alpha)}, \tag{5.8}$$

where $\varphi(\cdot) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ is the standard normal p.d.f., and $\Phi(\cdot)$ is its CDF.

Therefore, the objective function (5.1) can be rewritten as:

$$J_{\alpha} = \mathbb{E}[R|R \le percentile(\alpha), s] = \int_S \rho^{\mu}(s) \int_A \mu_{\theta_a}(a|s,\alpha) \Gamma^{\mu}(s,a,\alpha) da ds, \tag{5.9}$$

where ρ^{μ} denotes the stationary distribution over the state space given the policy μ.

Then, the actor is updated by the following deterministic gradient, which adopts the chain rule to the *α-percentile* expected return with respect to the actor parameters (Silver et al., 2014):

$$\nabla_{\theta_a} J_{\alpha} = \mathbb{E}\left[\nabla_{\theta_a} \mu(a|s,\alpha) \hat{Q}(s,a,\alpha) - \frac{\varphi(\alpha)}{\Phi(\alpha)} \nabla_a \sqrt{\hat{\mathcal{V}}(s,a,\alpha)} \nabla_{\theta_a} \mu(a|s,\alpha) \right]. \tag{5.10}$$

Note that the objective function J_{α} is dependent on the risk levels (αs). In the training process, we uniformly sample $\alpha \sim Uniform(0,1)$ at the beginning of the episode and fix α for the whole episode. During the testing period, the policy μ can yield different actions in given the same state s, conditioned on the setting of α. Intuitively, a small value of α leads to conservative behavior while a larger value of α leads to more aggressive behavior (see Algorithm 1). The structure of Distributional DDPG is displayed in Fig. 4.

5.2. The Hierarchical DDPG model

In this subsection, we propose a novel algorithm, called Hierarchical DDPG, which adds the Hierarchical structure to the DDPG algorithm. Original Hierarchical RL refers to the concept of decomposing RL problem into sub-problems (sub-tasks). Solving each sub-task will be more

[4] Without this assumption, it requires choosing another algorithm to approximate the sample Bellman updates and minimize the Wasserstein metric in each step, which is computationally too expensive.

[5] We note that the 2-Wasserstein distance W_2 cannot be directly used to bound the variance difference.

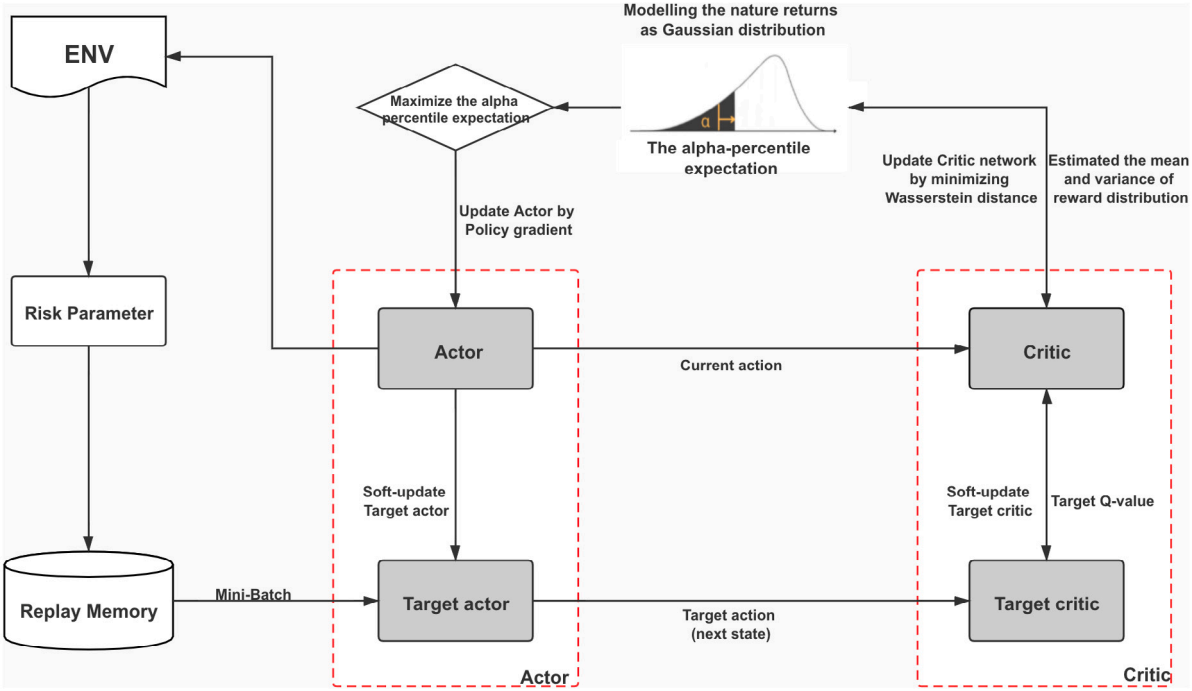

Fig. 4. The structure of Distributional DDPG.

Algorithm 1 Distributional DDPG

procedure TRAINING
Randomly initialize critic and actor network of agent with weights θ^{Q} and θ^{μ} .
Initialize target actor network and critic network with weights $\theta^{Q'} \leftarrow \theta^{Q}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$.
Initialize replay buffer $\mathcal{B}$
  for *episode* $= 1, M$ **do**
    Initialize an OU random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    Sample $\alpha \sim$ *Uniform*$(0, 1)$
    for $t = 1, T$ **do**
      Sample action $a_t = \mu(s_t, \alpha|\theta^{\mu}) + \mathcal{N}$
      Observe reward r_t, next state s_{t+1} from environment
      Store transition $\{s_t, a_t, r_t, s_{t+1}, \alpha\}$ into $\mathcal{B}$
      Sample a random mini-batch of N transitions from $\mathcal{B}$
      Using target network to approximate $\mathcal{T}^{\mu}Z(s, a)$ distribution by calculating the mean and variance from the critic network.
      Update critic network θ^{Q} by minimizing Wasserstein distance in equation (5.6)
      Update actor network θ^{μ} by using sample deterministic policy in equation (5.10)
      Update the target network by soft-update
      $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
      $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
    end for
  end for
end procedure

vigorous and efficient than solving the whole problem. The investor's goal is to select the best portfolio with the highest total profit and lowest portfolio risk for his/her investment. However, when there is a chance of gains and losses, most investors would prefer to avoid losses. Our Hierarchical DDPG algorithm utilizes the structure of Hierarchical RL in which the two-level mechanism allows the agent to avoid the potential loss, and balance the portfolio profit and risk for different scenarios.

The Hierarchical DDPG framework develops from classical DDPG and Hierarchical RL, which consists of lower-level and higher-level policy. The lower-level policy is an actor–critic based structure, and it is interpreted as a *worker* that selects primitive actions at every time step by maximizing the logarithmic rate of return when the portfolio risk is lower than a certain level. The higher-level policy is also an actor–critic based structure, and it is interpreted as a *manager* that selects the action according to the observation state and the action generated from the lower-level policy by minimizing the portfolio risk when the portfolio risk exceeds the tolerance of investors. In other words, the *manager* makes an adjustment to reduce the portfolio risk based on the *worker*'s action when the portfolio risk exceeds a certain level of risk. The critics of the *manager* and *worker* are used to evaluate their works. More specifically, by exploiting the market information from the environment, the *worker* observes the state from the environment, and produces the action g_t to maximize the total profit. Then, we employ an indicator to check whether the portfolio risk exceeds the investor's tolerance. If the investor can afford the potential loss, then the action g_t will be executed and received the rewards from the environment. If not, the *manager* will adjust the *worker*'s trading strategy and yield the action a_t based on the observation state and the *worker*'s action g_t to reduce the portfolio risk. At the final stage, the *manager* will execute the action a_t in the environment. The main idea of Hierarchical DDPG is displayed in Fig. 5.

Now, we introduce the indicator to measure the portfolio risk. Conditional Value-at-Risk (CVaR) is often used as a measure of risk and is also referred to as expected excess loss or expected shortfall. CVaR is a coherent risk measure and more attractive compared to Value-at-Risk (VaR) because it takes into account the contribution from the very rare but very large losses. Rockafellar, Uryasev, et al. (2000) employ CVaR as the risk measure and minimize CVaR to compute an optimal investment portfolio. Krokhmal, Palmquist, and Uryasev (2002) propose a new approach for optimizing CVaR in portfolio optimization problems. They extend the Rockafellar et al. (2000) approach by maximizing expected returns under CVaR constraints. CVaR constraints are used to limit the percentiles of the loss distribution and sculpt the loss distribution according to the decision makers' preferences. Linear Programming

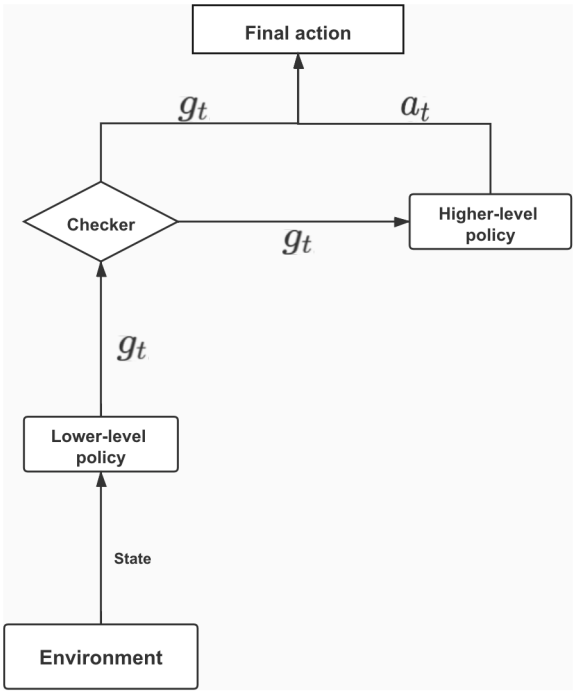

Fig. 5. The main idea of Hierarchical DDPG.

(LP) approach is one of the standard approaches for solving CVaR optimization problems. A piecewise linear function can approximate the typical continuously differentiable CVaR function by adopting the Monte Carlo simulation. In this paper, we apply the parametric CVaR approach under the Gaussian distribution to measure the portfolio risk. Parametric CVaR_α can be formulated as a closed-form:

$$\text{CVaR}_\alpha(a_t) = \mathcal{R}_t \frac{\varphi(\Phi^{-1}(\alpha))}{\alpha} - \mu_t, \tag{5.11}$$

where μ_t and $\mathcal{R}_t$ are the mean and variance of the portfolio return defined in (3.8), and $\varphi(\cdot)$ is the standard normal p.d.f., $\Phi(\cdot)$ is the standard normal CDF, so $\Phi^{-1}(\alpha)$ is the standard normal quantile. When the portfolio risk is below the CVaR risk constraints ($\text{CVaR}_\alpha \leq C$), the lower-level policy aims to seek an aggressive trading strategy by maximizing the total profit (logarithm rate of return). On the other hand, when it exceeds the CVaR constraints ($\text{CVaR}_\alpha > C$), the main goal is to reduce the portfolio risk instead of maximizing the total profit. The higher-level policy makes an adjustment of trading strategy and aims to seek a conservative trading strategy to reduce the risk by maximizing the expected future discount reward of the higher-level policy. The structure of Hierarchical DDPG is shown in Fig. 6.

Next we introduce the reward of the lower-level policy and higher-lever policy. The reward of the lower-level policy is the same as in the classical DDPG algorithm (see (4.2)), and the reward function for higher-level policy is defined as

$$r'(s_t, g_t, a_t) = \text{CVaR}_\alpha(g_t) - \text{CVaR}_\alpha(a_t). \tag{5.12}$$

This implies the main goal of higher-level policy is to reduce the portfolio risk immediately compared with the lower-level's action. Then, the objective function for higher-level policy is given by

$$J^{(H)} = \mathbb{E}[R'_t | s_t], \tag{5.13}$$

where $R'_t = \sum_{i=t}^{T} \gamma^{i-t} r'(s_i, g_i, a_i)$.

Define the $Q^{(H)}$ function for higher-level network as:

$$Q^{(H)}(s_t, g_t, a_t) = \mathbb{E}[R'_t | s_t, g_t, a_t], \tag{5.14}$$

where g_t is the action from the lower-level policy, and a_t is the action from the higher-level policy.

Table 2
Data description.

ETFs	Description
SPY	SPY is one of the most popular and oldest ETFs designed to track the Standard & Poor's 500 index. It holds a portfolio of 500 securities, which are selected by the S&P Committee to represent large-cap companies in the United States. At present, the top 3 sectors in which SPY holds shares are technology, finance and health care, and the top 5 stocks in the portfolio include Microsoft, Apple, Amazon, Facebook and Berkshire Hathaway.
VGK	VGK tracks all capitalization and market capitalization weighted indices of developed European securities. It is a subset of the FTSE global stock index series, covering about 98% of the global market, and diversified enough to invest across various industries.
GXC	GXC tracks the S&P China BMI, which is a rules-based index that measures the performance of global equity markets. It includes major share classes like A, B, H, red chips, P chips, and foreign listings. The fund typically invests almost all (but at least 80%) of its total assets in the securities comprising the index.
EWG	EWG tracks a market-cap-weighted index of large and midcap German companies. It aims to provide concentrated exposure to large- and midcap segments of the German equity market, meaning it covers the top 85% of the German companies by market cap. It primarily consists of stocks traded on the Frankfurt Stock Exchange.

Applying the Bellman equation to (5.14), we have

$$Q^{(H)}(s_t, g_t, a_t) = r(s_t, g_t, a_t) + \gamma Q^{(H)}(s_{t+1}, \mu_1(s_{t+1}), \mu_2(s_{t+1}, \mu_1(s_{t+1}))), \tag{5.15}$$

where $\mu_1(\cdot)$ is the policy from the lower-level, and $\mu_2(\cdot,\cdot)$ is the policy from the higher-level.

To update the higher-level policy network, the policy gradient with respect to the parameter θ^{μ_2} is given by

$$\begin{aligned}\nabla_{\theta^{\mu_2}} J^{(H)} &= \mathbb{E}_{s_t}\left[\nabla_{\theta^{\mu_2}} Q^{(H)}(s, g, a|\theta^H)\Big|_{s=s_t, g=\mu_1(s_t), a=\mu_2(s_t, \mu_1(s_t)|\theta^{\mu_2})}\right] \\ &= \mathbb{E}_{s_t}\left[\nabla_a Q^{(H)}(s, g, a|\theta^H)\Big|_{s=s_t, g=\mu_1(s_t), a=\mu_2(s_t, \mu_1(s_t)|\theta^{\mu_2})} \nabla_{\theta^{\mu_2}} \mu_2(s, g|\theta^{\mu_2})\Big|_{s=s_t, g=\mu_1(s_t)}\right].\end{aligned} \tag{5.16}$$

This is derived in the same way as (4.7). The Hierarchical DDPG algorithm is given below (see Algorithm 2).

6. Experiment results

6.1. Data

We conduct various experiments to verify our proposed approaches by using four different index ETFs: "SPY", "VGK", "GXC", and "EWG"(see Table 2). SPY is the S&P 500 index ETF, which measures the stock performance of 500 large companies in the U.S. Market. VGK is the index ETF for the European All Cap developed by FTSE, which tracks the performance of major markets in Europe. GXC is one of the most comprehensive China equity funds available to U.S. investors, which is dominated by holding large-cap stocks and delivers greater diversification from a security perspective. Lastly, EWG aims to provide concentrated exposure to large and midcap segments of the German equity market, meaning it covers the top 85% of the German companies by market cap. It primarily consists of stocks traded on the Frankfurt Stock Exchange. The dataset, obtained from Yahoo Finance, consists of daily prices and volume data over a 10-year period from 2010-01-01 to 2020-07-30. The training set and testing set are distributed according to the ratio of 8:2. For the purpose of training and testing, two independent trading environments are designed. In addition, short selling is not allowed, and a commission rate of 0.25% will be deducted for each transaction for all experiments.

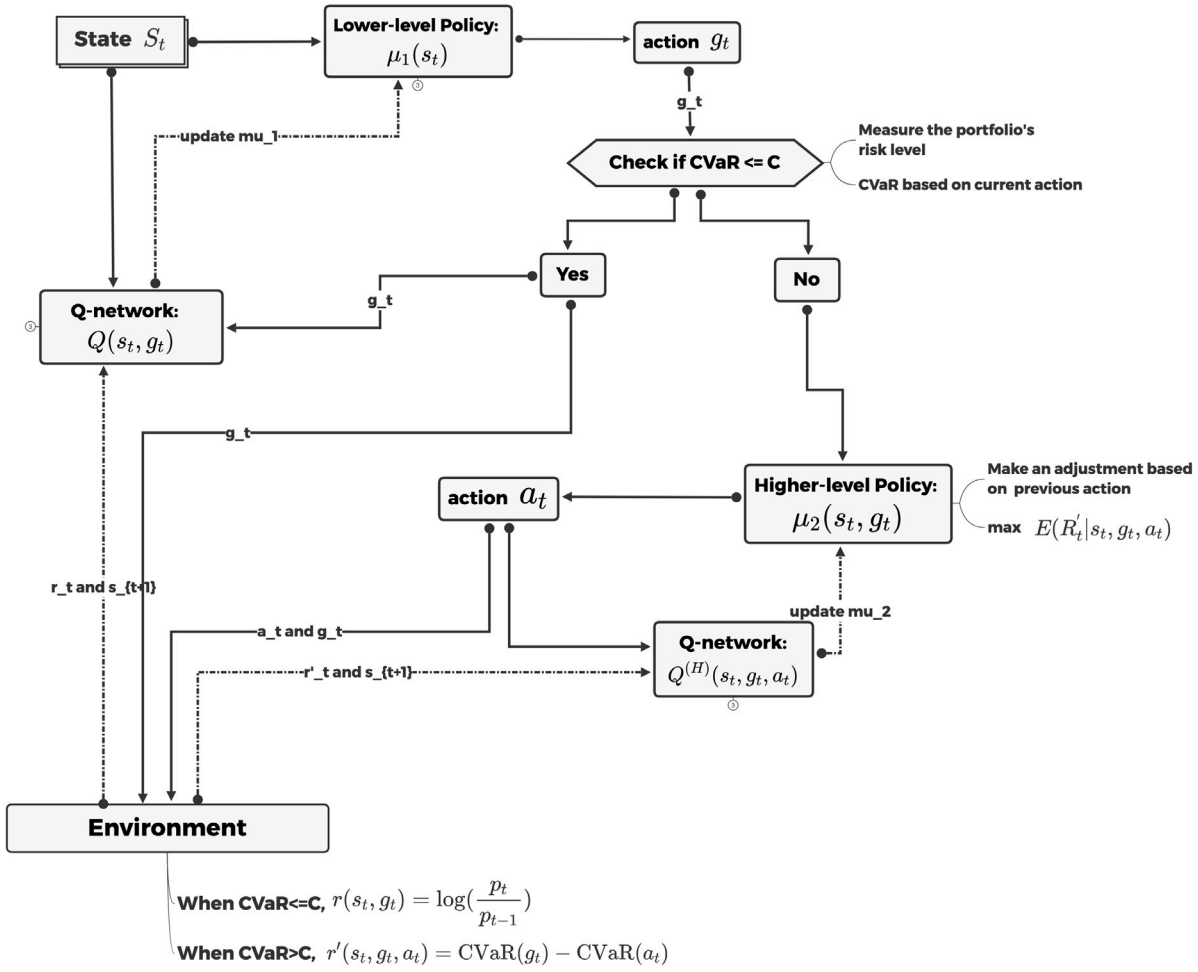

Fig. 6. The structure of Hierarchical DDPG.

Data Preprocessing

The absolute prices and volumes of the assets, i.e., opening, highest, lowest, closing prices, and volume in the problem are not sensitive to the agent for making any trading decisions, but the changes in prices and volumes are important to the agent. Therefore, the input prices and volumes to the network need to be normalized. To be specific, we divide the opening, closing, highest, lowest prices by the closing price on the last day of the period, and divide the volumes by the volume on the last day of the period. For example, the input state with window size m and number of assets n is given by

$$s_t = \left[V'_{1,t}, V'_{2,t}, \ldots, V'_{n,t}\right], \tag{6.1}$$

where $V'_{i,t}$ is the information of the ith stock at time t after normalization, given by

$$V'_{i,t} = \begin{bmatrix} \frac{v^o_{i,t}}{v^c_{i,t}}, & \frac{v^h_{i,t}}{v^c_{i,t}}, & \frac{v^l_{i,t}}{v^c_{i,t}}, & 1, & 1 \\ \frac{v^o_{i,t-1}}{v^c_{i,t}}, & \frac{v^h_{i,t-1}}{v^c_{i,t}}, & \frac{v^l_{i,t-1}}{v^c_{i,t}}, & \frac{v^c_{i,t-1}}{v^c_{i,t}}, & \frac{v^v_{i,t-1}}{v^v_{i,t}} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \frac{v^o_{i,t-m+1}}{v^c_{i,t}}, & \frac{v^h_{i,t-m+1}}{v^c_{i,t}}, & \frac{v^l_{i,t-m+1}}{v^c_{i,t}}, & \frac{v^c_{m,t-m+1}}{v^c_{i,t}}, & \frac{v^v_{i,t-m+1}}{v^v_{i,t}} \end{bmatrix}. \tag{6.2}$$

6.2. Evaluation metrics

Portfolio optimization problems involve determining the best asset allocation for an investment fund in accordance with specific objectives, such as maximizing portfolio return and minimizing portfolio risk. Assessing the performance of a trading strategy objectively and rationally can be a challenging and difficult task. An outperforming trading strategy is not only expected to generate a higher profit but also alleviate the portfolio risk associated with the trading activity. In this subsection, three evaluation metrics are introduced to assess the portfolio performance, that is, *Accumulated return*, *Sharpe ratio*, and *Maximum drawdown*.

Accumulated return

The Accumulated return is one of the popular evaluation metrics used to assess the portfolio profit. The higher Accumulated return implies the portfolio yields a higher profit. Consider a portfolio having the arithmetic return R_t at time t, the accumulated return can be calculated as

$$Accumulated\ return = \prod_{i=1}^{t}(1 + R_i). \tag{6.3}$$

This is the standard metric used to compare performance and relates the wealth at time t, W_t, with the initial wealth, W_0, as $W_t = W_0 \times Accumulated\ Return$. In this paper, all trading experiments adopt initial wealth $W_0 = 1$.

Sharpe ratio

The Sharp ratio is a performance metric that is widely and frequently used in the fields of finance and portfolio management because it takes into account both the profit and risk of the portfolio. This indicator is developed by Nobel laureate William F. Sharpe, and expresses as the excess return per unit of risk that is evaluated as the standard deviation of return. The Sharp ratio can be written as follows:

$$Sharpe\ ratio = \frac{E(R_t) - R_f}{\sigma(R_t)}, \tag{6.4}$$

Algorithm 2 Hierarchical DDPG

procedure TRAINING
Randomly initialize the critic and the actor networks of agent with weights θ^{Q} and θ^{μ_1}.
Randomly initialize the higher-level actor and critic networks with weights θ^{H} and θ^{μ_2}.
Initialize target networks and critic networks with weights $\theta^{Q'} \leftarrow \theta^{Q}$ and $\theta^{\mu_1'} \leftarrow \theta^{\mu_1}$.
Initialize replay buffer $\mathcal{B}_1$ and $\mathcal{B}_2$.
  for *episode* $= 1, M$ **do**
    Initialize an OU random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for $t = 1, T$ **do**
      Sample action $g_t = \mu_1(s_t|\theta^{\mu}) + \mathcal{N}$
      Check the portfolio's risk level
      if CVaR $\leq$ C **then**
        Observe reward r_t, next state s_{t+1} from environment
        Store transition $\{s_t, g_t, r_t, s_{t+1}\}$ into $\mathcal{B}_1$
        Sample a random mini-batch of N_1 transitions from $\mathcal{B}_1$
        Set $y_i = r(s_i, g_i) + \gamma Q'(s_{i+1}, \mu_1'(s_{i+1}|\theta^{\mu_1'}))$ for all $i \in N_1$
        Update θ^{Q} by minimizing loss $L(\theta^{Q}) = \frac{1}{N}\sum_i (y_i - Q(s_i, g_i|\theta^{Q}))^2$
        Update the actor policy θ^{μ_1} using the sampled policy gradient:

$$\frac{1}{N_1}\sum_i \nabla_g Q(s, g|\theta^{Q})|_{s=s_i, g=\mu_1(s_i)} \nabla_{\theta^{\mu_1}} \mu_1(s|\theta^{\mu_1})|_{s=s_i}. \tag{5.17}$$

        Update the target network by soft-update
        $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$.
        $\theta^{\mu_1'} \leftarrow \tau\theta^{\mu_1} + (1-\tau)\theta^{\mu_1'}$.
      end if
      if CVaR $>$ C **then**
        $a_t = \mu_2(s_t, g_t, a_t)$
        Store transition $\{s_t, g_t, a_t, r_t, s_{t+1}\}$ into $\mathcal{B}_2$
        Sample a random mini-batch of N_2 transitions from $\mathcal{B}_2$
        Set $y_j^{(H)} = r(s_j, g_j, a_j) + \gamma Q^{(H)}(s_{j+1}, \mu_1(s_{j+1}), \mu_2(s_{j+1}, \mu_1(s_{j+1})))$ for all $j \in N_2$
        Update θ^{H} by minimizing loss $L(\theta^{H}) = \frac{1}{N_2}\sum_j (y_j^{(H)} - Q^{(H)}(s_j, g_j, a_j|\theta^{H}))^2$
        Update the higher-level policy θ^{μ_2} using the sampled policy gradient:

$$\frac{1}{N_2}\sum_j \nabla_a Q^{(H)}(s, g, a|\theta^{H})\Big|_{s=s_j, g=\mu_1(s_j), a=\mu_2(s_j, \mu_1(s_j))} \nabla_{\theta^{\mu_2}} \mu_2(s, g|\theta^{\mu_2})\Big|_{s=s_j, g=\mu_1(s_j)}. \tag{5.18}$$

      end if
    end for
  end for
end procedure

where R_f is a risk-free return, $E(R_t)$ and $\sigma(R_t)$ represent the expectation and standard deviation of returns, respectively.

Maximum drawdown

Maximum drawdown (MDD) is another metric to assess the potential loss that seeks the maximum change from the highest to the lowest. It is dedicated to capital preservation that is the main anxiety for most rational investors. For example, when the MDD is quite small, it implies a minor loss from investment, and when an investment has never been lost, the MDD would be zero. On the other hand, the worst possible maximum drawdown would be −100%, meaning the investment is completely worthless. The maximum drawdown can be calculated as follow:

$$Max\ drawdown = \max \frac{R_t - R_{t+1}}{R_t}, \tag{6.5}$$

where R_t and R_{t+1} represent the rate of return in period t and $t+1$, respectively.

6.3. The results

This subsection presents the experimental results of our proposed methods and evaluates the effectiveness of our approaches. We obtain the opening, highest, lowest, closing prices, and volume of four ETFs. These four ETFs have different patterns, the movement of the closing prices and their details are described in Fig. 7. These prices are expressed in the U.S. dollar. As shown in Fig. 7, GXC shows the most gradual increase in these ETFs; SPY shows an upward trend but has been more volatile; EWG and VGK do not show a particular movement.

6.3.1. The experimental results for DDPG

The DDPG agent is trained on the training environment and tested on the testing environment separately.[6] The window size of the trading system is ten trading days, which indicates that the DDPG agent can observe the prices and trading volumes in the past ten days. The training process is carried for 500 iterations, and each iteration consists of 128 steps until the actor and critic networks get convergence to the optimal. To avoid convergence to local optimum policies, the weights of actor and critic are saved for the best performance. The actor and critic network adopts CNN with three hidden layers, and each convolution layer is a fully-connected layer with the activation of *ReLu*. The weights of actor and critic networks are randomly initialized at the beginning of each episode. The *Softmax* outputs of the actor network generate the actual corresponding portfolio weights.

Figs. 8 and 9 show the performance of DDPG with the ten-day window size for the training and testing period, respectively. As shown in Fig. 8, the portfolio value has surprisingly increased by 138.84% during the training period; it achieves outstanding performance compared to the market value. Here, the market value presents a portfolio that consists of equally-weighted investment assets. The maximum drawdown and Sharpe ratio of the DDPG portfolio are 8.66% and 80.48%, respectively. In addition, Fig. 9 shows that the portfolio of DDPG has given a 10.05% accumulated rate of return on investment at the end of the testing period, and their maximum drawdown and Sharpe ratio indices are 13.58% and 36.47%, respectively. These evidences indicate that the trading strategy of the DDPG agent has a higher potential risk and suffers a massive loss when the financial market crashes.

In addition, we test the effect of window size on the portfolio performance. The different window sizes $(5, 10, 20, 25)$ are applied to our experiment, the experimental results during the testing period are displayed in Table 3. While using window sizes of 20 and 25, the accumulated return increases to 12.93% and 14.74%, improved by 2.43% and 4.24% compared to the case of ten-day window size, respectively. Furthermore, the Sharpe ratio rises to 38.44% and 39.94% at the end of the testing period compared to the case of ten-day window size. These imply that the DDPG agent can construct a better portfolio when she observes more trading prices and volumes. One possible explanation is that the DDPG agent can predicate more accurate trends or movements based on more information she observed. The portfolio performances with different window sizes for the training and testing period are presented in Appendix B.

[6] Programming Code is deposited in Code Ocean at https://codeocean.com/capsule/0769244/tree/v1.

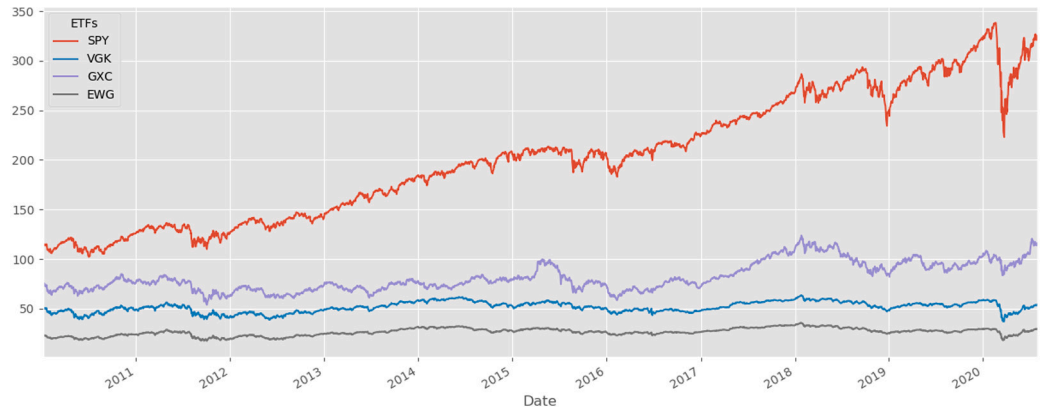

Fig. 7. The closing prices of each ETF.

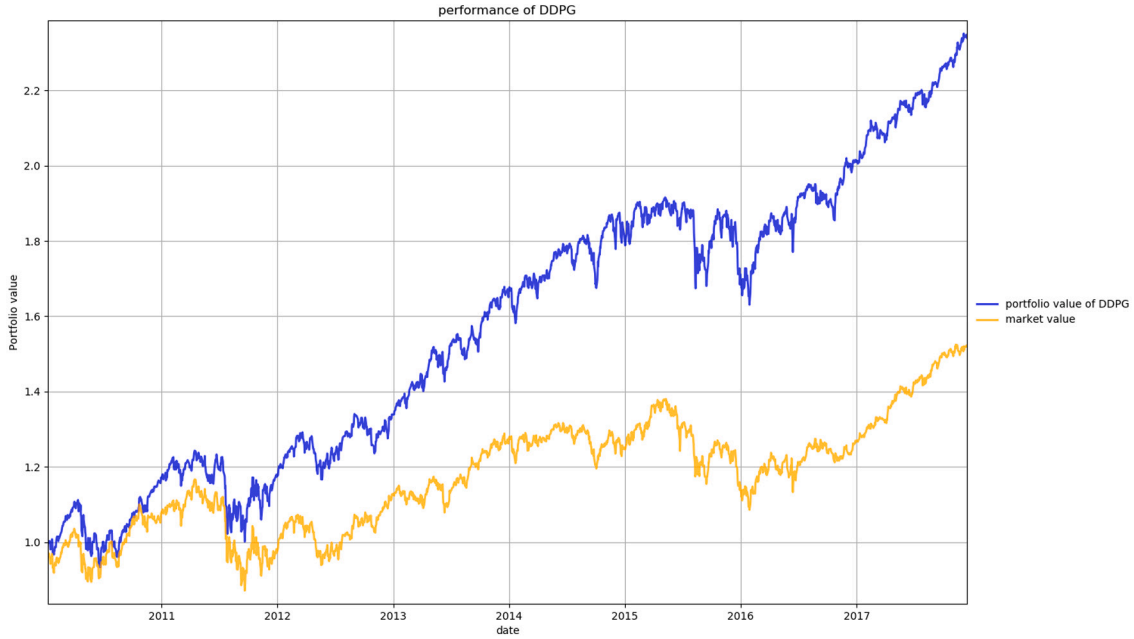

Fig. 8. The portfolio value of DDPG under window size of ten-day during the training period.

6.3.2. *The experimental results for distributional DDPG*

Fig. 10 shows the price movements of the portfolio with different risk parameters α during the testing period with the ten-day window size. Although the accumulated portfolio value has not increased much, the maximum drawdown has significantly decreased. When $\alpha = 5\%$, the agent only takes into consideration the worst-case, the agent is willing to choose cash instead of investing funds in other risky assets. As α increases, the agent is not willing to consider the extreme cases, and aims to allocate more investment funds into these risky assets. Therefore, we can see from Fig. 10 that the accumulated return increases to 4.22% and 10.32% when risk parameter α are 15% and 30%, respectively. Also, it reveals that the maximum drawdown of the distributional DDPG portfolio decreases as the risk parameter α decreases. This illustrates that the investor may suffer a larger potential loss during the financial crisis when the investor is willing to tolerate more risk. Furthermore, the Sharpe ratio increases as the α increases, which indicates that the earning per unit risk of this portfolio increases when the agent is willing to take more risk. These points demonstrate that Distributional DDPG can construct a more robust trading policy according to the investor's risk preference.

In addition, we test the effect of window size on the portfolio performance, and the experimental results of the portfolio performance with different window sizes and αs are shown in Table 3. No matter what the value of windows size is, the optimal trading strategies are very conservative when the agent only considers the worst-case scenarios. When the window size is large, the agent observes more information for decision-making. Thus she can learn more accurate trends or price movements because the noise and uncertainty of the market can be significantly reduced. For instance, when the window size is 25, the agent is willing to allocate more investment funds into risky assets instead of only holding cash even for the extreme case of risk parameter $\alpha = 5\%$. Therefore, the accumulated return under window size of twenty-five-day is higher than in other window sizes when the risk parameter α is 5%. Overall, these experimental results provide strong evidence to demonstrate that the distributional DDPG method is an effective and efficient way to avoid a huge potential loss. However, we find it hard to balance the total profit and portfolio risk. If the agent only considers the worst-case scenario, the portfolio will lose the potential gains; on the other hand, if the agent is willing to take more risk, the agent has to face large possible losses.

6.3.3. *The experimental results for Hierarchical DDPG*

Fig. 11 shows the Hierarchical DDPG portfolio with different CVaR constraints under the window size of ten-day. We obtain that the

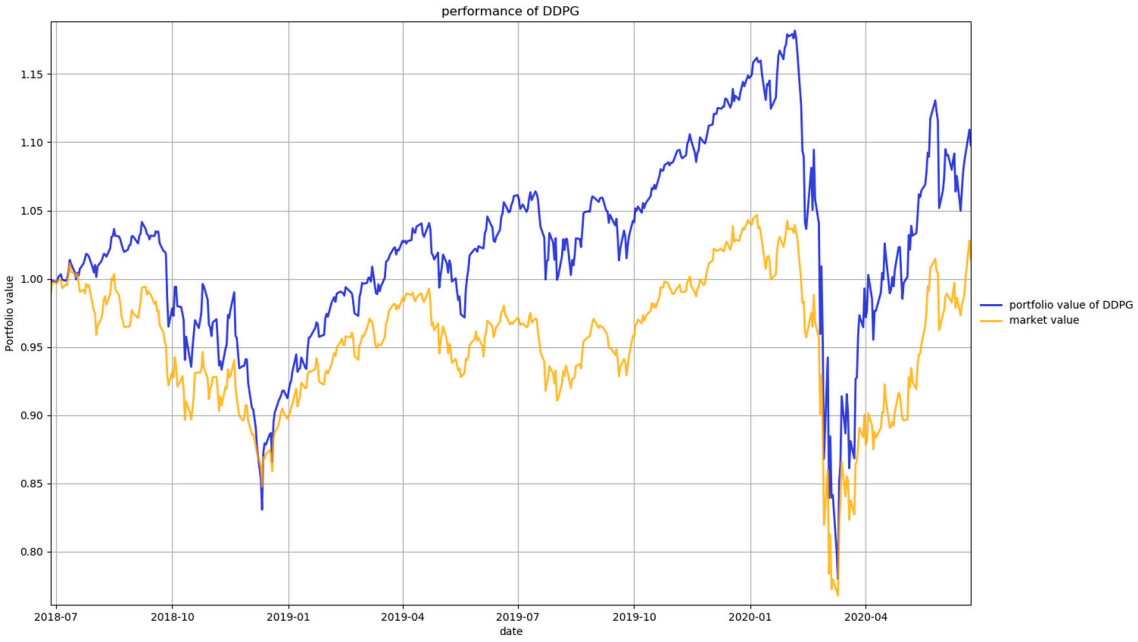

Fig. 9. The portfolio value of DDPG under window size of ten-day during the testing period.

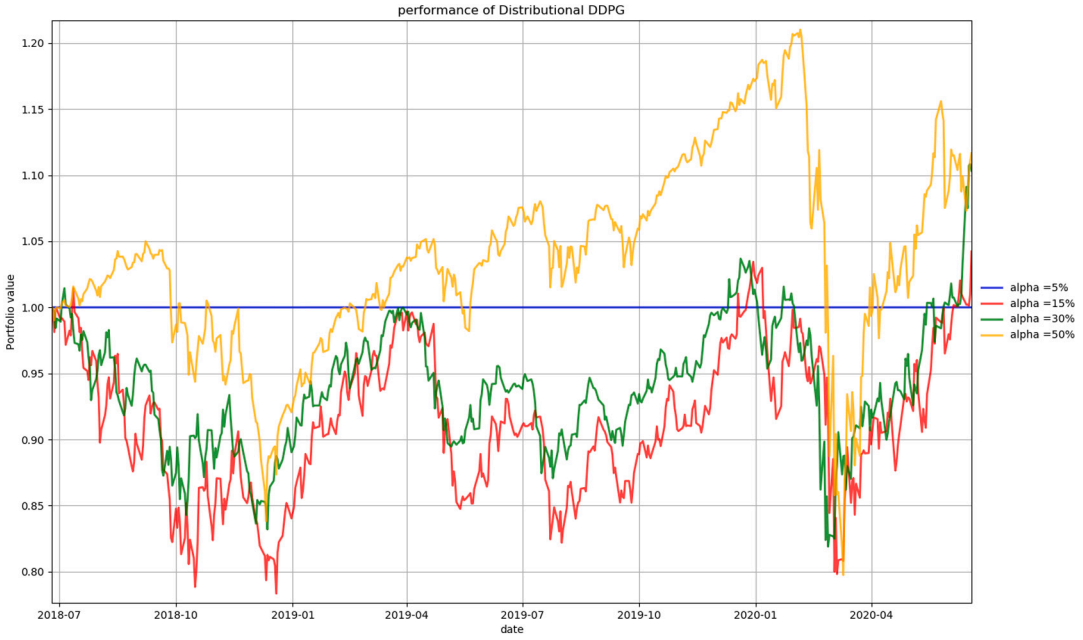

Fig. 10. The portfolio value of Distributional DDPG with different risk parameters α under window size of ten-day.

maximum drawdown has decreased in all cases, e.g., when the CVaR constraint is 5%, the maximum drawdown is 10.05%, reduced by 3.54% compared to classical DDPG. We observe that the accumulated rate of return and Sharpe ratio of Hierarchical DDPG have improved by 4.57% and 32.91%, respectively. Also, the case of the CVaR constraint $C = 13\%$ provides the most significant maximum drawdown of 12.95%, and the case of $C = 8\%$ provides the highest Sharpe ratio of 76.83%. These shreds of evidence illustrate that the Hierarchical DDPG model is superior to DDPG to avoid losses in the recession market. Table 3 shows the experimental results of Hierarchical DDPG with different CVaR constraints and window sizes. Compared to classical DDPG, it reveals that the Hierarchical DDPG algorithm performs better than the classical DDPG method in perspective of the maximum drawdown in most cases except the cases that window size is 20 or 25, and constraint C is 8%. The accumulated return and Sharpe ratio of Hierarchical DDPG are higher than those of classical DDPG in many cases. For example, when the window size is 5, the accumulated rates of return with different constraints $(5\%, 8\%, 13\%)$ have improved by $1.77\%, 6.2\%$, and 1.77%, respectively. These experimental results demonstrate that our proposed Hierarchical DDPG provides stable results with different window sizes and constraints. Overall, the Hierarchical DDPG agent can achieve a higher rate of return and Sharpe ratio compared to classical DDPG, and moreover, control the short-term risk within a reasonable range.

In Fig. 12, we compare the performance of classical DDPG and the proposed approaches during the testing period. For the window size of ten-day, we display the case of CVaR constraint $C = 5\%$ as an example to present the performance of Hierarchical DDPG, and the case of risk parameter $\alpha = 30\%$ as an example to present the

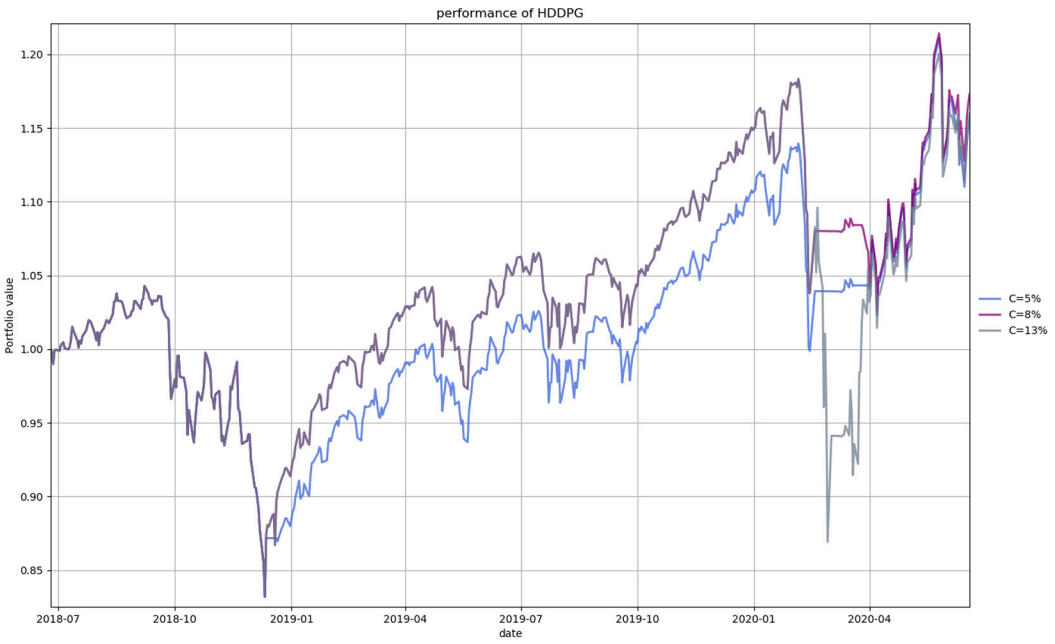

Fig. 11. The portfolio value of Hierarchical DDPG with different constraints C under window size of ten-day.

Table 3
Experimental results.

Window size	DDPG			Distributional DDPG				Hierarchical DDPG			
	AR	MDD	SR	α	AR	MDD	SR	C	AR	MDD	SR
5	11.10%	13.59%	38.45%	5%	0.00%	0.00%	0.00%	5%	12.88%	10.30%	35.68%
				15%	2.41%	9.88%	17.71%	8%	17.31%	11.70%	51.24%
				30%	11.18%	13.59%	33.76%	13%	12.88%	11.70%	38.98%
				50%	12.70%	13.59%	36.52%				
10	10.50%	13.58%	36.47%	5%	0.00%	0.00%	0.00%	5%	15.07%	10.05%	69.38%
				15%	4.22%	6.88%	21.10%	8%	17.30%	11.68%	76.83%
				30%	10.32%	9.68%	34.05%	13%	16.00%	12.95%	58.14%
				50%	11.67%	13.59%	34.61%				
20	12.93%	13.59%	38.44%	5%	0.00%	0.00%	0.00%	5%	21.18%	10.30%	66.14%
				15%	8.37%	9.95%	28.58%	8%	2.70%	14.23%	15.24%
				30%	13.96%	13.59%	38.55%	13%	4.70%	13.28%	21.88%
				50%	13.96%	13.59%	38.55%				
25	14.74%	13.59%	39.94%	5%	8.37%	9.95%	28.58%	5%	8.60%	12.46%	32.37%
				15%	13.77%	13.59%	38.23%	8%	−0.31%	14.23%	9.30%
				30%	13.77%	13.59%	38.26%	13%	13.72%	13.59%	38.14%
				50%	13.77%	13.59%	38.26%				

AR represents the accumulated return.
MDD represents the maximum drawdown.
SR represents the Sharpe ratio.

performance of Distributional DDPG. As shown in Fig. 12, we can see that these three approaches outperform the market value. Hierarchical DDPG provides the highest accumulated return and the lowest maximum drawdown compared to classical DDPG and Distributional DDPG. Specifically, the maximum drawdown drops from 13.59% to 10.05%, and the accumulated return rises from 10.5% to 15.07%. It illustrates that Hierarchical DDPG is an effective approach to avoid a huge loss caused by the financial crisis. Obviously, Distributional DDPG has a lower maximum drawdown and a higher accumulated return compared to the classical DDPG method. On the other hand, Fig. 13 shows the portfolio risk of our approaches. In this paper, we apply parametric CVaR as a risk measure for evaluating portfolio risk. It shows that the portfolio risk of Hierarchical DDPG keeps lower than the CVaR constraint, and Distributional DDPG has a lower portfolio risk than classical DDPG.

In summary, the results demonstrate that Hierarchical DDPG and Distributional DDPG perform better than the classical DDPG algorithm for the rare occurrences of catastrophic events, and they provide the capability to protect the investor who may suffer a massive loss in the recession market. Furthermore, Hierarchical DDPG appears to be a better approach to balance the portfolio risk and portfolio profit compared to Distributional DDPG, which provides a higher return and a lower portfolio risk or maximum drawdown.

6.3.4. Model validation with additional dataset

In this subsection, we validate our approaches by applying four different stocks from the U.S. stock market, that is, “AMZN”, “CCL”, “CVX”, and “LUV”. AMZN represents Amazon.com Inc, one of the world’s largest e-commerce companies headquartered in Seattle, which focuses on cloud computing, digital streaming, and artificial intelligence. CCL stands for Carnival Corporation, the world’s leading leisure travel company that offers extraordinary vacations to travelers around the world. CVX stands for Chevron Corporation, one of the world’s

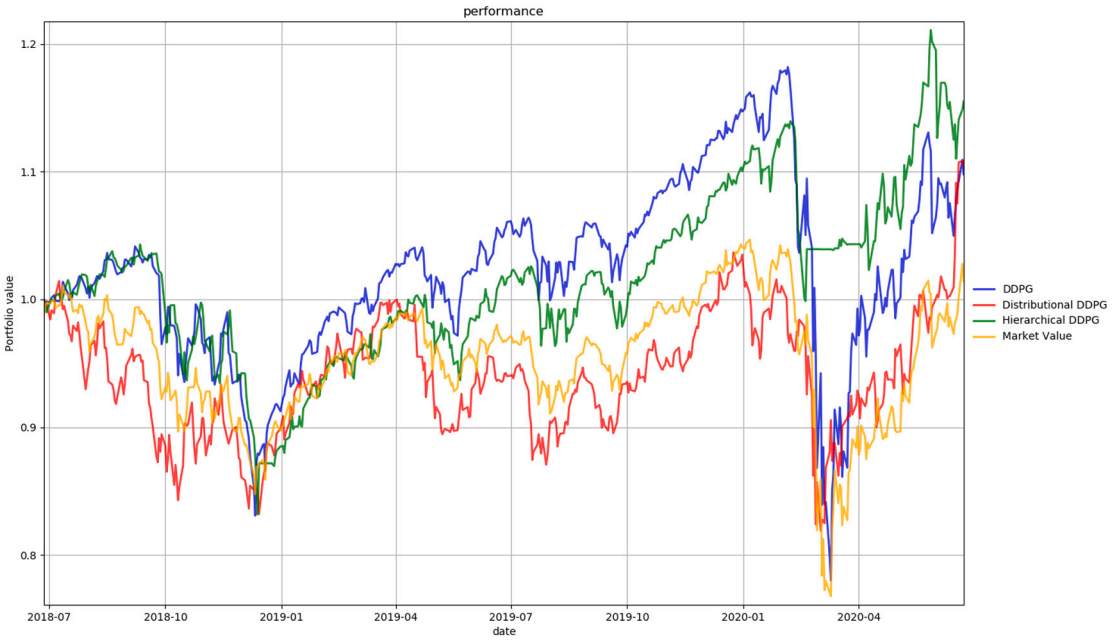

Fig. 12. The performance comparison of Hierarchical DDPG, Distributional DDPG, and classical DDPG under window size of ten-day. Among them, Hierarchical DDPG represents the case of CVaR constraint $C = 5\%$, Distributional DDPG represents the case of risk parameter $\alpha = 30\%$.

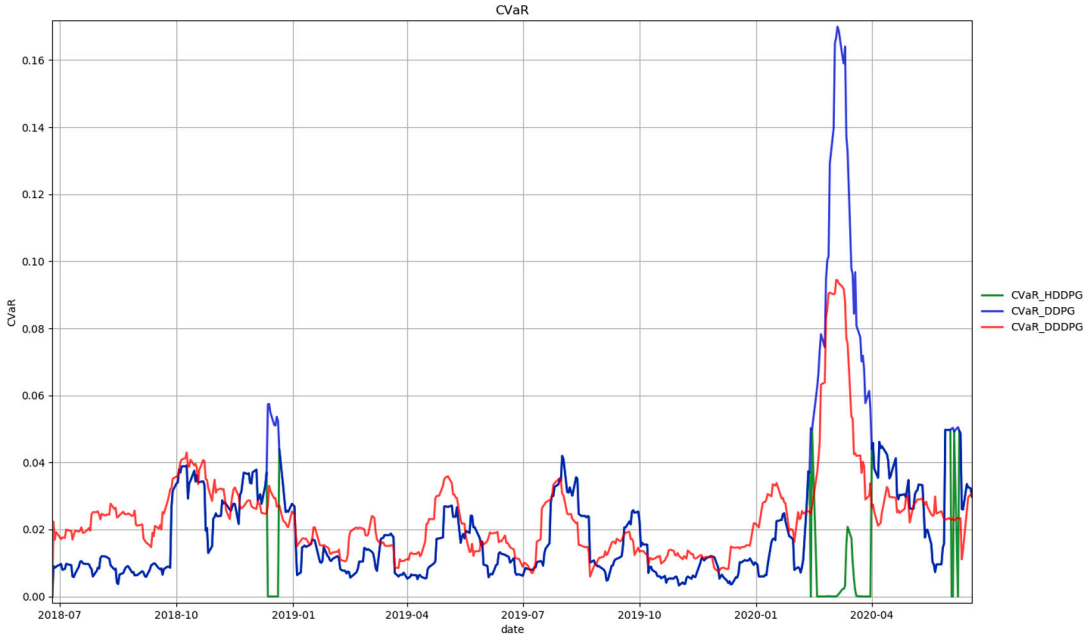

Fig. 13. The portfolio risk comparison of Hierarchical DDPG, Distributional DDPG, and classical DDPG under window size of ten-day. Among them, Hierarchical DDPG represents the case of CVaR constraint $C = 5\%$, Distributional DDPG represents the case of risk parameter $\alpha = 30\%$.

largest energy companies, which operates in integrated energy, chemicals, and petroleum operations in more than 180 countries worldwide. LUV typically refers to as Southwest Airlines Co., the world's largest low-cost airline offering cheaper air transportation in the United States. The data is collected from Yahoo Finance, and consists of daily prices and volumes from 2010-01-01 to 2020-07-30, the same as the period of the ETF indexes. Then, the data is split into training and testing sets in the ratio of 8:2. The data preprocessing is implemented similarly as in Section 6.1.

As shown in Table 4, we obtain that the maximum drawdown of Distributional DDPG has significantly decreased for different window sizes compared to classical DDPG. Also, we observe that the accumulated return and Sharpe ratio of Distributional DDPG tend to increase as α increases in most circumstances. This is not always the case though. That is because when α is small, the Distributional DDPG

Table 4
Experimental results for additional dataset.

Window size	DDPG			Distributional DDPG				Hierarchical DDPG			
	AR	MDD	SR	α	AR	MDD	SR	C	AR	MDD	SR
5	7.15%	13.99%	26.44%	5%	0.00%	0.00%	0.00%	5%	12.99%	9.05%	41.39%
				15%	1.26%	7.81%	5.15%	8%	19.13%	9.05%	54.63%
				30%	6.19%	11.60%	24.86%	13%	21.88%	8.83%	59.45%
				50%	8.69%	12.97%	29.35%				
10	15.29%	15.73%	39.37%	5%	0.00%	0.00%	0.00%	5%	30.71%	8.26%	76.01%
				15%	1.19%	1.12%	25.05%	8%	12.13%	11.97%	38.44%
				30%	9.46%	13.62%	30.44%	13%	28.16%	14.16%	63.48%
				50%	27.18%	14.25%	56.97%				
20	16.90%	19.41%	30.17%	5%	0.00%	0.00%	0.00%	5%	27.27%	12.83%	66.68%
				15%	1.83%	8.66%	9.83%	8%	14.64%	12.97%	40.87%
				30%	21.85%	14.14%	49.62%	13%	20.11%	14.46%	48.82%
				50%	16.18%	16.34%	41.12%				
25	22.38%	16.34%	36.32%	5%	1.44%	0.41%	53.94%	5%	4.12%	10.69%	20.79%
				15%	5.61%	13.25%	23.58%	8%	35.33%	14.52%	66.98%
				30%	27.35%	14.16%	60.19%	13%	14.35%	16.13%	38.42%
				50%	37.53%	14.94%	63.38%				

AR represents the accumulated return.
MDD represents the maximum drawdown.
SR represents the Sharpe ratio.

agent examines the worst-case scenarios and constructs an extremely conservative trading strategy, but as α increases, the agent takes on more risk to gain profit and, as a result, could suffer a larger loss in the market crash compared to the small value of α. Table 4 illustrates that Hierarchical DDPG outperforms classical DDPG in perspective of the maximum drawdown. Also, the Hierarchical DDPG algorithm can achieve a higher accumulated return and Sharpe ratio in many cases. For example, when the window size is 20 or 25, we can observe that the maximum drawdown of Hierarchical DDPG for both cases has significantly decreased, and the Sharpe ratio has increased in most cases in comparison with the classical DDPG method. We note that it is crucial to choose the appropriate window size and risk tolerance parameters for the superior performance of Hierarchical DDPG.

7. Conclusion

Portfolio management has always been a crucial topic in the financial field, which allocates investment funds in a group of assets to gain the maximum return of investors. It is a challenging task to construct a trading policy in the financial market because it requires professional knowledge in several fields, such as quantitative finance and risk management. The Deep RL algorithms can provide a more effective way to construct trading policies. Although Deep RL has achieved remarkable performance in portfolio management problems, most of the existing methods have not considered the worst-case scenarios in constructing trading policy. In this paper, we propose two novel approaches, Hierarchical DDPG and Distributional DDPG to address this issue.

Hierarchical DDPG was developed based on DDPG and enhanced with a risk indicator CVaR. Hierarchical DDPG enables alternative trading strategies depending on the level of risk tolerance, and also its performance is determined by the risk indicator's ability to detect possible risk in a timely manner. In general, Hierarchical DDPG can deliver better results for portfolios with a greater risk tolerance, while lower levels of risk tolerance may lead to higher transaction costs. On the other hand, Distributional DDPG was modeled based on the Gaussian distribution, which only takes into account the bottom α-*percentile* distribution. Under the worst-case scenarios, Distributional DDPG will provide more steady and cautious trading strategies for risk-averse investors.

To validate the applicability of the proposed learning analytics methods, a back-test is carried out on the real-world stocks from the U.S. financial market. Our study illustrates the superior performance of Hierarchical DDPG. It is impressive that the Hierarchical DDPG agent can not only maximize the portfolio profit but also keep the portfolio risk below a certain level of risk, which produces a portfolio with higher return and lower risk. Also, the experiment results reveal that Distributional DDPG produces risk-sensitive policies to reduce the effects of disaster events depending on the risk parameter. When the risk parameter α is small, the agent optimizes the performance for the worst-case scenario, which provides a conservative trading strategy. In contrast, when the risk parameter α is large, the agent is more willing to select an aggressive trading strategy. We can conclude that our Hierarchical DDPG and Distributional DDPG models outperform the classical DDPG method in the sense that they provide risk-sensitive strategies that protect investors who may suffer a huge loss caused by rare disaster events. Our proposed approaches provide effective methods to learn a risk-sensitive solution for the portfolio optimization problem.

The limitation of this work is that the Distributional DDPG method is developed based on the assumption of the returns distribution that leads to a closed-form of calculation for the objective function. In addition, it is important to select the appropriate window size and risk tolerance parameters as needed for superior performance of the proposed models. For future research, we may involve textual data such as news or tweets, to improve the performance of DDPG, Hierarchical DDPG, and Distributional DDPG.

CRediT authorship contribution statement

Mingfu Wang: Data curation, Investigation, Methodology, Visualization, Writing – original draft. **Hyejin Ku:** Conceptualization, Formal Analysis, Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research was partly supported by Natural Sciences and Engineering Research Council of Canada, Discovery grant 504316.

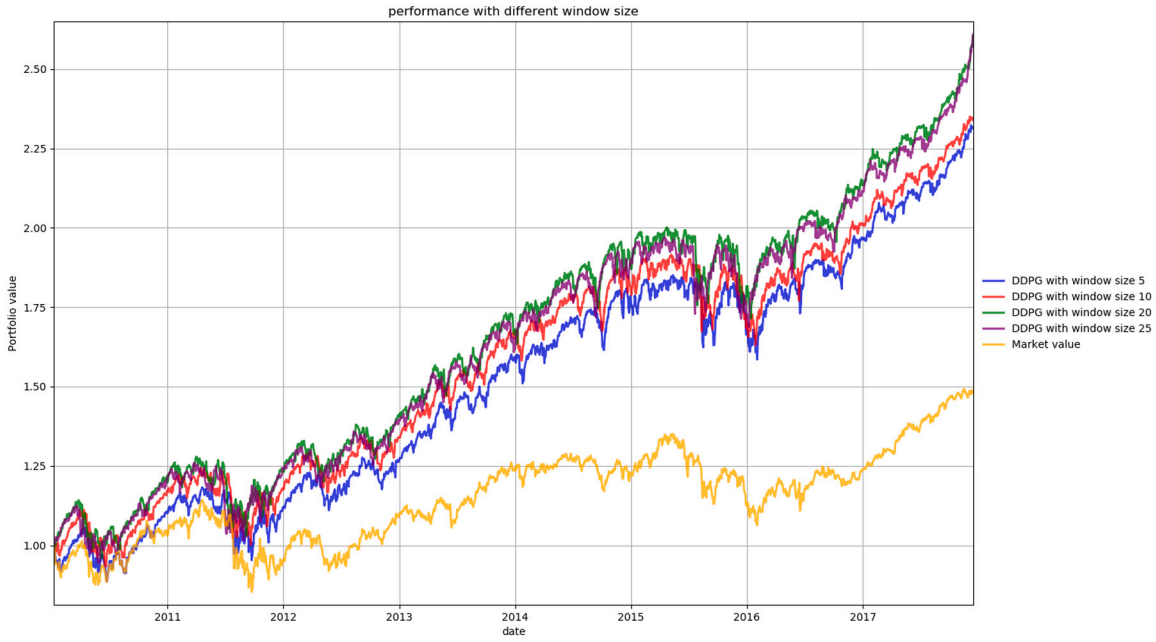

Fig. 14. The portfolio values of classical DDPG with different window sizes during the training period.

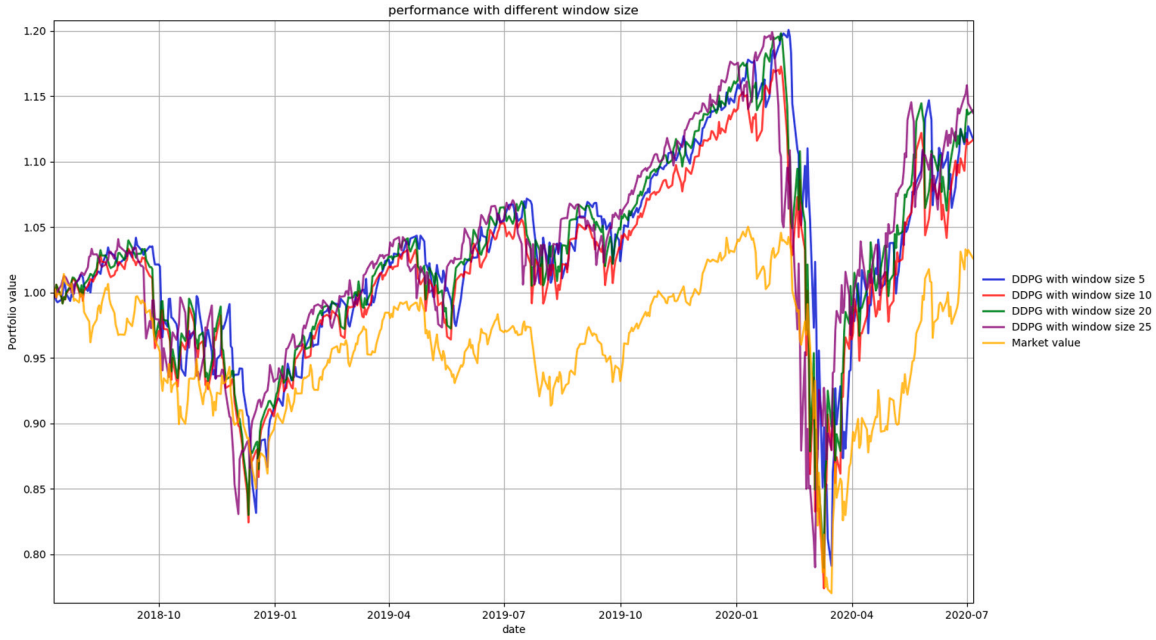

Fig. 15. The portfolio values of classical DDPG with different window sizes during the testing period.

Appendix A. Convergence property of our distributional DDPG model

Let us make use of the probability space $(\Omega, \mathcal{F}, P)$ and view value distributions as random vectors with finite moments in $\mathbb{R}^{S\times\mathcal{A}}$ as usual in the distributional RL model. Consider the process $Z_{k+1} := \mathcal{T}^{\mu} Z_k$ starting with some Z_0. The distributional Bellman operator T^{μ} is a contraction mapping whose unique fixed point is the random return Z^{μ} using the 2-Wasserstein distance. Then, the sequence $\{Z_k\}$ converges to Z^{μ} in distribution. However, this does not necessarily mean pointwise convergence of the sequence $\{Z_k\}$ to Z^{μ}. Bellemare et al. (2017) mention that all moments also converge, in particular $\mathbb{E}[Z_k]$ converges, but one cannot directly use the Wasserstein metric to get the variance convergence.

Let $\mathcal{Q}_k$ and $\mathcal{V}_k$ be the mean and variance of Z_k. Then we have the following convergence results for the proposed distributional DDPG model.

Lemma A.1. *For the sequences of mean and variance of Z_k, the following inequalities hold for $k = 1, 2, \ldots$*

$$\|\mathcal{Q}_{k+1} - \mathcal{Q}_k\|_{\infty} \le \gamma \|\mathcal{Q}_k - \mathcal{Q}_{k-1}\|_{\infty},$$
$$\|\mathcal{V}_{k+1} - \mathcal{V}_k\|_{\infty} \le \gamma^2 \|\mathcal{V}_k - \mathcal{V}_{k-1}\|_{\infty},$$

where γ is the discount factor.

Proof. Since $\mathcal{Q}_{k+1} = \mathbb{E}[Z_{k+1}] = \mathbb{E}[\mathcal{T}^{\mu} Z_k]$, we have

$$\|\mathcal{Q}_{k+1} - \mathcal{Q}_k\|_{\infty} = \|\mathbb{E}[\mathcal{T}^{\mu} Z_k] - \mathbb{E}[\mathcal{T}^{\mu} Z_{k-1}]\|_{\infty}$$

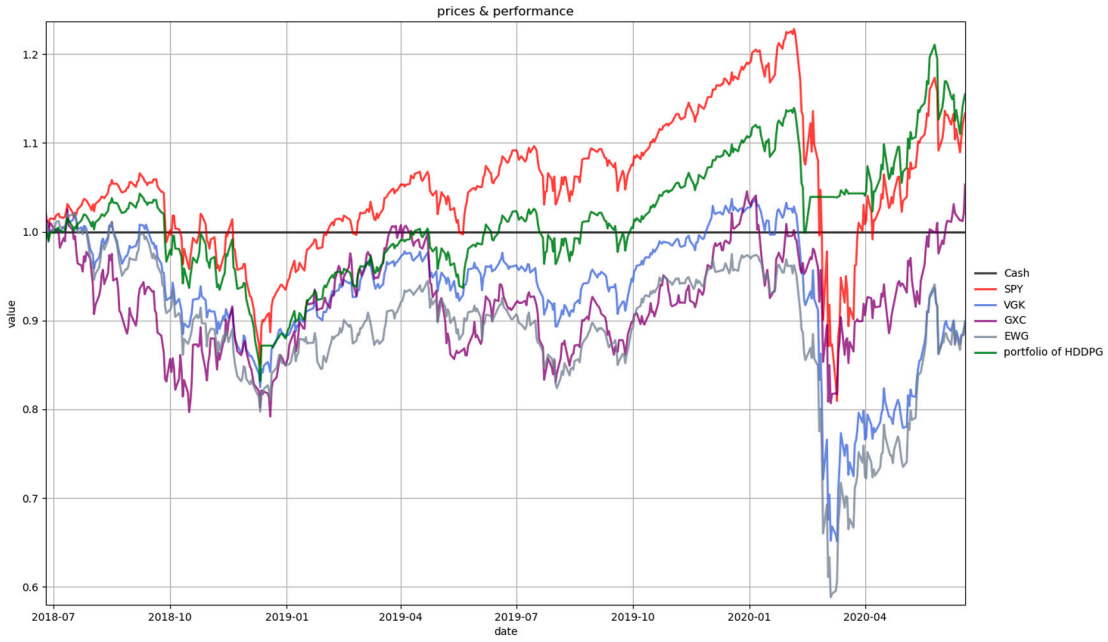

Fig. 16. The price movements of each stock and portfolio values of Hierarchical DDPG with window size of ten-day and CVaR constraint $C = 5\%$.

$$
\begin{aligned}
&= \sup_{s,a} \gamma |\mathbb{E}[P^{\mu} Z_k(s,a)] - \mathbb{E}[P^{\mu} Z_{k-1}(s,a)]| \\
&= \sup_{s,a} \gamma |\mathbb{E}[Z_k(s',a')] - \mathbb{E}[Z_{k-1}(s',a')]| \quad (s' \sim p(\cdot|s,a), a' \sim \mu(\cdot|s)) \\
&\leq \sup_{s',a'} \gamma |\mathbb{E}[Z_k(s',a')] - \mathbb{E}[Z_{k-1}(s',a')]| \\
&= \gamma \|\mathbb{E}[Z_k] - \mathbb{E}[Z_{k-1}]\|_{\infty} = \gamma \|\mathcal{Q}_k - \mathcal{Q}_{k-1}\|_{\infty}.
\end{aligned}
$$

Also, since $r(s,a)$ and $P^{\mu} Z_k(s,a)$ are independent, we get

$$
\begin{aligned}
\|\mathcal{V}_{k+1} - \mathcal{V}_k\|_{\infty} &= \|\mathrm{Var}(\mathcal{T}^{\mu} Z_k) - \mathrm{Var}(\mathcal{T}^{\mu} Z_{k-1})\|_{\infty} \\
&= \sup_{s,a} \gamma^2 |\mathrm{Var}(P^{\mu} Z_k(s,a)) - \mathrm{Var}(P^{\mu} Z_{k-1}(s,a))| \\
&\leq \sup_{s',a'} \gamma^2 |\mathrm{Var}(Z_k(s',a')) - \mathrm{Var}(Z_{k-1}(s',a'))| \\
&= \gamma^2 \|\mathrm{Var}(Z_k) - \mathrm{Var}(Z_{k-1})\|_{\infty} = \gamma^2 \|\mathcal{V}_k - \mathcal{V}_{k-1}\|_{\infty}. \quad \square
\end{aligned}
$$

Lemma A.2. *With the discount factor $\gamma < 1$, $\{\mathcal{Q}_k\}$ and $\{\mathcal{V}_k\}$ are Cauchy sequences in L^{∞}.*

Proof. We need to show that for every positive $\epsilon > 0$, there is a positive integer N such that for every $m, n > N$, $\|\mathcal{V}_m - \mathcal{V}_n\|_{\infty} < \epsilon$. Without loss of generality, we assume $\|\mathcal{V}_1 - \mathcal{V}_0\|_{\infty} \leq 1$.

By Lemma A.1, we have

$$
\|\mathcal{V}_2 - \mathcal{V}_1\|_{\infty} \leq \gamma^2, \ \|\mathcal{V}_3 - \mathcal{V}_2\|_{\infty} \leq \gamma^4, \ \ldots, \ \|\mathcal{V}_{k+1} - \mathcal{V}_k\|_{\infty} \leq \gamma^{2k}.
$$

Also, for $m > n$,

$$
\begin{aligned}
\|\mathcal{V}_m - \mathcal{V}_n\|_{\infty} &\leq \|\mathcal{V}_m - \mathcal{V}_{m-1}\|_{\infty} + \cdots + \|\mathcal{V}_{n+1} - \mathcal{V}_n\|_{\infty} \\
&\leq \gamma^{2(m-1)} + \cdots + \gamma^{2n} = \frac{(1 - \gamma^{2(m-n)})}{1 - \gamma^2} \gamma^{2n} \\
&\leq \left(\frac{1}{1 - \gamma^2}\right) \gamma^{2n}.
\end{aligned}
$$

Therefore, we can find a large N such that $\gamma^{2N} < \epsilon(1 - \gamma^2)$ for every given $\epsilon > 0$. The result for sequence $\{\mathcal{Q}_k\}$ can be obtained by following essentially the same steps. The proof is completed. $\square$

Theorem A.3. *The sequences of $\{\mathcal{Q}_k\}$ and $\{\mathcal{V}_k\}$ converge pointwise to their limits in the critic network for policy evaluation.*

Table 5
Hyperparameters of our proposed model.

Parameter	DDPG	Distributional DDPG	Hierarchical DDPG
Batch size	64	32	64
Steps	128	128	128
Episode	3000	5000	5000
Trading period	1 day	1 day	1 day
Learning rate of actor	10^{-5}	10^{-5}	10^{-5}
Learning rate of critic	10^{-4}	10^{-4}	10^{-4}
Regularization rate	0.001	0.001	0.001
Discount rate	0.99	0.99	0.99
Memory size	10^6	10^6	10^6
Number of layer of actor	5	4	5
Number of layer of critic	4	5	4
Activation function of actor	Relu	Relu	Relu
Activation function of critic	Relu	Relu, Softplus	Relu
Training set portion	0.8	0.8	0.8
Test set portion	0.2	0.2	0.2
Commission rate	0.25%	0.25%	0.25%

Proof. By combining the fact that every Cauchy sequence converges in L^{∞} to the limit and Lemma A.2, we conclude that the limits of $\{\mathcal{Q}_k\}$ and $\{\mathcal{V}_k\}$ exist and the sequences converge in L^{∞}. This implies that the convergence takes place for all sample transitions by repeated applications of the distributional Bellman operator $\mathcal{T}^{\mu}$. $\square$

Appendix B. Experimental results and parameters settings

See Figs. 14–16 and Table 5.

References

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

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Tb, D., et al. (2018). Distributed distributional deterministic policy gradients. arXiv preprint arXiv:1804.08617.

Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems, 13*(1), 41–77.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A distributional perspective on reinforcement learning. *International Conference on Machine Learning*, 449–458.

Bertoluzzo, F., & Corazza, M. (2012). *Reinforcement learning for automatic financial trading: Introduction and some applications*: *University Ca'Foscari of Venice, Dept. of Economics Research Paper Series No. 33*.

Bickel, P. J., & Freedman, D. A. (1981). Some asymptotic theory for the bootstrap. *The Annals of Statistics*, 1196–1217.

Bielecki, T. R., & Pliska, S. R. (1999). Risk-sensitive dynamic asset management. *Applied Mathematics and Optimization*, *39*(3), 337–360.

Chen, L., & Gao, Q. (2019). Application of deep reinforcement learning on automated stock trading. In *2019 IEEE 10th international conference on software engineering and service science* (pp. 29–33). IEEE.

Dayan, P. (2002). Reinforcement learning. In *Stevens' Handbook of Experimental Psychology*. Wiley Online Library.

Dearden, R., Friedman, N., & Russell, S. J. (1998). BayesIan Q-learning. In *AAAI/IAAI*.

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In *ICML, vol. 98* (pp. 118–126). Citeseer.

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd International conference on machine learning* (pp. 201–208).

Fleming, W. H., & Sheu, S. (2000). Risk-sensitive control and an optimal investment model. *Mathematical Finance*, *10*(2), 197–213.

Fleming, W. H., & Sheu, S. (2002). Risk-sensitive control and an optimal investment model II. *Annals of Applied Probability*, *12*(2), 730–767.

Gao, Z., Gao, Y., Hu, Y., Jiang, Z., & Su, J. (2020). Application of deep q-network in portfolio management. In *2020 5th IEEE international conference on big data analytics* (pp. 268–275). IEEE.

Hansen, L. P., & Sargent, T. J. (1995). Discounted linear exponential quadratic gaussian control. *IEEE Transactions on Automatic Control*, *40*(5), 968–971.

Hansen, L. P., & Sargent, T. J. (2001). Robust control and model uncertainty. *American Economic Review*, *91*(2), 60–66.

Hegde, S., Kumar, V., & Singh, A. (2018). Risk aware portfolio construction using deep deterministic policy gradients. In *2018 IEEE symposium series on computational intelligence* (pp. 1861–1867). IEEE.

Jeong, G., & Kim, H. Y. (2019). Improving financial trading decisions using deep Q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications*, *117*, 125–138.

Jiang, Z., & Liang, J. (2017). Cryptocurrency portfolio management with deep reinforcement learning. In *2017 Intelligent systems conference* (pp. 905–913). IEEE.

Krokhmal, P., Palmquist, J., & Uryasev, S. (2002). Portfolio optimization with conditional value-at-risk objective and constraints. *Journal of Risk*, *4*, 43–68.

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

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, *7*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, *12*(4), 875–889.

Moody, J., Wu, L., Liao, Y., & Saffell, M. (1998). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, *17*(5–6), 441–470.

Nachum, O., Gu, S., Lee, H., & Levine, S. (2018). Data-efficient hierarchical reinforcement learning. arXiv preprint arXiv:1805.08296.

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

Olkin, I., & Pukelsheim, F. (1982). The distance between two random vectors with given dispersion matrices. *Linear Algebra and its Applications*, *48*, 257–263.

Ormoneit, D., & Glynn, P. (2002). Kernel-based reinforcement learning in average-cost problems. *IEEE Transactions on Automatic Control*, *47*(10), 1624–1636.

Park, H., Sim, M. K., & Choi, D. G. (2020). An intelligent financial portfolio trading strategy using deep Q-learning. *Expert Systems with Applications*, *158*, Article 113573.

Pendharkar, P. C., & Cusatis, P. (2018). Trading financial indices with reinforcement learning agents. *Expert Systems with Applications*, *103*, 1–13.

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

Rowland, M., Bellemare, M., Dabney, W., Munos, R., & Teh, Y. W. (2018). An analysis of categorical distributional reinforcement learning. *International Conference on Artificial Intelligence and Statistics*, 29–37.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning* (pp. 387–395). PMLR.

Tang, Y. C., Zhang, J., & Salakhutdinov, R. (2020). Worst cases policy gradients. *Conference on Robot Learning*, 1078–1093.

Wu, J., & Li, H. (2020). Deep ensemble reinforcement learning with multiple deep deterministic policy gradient algorithm. *Mathematical Problems in Engineering*, *2020*, 1–12.

Xiong, Z., Liu, X.-Y., Zhong, S., Yang, H., & Walid, A. (2018). Practical deep reinforcement learning approach for stock trading. arXiv preprint arXiv:1811.07522.