axioms

Article

A Selective Portfolio Management Algorithm with Off-Policy Reinforcement Learning Using Dirichlet Distribution

Hyunjun Yang, Hyeonjun Park and Kyungjae Lee *

Department of Artificial Intelligence, Chung-Ang University, Seoul 06910, Republic of Korea
* Correspondence: kyungjae.lee@ai.cau.ac.kr; Tel.: +82-02-820-5371

Abstract: Existing methods in portfolio management deterministically produce an optimal portfolio. However, according to modern portfolio theory, there exists a trade-off between a portfolio's expected returns and risks. Therefore, the optimal portfolio does not exist definitively, but several exist, and using only one deterministic portfolio is disadvantageous for risk management. We proposed Dirichlet Distribution Trader (DDT), an algorithm that calculates multiple optimal portfolios by taking Dirichlet Distribution as a policy. The DDT algorithm makes several optimal portfolios according to risk levels. In addition, by obtaining the pi value from the distribution and applying importance sampling to off-policy learning, the sample is used efficiently. Furthermore, the architecture of our model is scalable because the feed-forward of information between portfolio stocks occurs independently. This means that even if untrained stocks are added to the portfolio, the optimal weight can be adjusted. We also conducted three experiments. In the scalability experiment, it was shown that the DDT extended model, which is trained with only three stocks, had little difference in performance from the DDT model that learned all the stocks in the portfolio. In an experiment comparing the off-policy algorithm and the on-policy algorithm, it was shown that the off-policy algorithm had good performance regardless of the stock price trend. In an experiment comparing investment results according to risk level, it was shown that a higher return or a better Sharpe ratio could be obtained through risk control.

Keywords: deep reinforcement learning; exploration methods; portfolio optimization

check for updates

Citation: Yang, H.; Park, H.; Lee, K. A Selective Portfolio Management Algorithm with Off-Policy Reinforcement Learning Using Dirichlet Distribution. *Axioms* **2022**, *11*, 664. https://doi.org/10.3390/axioms11120664

Academic Editors: Zhou Zhou, Xin-Jiang He, Xiang Yu and Bin Zou

Received: 29 October 2022
Accepted: 18 November 2022
Published: 23 November 2022

1. Introduction

Financial Portfolio Management (PM) is a problem of sequentially allocating and balancing a number of funds into several risky financial assets such as stocks, bonds, or cash. The goal of portfolio management is to maximize the expected return while minimizing investment risks. Traditional portfolio management has determined the weights of financial assets based on statistical indicators. Specifically, there is a well-known portfolio optimization method called modern portfolio theory (MPT) [1]. MPT allows an efficient portfolio to be computed by using the relationship between the expected return and risk, where the risk is measured by the standard deviation of returns. While MPT provides a theoretically interpretable portfolio, it has a limitation in that a number of statistical features, which can be obtained from big data, cannot be considered [1]. Furthermore, MPT highly depends on the accurate prediction of the expected return and its standard deviation [1]. To overcome such limitations, recent studies have employed deep learning-based approaches that can automatically extract meaningful features from high-dimensional big data. In particular, applying deep reinforcement learning (Deep RL) methods to solve the portfolio management problem has continuously received attention since deep RL has the ability to learn sequential decision-making based on features extracted from deep neural networks. Hence, the importance of deep RL algorithms is being emphasized since portfolio management problems can be formulated as the sequential determination of the weight of each financial asset.

In this regard, several existing off-policy and on-policy deep RL methods have been applied to PM problems [2,3]. For off-policy RL, Ref [2] applied deep Q-learning to manage the portfolio of a fixed number of assets. In general, Q learning is an off-policy approach that can reuse experiences sampled during a training phase. Hence, off-policy approaches are usually more sample efficient than on-policy methods. However, the deep Q learning method used in [2] has a clear limitation in scalability since the architecture of the Q network is fixed depending on the number of assets, and, hence, it cannot function if the number of assets is extended. In PM, scalability means that the network can infer the weight of assets that have not been used for training. If the network is not scalable for the number of assets, it is inefficient since it should be re-trained every time a new stock is added.

To alleviate this issue, [3] applied proximal policy optimization [4], which is a well-known on-policy method, and they proposed a new architecture that can handle a dynamic number of financial assets. The fact that the model has scalability gives variety in stock selection and facilitates trading strategy construction, new assets can be added to our portfolio, or existing assets can be excluded from the portfolio. While the proposed model in [3] is scalable, it has a drawback in that on-policy methods have been mainly employed to train the network. In on-policy algorithms, the samples used to update the agent cannot be reused after updating the parameter, which often hampers sample inefficiency. Hence, it is essential to develop an off-policy algorithm to improve sample efficiency.

Furthermore, the previous models often consider a deterministic policy that can decide on only one optimal portfolio vector [2,3]. However, designing a deterministic policy is not robust to a variety of situations. If the policy network is stochastic, the agent can have more than one optimal action, and we can have portfolio options according to the expected rate of return and risk. In this regard, it is inevitable to develop scalable and multi-modal methods for PM to consider a dynamic number of assets in the test phase.

Hence, we propose Dirichlet Distribution Trader (DDT), which can incorporate both sample efficiency and scalability. To increase sample efficiency, we apply the off-policy learning algorithm to train the scalable network. In particular, we use an importance sampling technique to implement the off-policy algorithm. To compute importance weights, we define a distribution over the portfolio vector. Since the valid portfolio vector consists of the simplex, the Dirichlet distribution, which is a distribution on the simplex, is employed, while previous studies only employ a deterministic policy instead of learning a policy distribution. Furthermore, learning an optimal distribution of a portfolio has a clear benefit in a test phase. By using a portfolio distribution, we can construct various trading strategies during the test time by sampling the proper portfolio from the optimal distribution. To make the model scalable for a dynamic number of assets, we employ similar network architecture to that proposed in [5], where the information of each asset flowed identically in the network. However, we do not use Recurrent Neural Network, and use MLP for a single time step state.

In the experimental part, companies were selected from the NASDAQ 100 index to find companies with sufficient trading volume. The selected stocks are divided into Up and Down trends, and we construct datasets variously according to the trend with these stocks. After such settings, we compared the proposed model with other DRL algorithms, such as DQN, PPO, and other PM methods, as benchmarks. As a result of the experiments, our proposed model outperforms the existing methods and shows a state-of-the-art performance with respect to profit. Furthermore, we also empirically show the advantages of using the Dirichlet distribution for PM. Therefore, the contributions of our proposed methods can be summarized as follows:

- We propose a model using Dirichlet distribution, which is suitable for PM and off-policy learning.
- We conduct experiments comparing on-policy and off-policy algorithms.
- We conduct experiments with various trend datasets and compare them with other algorithms. As a result, our proposed model outperforms the other algorithms and SOTA.

- The proposed model utilizes representative values of the Dirichlet distribution to allow investors to selectively adopt investment attitudes according to risk propensities such as risk-averse and risk-seeking types.

2. Related Work

For decades, researchers in the financial field have tried to replace human traders with computers [6–8]. In recent years, thanks to the development of machine learning [9,10], stock investment strategies using deep learning have been proposed. These learning-based methods can be categorized into supervised learning or on/off-policy RL methods. Furthermore, we classify learning-based methods under the criterion of whether they are scalable or nonscalable. We present our categorization in Table 1.

As with the cases of natural language processing and computer vision, the financial field has benefited from progress in machine learning. For example, Ref [11] improved the traditional momentum strategy [8] using an autoencoder. Ref [12] analyzed the effectiveness of deep neural network (DNN) within the context of stock investment strategy. Ref [13] implemented long short-term memory networks (LSTM) [9] for price prediction in the financial market and showed that LSTM can outperform memory-free classification models, including DNN. However, in general, these supervised learning-based models for stock market trading strategies are trained to make price predictions. Therefore, additional layers are needed to convert prices into action. Furthermore, training supervised learning models requires binary labels for actions, which are lacking in general cases.

Table 1. Categorization of learning-based portfolio management approaches.

	Scalable	**Not Scalable**
On-Policy RL	[3]	[14]
Off-Policy RL	Ours [3,5,15]	[2,16,17]
Supervised Learning	-	[11–13]

Apart from the supervised learning-based methods, reinforcement learning algorithms have long been attracting the attention of researchers in the financial field. In recent years, deep reinforcement learning (DRL), which combines deep learning as a function approximation for reinforcement learning, has been adopted for many sequential decision-making problems, such as controlling robotic systems [18,19], generating humanoid motions, learning how to play video games [20–23], and learning trading strategies. These DRL models can be divided into two categories, on-policy or off-policy, depending on which algorithm is used for the model. We first introduce recent advances in DRL algorithms and second introduce recent research to apply DRL to portfolio management problems. Lee et al. [18,19] proposed a novel exploration method in DRL by using the generalization of the entropy term. Cetin and Çeliktutan [20] proposed a novel action space, called a routine space, to enable agents to learn effective behavior. Wang et al. [21] extended the classical value iteration method to multi-step RL by proposing a new multi-step Bellman optimality equation, which uses the latent space of multi-step bootstrapping. Kong et al. [22] proposed a new sampling strategy for off-policy RL, which imposes a high sampling probability to recently added data in the replay buffer to mitigate unbalanced exploitation issues between initial data and new data. Vijayan and A. [23] introduced a policy gradient algorithm that employs importance sampling for policy evaluation with a smoothed functional-based gradient estimation. Then, recent research in PM has applied DRL methods to the problem of decision-making on portfolio vectors. On-policy RL algorithms have the target policy as the policy used to make decisions in the stock market. Specifically, models for PM using on-policy RL algorithms mainly use proximal policy optimization (PPO) [4], policy gradient (PG) [24], and A2C algorithms [25]. For example, Ref [14] transplanted modern portfolio theory [26] into an RL framework to make direct mapping between market situation and trade position. In this process, Ref [14] used the on-policy RL algorithms PG and A2C compared to DQN.

On the other hand, the off-policy RL algorithms, whose target policy is different from the behavior policy, have an advantage in sample efficiency. In particular, Deep deterministic policy gradient(DDPG) [27] and deep Q-network(DQN) [28] have been widely adopted as the off-policy RL algorithms for stock market trading. For instance, Ref [16] used DQN with a convolutional neural network(CNN) to take the stock market images as input for making investment strategies. Ref [2] introduced a novel mapping function that can handle infeasible action caused by combinatorial operating spaces and applied DQN to derive PM investment strategies. Ref [17] showed model using DRL can outperform the classical RL algorithm SARSA [29] on stock market trading, and MLP has a better performance than gated recurrent unit(GRU) and CNN on feature extracting the stock market data.

However, both the on-policy and off-policy DRL models mentioned so far did not consider the scalability of the model. Although scalability is needed to handle portfolios of various sizes, only a few studies have considered the scalability of the trading model. For example, Ref [5] introduced a novel DRL structure, which consists of an ensemble of identical independent evaluators (EIIE) topology and portfolio-vector memory (PVM). By using the EIIE, including three networks CNN, RNN, and LSTM, the model is able to deal with multi-channel input, and the portfolio, with scalability. Ref [15] also used IIE to handle portfolios of various sizes. However, this model exploited the softmax function to normalize the output. Similarly, Ref [3] employed IIE for scalability, but its activation function is a Min-Max normalized function. Since both softmax and Min-Max activation functions induce one optimal portfolio vector, the existing models with scalability have a weakness in the diversity of the trading environment. On the contrary, we set the model to learn the distribution of the portfolio, making our model have robustness in a variation of the stock market.

3. Preliminaries

3.1. Financial Portfolio Management

Financial portfolio management is the task of sequentially re-balancing all weights of financial assets by considering future profits. To develop an automatic portfolio management system, in this section, we mathematically formulate the portfolio management problem.

We first define a time step t. The time step t is defined as a time slot from the opening to the closing of a stock market in which a stock can be bought or sold. At time step t, the environment (or the market) provides the agent (or the trading system) with stock data for the portfolio assets from the previous business day. Then, the time step t is divided into two periods: the period before re-balancing the portfolio and the period after re-balancing the portfolio. If our trading agent determines the portfolio based on previous stock data, it makes changes in the portfolio and its value within the time step t. Hence, we can separate the time step t into the period before the agent's action and the term after the agent's action.

At every time step t, an agent determines the portfolio weights based on previous stock data. The portfolio weights can be expressed as a vector of all holding weights of K assets, including both cash and non-cash assets. We denote the portfolio vector as $\mathbf{w}_t$ at the period before the agent's action in time step t.

$$\mathbf{w}_t = [w_{t,1}, w_{t,2}, ..., w_{t,K}]^\mathsf{T} \tag{1}$$

where $w_{t,i}$ indicates the percentage of the ith asset among total funds. The ith entry of $\mathbf{w}_t$ is denoted as $\mathbf{w}_{t,i}$, which means the weight of the ith asset before taking action. Specifically, the first element of portfolio vector, $w_{t,1}$, is a proportion of cash. Then, the portfolio vector changed after taking action at time step t is denoted as $\mathbf{w'}_t$. Finally, the portfolio vector at the initial time step $\mathbf{w}_0$ is $\mathbf{w}_0 = (1, 0, 0, ..., 0)^T$, which consists of cash only.

The goal of the trading agent is to determine the best $\mathbf{w}_t$ based on the previous stock market data, which is encoded as a stock feature vector. The stock feature vector $v_{t,i}$ of the ith asset at time t is defined as

$$v_{t,i} = (f_t^o, f_t^h, f_t^l, f_t^c, f_t^v), \tag{2}$$

where the five features o, h, l, c, and v represent open, high, low, close price, and volume, respectively. Then, stock feature matrix X_t is defined by stacking $v_{t,i}$ for all assets as follows,

$$X_t = [v_{t,1}^\intercal, v_{t,2}^\intercal, \cdots, v_{t,K}^\intercal]^\intercal, \tag{3}$$

where the shape of X_t is $K \times 5$.

To compute the profit of $\mathbf{w}_t$, we employ the concept of the portfolio value, which is an amount of investment evaluated by reflecting the changed stock price and transaction cost. PV_t and PV'_t are denoted as the portfolio value of $\mathbf{w}_t$ and $\mathbf{w'}_t$, respectively. To calculate a portfolio value and a changed portfolio by price fluctuations for every time step, the closing price vector at t is defined as Equation (5).

$$p_t = (p_{t,1}, p_{t,2}, \ldots, p_{t,K})^T \tag{4}$$

In a continuous time sequence from t to $t+1$, two factors affect the portfolio value: cost and closing price fluctuations. In time step t, the agent predicts the desired portfolio $\mathbf{w'}_t$. In order to change the current portfolio $\mathbf{w}_t$ to the desired one, the transaction cost must be paid, which can be computed as follows,

$$c_t = \sum_{i=1}^{K} c \times \tau_{t,i}, \tag{5}$$

where $\tau_{t,i}$ represents a trading amount of ith asset, and c indicates a commission rate.

After executing transactions, we can compute the pseudo portfolio weight as follows,

$$\hat{w}_{t,i} = \begin{cases} w_{t,i} - \frac{\tau_{t,i}}{PV_t} & \text{if } i \in S_t^- \\ w_{t,i} + \frac{\tau_{t,i}}{PV_t} & \text{if } i \in S_t^+ \\ w_{t,1} + \frac{\sum_{j \in S_t^-} \tau_{t,j} - \sum_{j \in S_t^+} \tau_{t,j} - c_t}{PV_t} & \text{if } i = 1 \\ w_{t,i} & \text{otherwise} \end{cases}, \tag{6}$$

where S^- and S^+ represent a set of indices of assets to sell and buy, respectively. Then, w'_t is determined by normalizing the pseudo portfolio as follows, $w'_t = \hat{w}_t / \sum_i \hat{w}_{t,i}$. Furthermore, PV'_t can be obtained by substituting the transaction cost, i.e., $PV'_t = PV_t - c_t$.

After computing w'_t and PV'_t, we reflect on the effect of the closing price change to computing the portfolio vector at time step $t+1$. When the time step changes from t to $t+1$, all assets' prices in the portfolio may be changed depending on closing price fluctuation. Hence, the portfolio value is changed. Equation (8) shows the changing formula of $\mathbf{w}'_t$ to w_{t+1}, and Equation (9) shows the changing formula of PV'_t to PV_{t+1} where r_t means a price change rate vector from t to $t+1$ and $r_{t,i} = p_{t+1,i}/p_{t,i}$,

$$r_t = (1, r_{t,2}, r_{t,3}, \ldots, r_{t,K}) \tag{7}$$

$$w_{t+1} = \frac{w'_t \odot r_t}{r_t^\intercal w'_t} \tag{8}$$

$$PV_{t+1} = PV'_t \cdot r_t^\intercal w'_t, \tag{9}$$

where $\odot$ indicates an element-wise multiplication. The flow chart of the portfolio and portfolio value changes is shown in Figure 1.

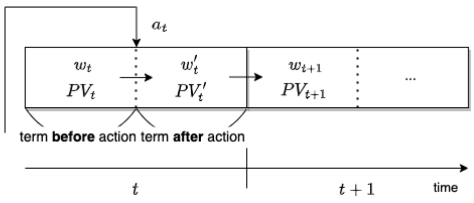


Figure 1. Illustration of the procedure of transactions during time step t. a_t indicates an action of the trading agent where $\mathbf{w}_t$ changes to $\mathbf{w}'_t$ after a_t.

3.2. Markov Decision Processes for Portfolio Management

We cast the portfolio management problem into the problem of learning an optimal strategy for sequential decision-making by using reinforcement learning algorithms. A reinforcement learning is generally formulated as a Markov decision process (MDP), which consists of a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, P indicates a transition probability of a next state given state and action pair, r is a reward function mapping from $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ to $\mathbb{R}$, and γ is a discount factor.

In the portfolio management problem, similar to [5], a state at t is defined by the stock feature matrix X_t and the portfolio vector $\mathbf{w}'_t$ at the end of t as follows,

$$s_t = (X_t, \mathbf{w}'_t). \tag{10}$$

Then, an action at t is defined as a gap between the current portfolio $\mathbf{w}_t$ and the desired portfolio $D_{t,i}$. Therefore, the ith entry of a_t is defined as Equation (11).

$$a_{t,i} = w_{t,i} - D_{t,i} \tag{11}$$

$a_{t,i}$ means the weight to be traded. Note that, for $\forall i$, $a_{t,i} \in [-1, 1]$ and $\sum_i a_{t,i} = 0$ hold. If $a_{t,i}$ is a negative number, the ith asset must be sold, and if $a_{t,i}$ is a positive number, the ith asset must be bought. However, in this way, the agent cannot have a hold position except when $a_{t,i}$ is 0, which can lead to a large transaction cost due to frequent transactions. In this regard, we introduced a hyperparameter for threshold δ and truncated the action value to zero if $|a_{t,i}| < \delta$ hold to make ith asset not be traded. Further, when the agent cannot sell or buy the asset by a proportion of $a_{t,i}$, $a_{t,i}$ is mapped to the maximum share that can be sold or bought.

For a reward function, since our agent re-invests the profits, we use a continuous compounding rate of return as a reward function, which is defined as

$$r_t = \ln(PV_t) - \ln(PV_0). \tag{12}$$

3.3. Distribution of a Portfolio

In this section, we define a distribution of a portfolio. Portfolio vector $\mathbf{w}_t$ has innate sum-to-one constraints. In other words, a set of valid portfolio vectors is equivalent to an $N-1$-dimensional simplex. Hence, to assign a distribution of a portfolio, we employ the Dirichlet distribution, which is a well-known parametric distribution over a simplex. Dirichlet distributions with concentration parameters α_1, α_2, ... $\alpha_K > 0$ have a multivariate probability density function as follows,

$$f(x_1, ..., x_K; \alpha_1, ..., \alpha_K) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} x_i^{\alpha_i - 1}. \tag{13}$$

where $B(\alpha)$ is the beta function defined as,

$$B(\alpha) := \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K} \alpha_i)}, \quad \alpha = (\alpha_i, ..., \alpha_K) \tag{14}$$

$$\Gamma(x) := \int_0^\infty t^{x-1} e^{-t} dt. \tag{15}$$

Note that random vector X sampled from Dirichlet distribution is an element in the $K-1$ simplex. Hence, $\sum_{i=1}^{K} X_i = 1$ and $X_i \in [0,1]$ hold. This property is suitable for representing a portfolio vector. By using a Dirichlet distribution, we can define a stochastic policy over action space. We would like to emphasize that this paper is the first to employ the Dirichlet distribution to represent a stochastic policy over a portfolio vector.

4. Proposed Method

We propose a novel off-policy reinforcement learning method for scalable portfolio management by using importance sampling with the Dirichlet policy. First, to obtain scalability, we employ an architecture that can easily add new assets to a test phase. Second, to increase the sample efficiency, we also apply the importance sampling method to the Dirichlet policy that can reuse the previously sampled experience.

4.1. Architecture

We propose a scalable architecture that allows a trained policy and value network to be applied to unseen financial assets, which are not experienced during a training phase but in a test phase. The overview of the proposed network architecture is given in Figure 2. Our architecture consists of two parts: a policy network as an actor with a score net and a value network as a critic with a score net and header.

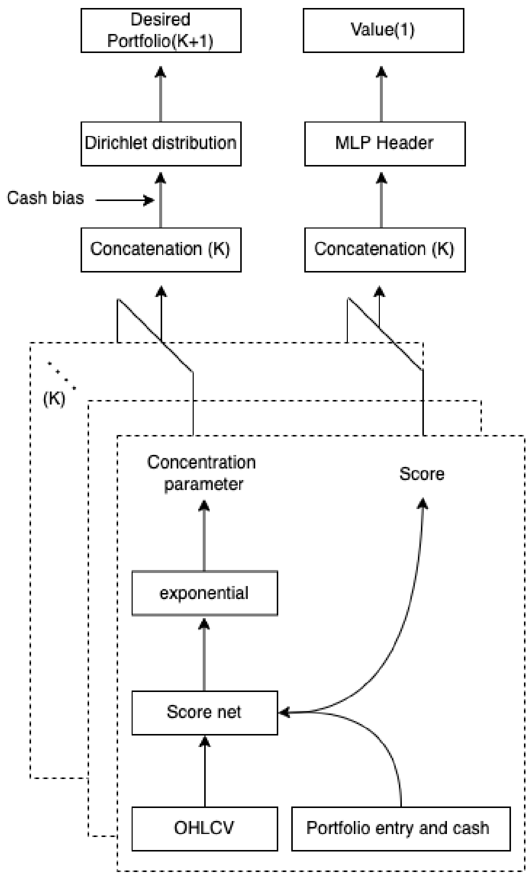


Figure 2. Architecture of score, policy, and value networks.

First, we introduce a score network g_θ whose output is the potential goodness of each asset where θ is the parameter of the network. Assume that we consider K assets in our portfolio. Then, in state S_t, we have K number of stock vectors $v_{i,t}$ and K number of portfolio weights $w'_{i,t}$. For each asset, $v_{i,t}$ and $w'_{i,t}$ are an input of the score network and $m_i := g_\theta(v_{i,t}, w'_{i,t})$ indicates the score of the ith asset. For all assets, the scores $\{m_i\}_{i=1}^K$ are predicted by the score network.

Based on $\{m_i\}_{i=1}^K$, the Dirichlet policy can be defined. Since the characteristics of the Dirichlet distribution are fully determined by the concentration parameters; we define the concentration parameters as a function of scores. Specifically, since the concentration parameters must be positive, all parameters are computed as the exponential of the scores. Finally, to consider a cash asset, we add a cash bias. Hence, the distribution policy is defined as

$$a = \mathbf{w}_t - D, \;\; D \sim \text{Dir}([1, e^{m_2}, e^{m_3}, ..., e^{m_K}]), \tag{16}$$

where 1 indicates a cash bias and D indicates the desired portfolio. By using this policy, the agent can sample various portfolio vectors from the Dirichlet distribution.

Second, the value network, which is parameterized by ϕ, is defined by combining both the score network and multi-layer perceptron (MLP) header to estimate the state value, which is defined as

$$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|S_t = s] \tag{17}$$

The state value indicates the sum of future rewards. The MLP header uses the scores of each asset as an input, and its output $V_\phi(S_t)$ predicts the state value of the given policy π. The detailed optimization method for updating ϕ to predict V_π will be proposed in the next section.

By using this architecture, we can easily add or remove assets from the portfolio in a test phase. In the test phase, a new asset can be considered by estimating its score and the Dirichlet distribution can be defined over scores, including a new one. To estimate the score of a new asset, we only need v, w' for a new asset, i.e., $m := g_\theta(v, w')$, where w' can be set to zero for the first estimation. Then, a new asset can be considered by sampling a portfolio weight from a Dirichlet distribution defined over $\{m_i\}_{i=1}^K \cup m$. Similarly, removing the existing asset can be achieved by defining a Dirichlet distribution defined over $\{m_i\}_{i=1}^K / m$.

4.2. Off-Policy Optimization

In this section, we introduce the optimization technique for value header ϕ and score network θ. To increase sample efficiency, we propose the off-policy optimization method. First, after executing actions, the transition pairs $(s_t, a_t, r_t, s_{t+1}, \pi_\theta(a_t|s_t))$ are stored in a replay buffer where $\pi(a_t|s_t)$ is stored to compute the importance weight. For the value header, ϕ is updated every time step with importance-weighted TD-target by using importance sampling,

$$\mathcal{L}_\phi = \sum_{t \in B} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} (r_t + \gamma V_{\phi'}(s_{t+1})) - V_\phi(s_t) \right)^2, \tag{18}$$

where ϕ' is a parameter for the target network, and B indicates the set of batch indices. For each update, we sample a batch of N transitions from the replay buffer. In Equation (18), we estimate the expectation of the squared TD loss with respect to π_θ; however, the transition samples are generated by the previous policy $\pi_{\theta_{old}}$, which is different from distribution π_θ. Hence, this discrepancy can be compensated by multiplying the importance weight.

To update score network θ, we employ the clipped ratio loss proposed in PPO [4].

$$\mathcal{L}_\theta = \sum_{t \in B} \left[\min(w_t(\theta)\hat{A}_t, \text{clip}(w_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right], \tag{19}$$

where $w_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$, ϵ is a clip threshold of importance ratio, and $\hat{A}_t = r_t + \gamma V_{\phi'}(s_{t+1}) - V_\phi(s_t)$. All processes of the off-policy optimization of value function V_ϕ and policy π_θ is summarized in Algorithm 1.

Algorithm 1 Drichlet Distribution Trader: DDT.

```
1:  Input: $\epsilon$, $\gamma$, $N$
2:  Initialize critic network $V_\phi$ and actor network $\pi_\theta$
3:  Initialize target critic network $\phi' \leftarrow \phi$
4:  Initialize replay buffer with size $N$
5:  for episode from 1 to $M$ do
6:    Observe the state $s_1$
7:    for $t$ from 1 to $T$ do
8:      Select action $a_t \sim \pi_\theta(s_t)$
9:      Execute action $a_t$, observe reward $r_t$ and state $s_{t+1}$
10:     Store transition tuple $(s_t, a_t, r_t, s_{t+1}, \pi(a_t|s_t))$
11:     Set $s_t \leftarrow s_{t+1}$
12:     Sample $N$ transitions $\big(s_j, a_j, r_j, s_{j+1}, \pi(a_j|s_j)\big)$ from the replay buffer
13:     Update $\phi$ by the gradient: $N^{-1}\nabla_\phi \mathcal{L}_\phi$
14:     Update $\theta$ by the policy gradient:$N^{-1}\nabla_\theta \mathcal{L}_\theta$
15:     Update target networks: $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
16:   end for
17: end for
```

4.3. Sampling by Risk

Modeling a policy of a portfolio vector by using a Dirichlet distribution has the advantage of learning multiple optimal strategies. While a deterministic policy only suggests a single optimal portfolio weight [5,15], the proposed method can model a distribution over a portfolio. Hence, our method can propose multiple sets of optimal portfolio weights. Knowing multiple optimal strategies is essential to suggesting a proper portfolio to various types of investors depending on an investment propensity, from an aggressive one to a passive one. For example, a risk-averse investor prefers a low-risk portfolio even if its expected return is low, and, on the contrary, a risk-taking one rather prefers a high-expected return even if its failure probability (or its variance) is high. In this regard, multiple portfolios are needed. Fortunately, since our policy is modeled as a distribution, we can sample multiple portfolios from the optimized distribution, and the sampled portfolios have a near-optimal expected return. By using these samples, a risk-averse or risk-taking investment can be conducted.

We propose a portfolio generation method from the resulting policy optimized by reinforcement learning. The first method uses a mean desired portfolio, which is a mean of a Dirichlet distribution, computed as follows,

$$D_i = \frac{\alpha_i}{\sum_{i=1}^{K} \alpha_i}, \tag{20}$$

where D_i indicates the desired weight of the ith asset, and α_i is a concentration parameter of a Dirichlet distribution defined in (16). The second method employs a mode of a Dirichlet distribution, where the desired portfolio can be computed as

$$D_i = \frac{\alpha_i - 1}{\sum_{i=1}^{K} \alpha_i - K}, \quad \forall a_i > 1, \tag{21}$$

The final method employs a rejection sampling based on the risk measure proposed in [1]. In this method, first, several portfolios are sampled from $\pi_\theta(\cdot|s)$. Let us denote the set of N samples as $\{D^j\}_{j=1}^N$ where D^j is the jth sampled portfolio vector, and N is set to be 1000 in our experiments. After sampling $\{D^j\}_{j=1}^N$, second, we compute the transaction cost of each sampled portfolio and select the N_{top} lowest-cost portfolios, which are denoted as $\{D^{(j)}\}_{j=1}^{N_{top}}$ where N_{top} is set to be 10. Then, in the final step, we compute the risk of each portfolio $D^{(j)}$ as follows,

$$\text{Risk} := (D^{(j)})^{\intercal}\Sigma D^{(j)} \tag{22}$$

where Σ denotes the covariance matrix of daily profit loss. By computing the risk, we can finally select three types of portfolios, which are with the highest risk, the median risk, and the lowest risk. We denote the portfolio with the lowest risk as D^{low}, with the median risk as D^{mid}, and with the highest risk as D^{high}. In the test phase, we compare these portfolio generation techniques and show the characteristics of each portfolio generation. The overview of the process of our proposed methodology, Dirichlet Distribution Trader (DDT), is given in Figure 3.

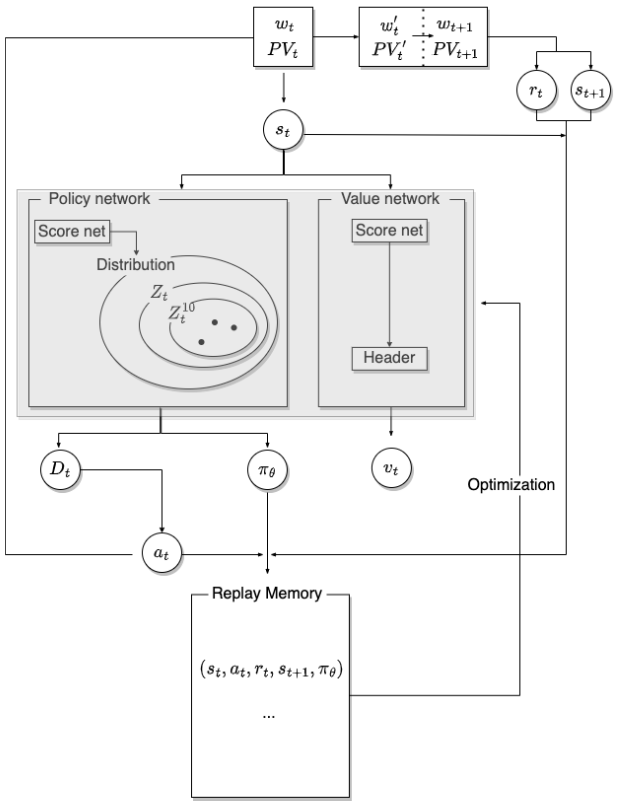


Figure 3. Process of the Dirichlet distribution trader (DDT).

5. Experiment

5.1. Performance Metrics

The performance of trading methods is measured by the degree of return or risk. Hence, we use four metrics for evaluating the performance. The first metric is the cumulative return (CR).

$$\text{CR} = \frac{PV'_T - PV_0}{PV_0} \times 100(\%), \tag{23}$$

where T is the entire trading period. The CR is the rate of change between the initial portfolio value and the portfolio value after an action in the terminal time step. The second metric is the average return (AR) defined as

$$\text{AR} = \frac{1}{T}\sum_{k=0}^{T-1} \frac{PV_{k+1} - PV_k}{PV_k}. \tag{24}$$

The AR is the mean of the daily rate of change between the portfolio value at $k+1$ and the portfolio value at k. The third metric is the Sharpe ratio (SR),

$$\mathrm{SR} = \frac{\mathbb{E}[R_D - R_f]}{\sigma}, \tag{25}$$

where R_D is a daily rate of return by desired portfolio D, R_f is a return of risk-free asset f for benchmarking, and σ is the standard deviation of R_D. The SR measures the performance or risk of an investment method compared to a risk-free asset, such as a bond. The last metric is the Maximum Drawdown (MDD) as follows,

$$\mathrm{MDD} = \frac{\mathrm{Min}(PV_{1:T}) - \mathrm{Max}(PV_{1:m})}{\mathrm{Max}(PV_{1:m})}. \tag{26}$$

where Min($PV_{1:T}$) denotes the trough portfolio value, and Max($PV_{1:m}$) denotes the peak portfolio value, where m is an index corresponding to the trough value. Therefore, the MDD measures the greatest fluctuation range from the highest portfolio value point to the lowest portfolio value point before a new peak is achieved, which we can consider the worst case in our investment.

5.2. Data Configuration

To use companies with sufficient trading volume and fluctuations, we selected seven companies for an experiment from among companies in the Nasdaq 100 index. As shown in Table 2, we constructed seven datasets by adding or removing stocks for a variety of experiments.

Table 2. Datasets. HA (Hawaiian Holdings), WBA (Walgreens Boots Alliance), INCY (Incyte), BIDU (Baidu), TCOM (Trip.com), AAPL (Apple), and COST (Costco). Companies marked in bold are not included in the base dataset, Dataset 1.

Dataset	Companies	Trend
1	HA, WBA, INCY	-
2	HA, WBA, INCY, **BIDU**	Down
3	HA, WBA, INCY, **BIDU, TCOM**	Down
4	HA, WBA, INCY, **AAPL**	Up
5	HA, WBA, INCY, **AAPL, COST**	Up
6	HA, WBA, INCY, **BIDU, TCOM, AAPL, COST**	Mix
7	**BIDU, TCOM, AAPL, COST**	Mix

Dataset 1 is the base dataset. Dataset 1 consists of stocks that include both an up-trend and a down-trend section. The down-trend dataset is a dataset in which stocks include a falling interval within the test period, such as datasets 2 and 3. Likewise, the Up-trend dataset is a dataset in which stock include a rising interval within the test period, such as dataset 4 and 5. The mix-trend dataset is a dataset that mixes the Up-trend dataset and the Down-trend dataset, which are datasets 6 and 7. All datasets are divided into two periods, one for training and one for testing in Table 3, and the closing price plots of the seven stocks during the full period are shown in Figure 4.

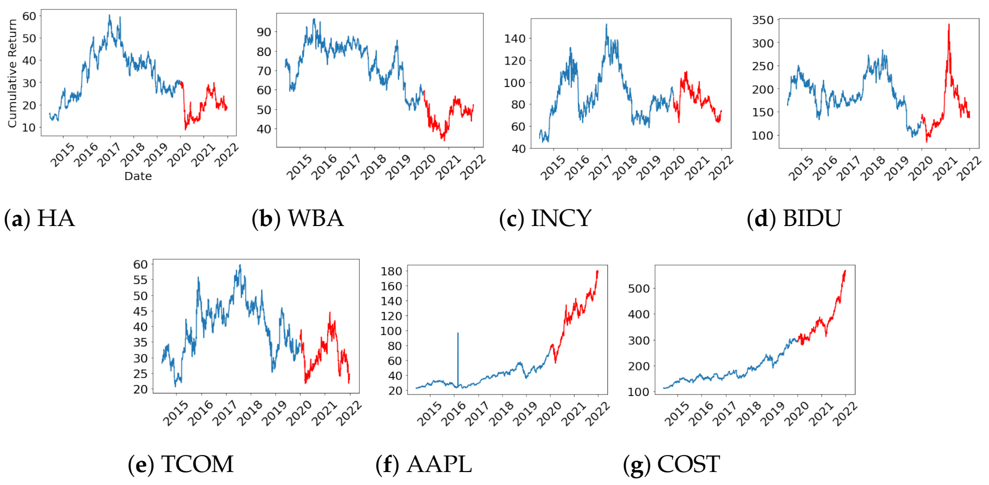


Figure 4. Closing price of the seven stocks in the experiments: Before January 2020, the closing price plot during the training period is shown, and after that, the closing price during the test period is shown.

Table 3. Train and test period.

Index	Training Period	Test Period
Nasdaq 100	June 2014–December 2019	January 2020–December 2021

5.3. Scalability Experiment

5.3.1. Experiment Setup

To test the scalability of the proposed model, we compare DDT-B, a model trained on dataset 1 only, with other nonscalable models, including DDT-A, in all datasets. If we run the experiment on dataset n $(n = 2, 3, ..., 7)$, the nonscalable models are trained during the training period of dataset n and tested in the test period of dataset n, but the scalable model DDT-B is trained only during the training period of dataset 1 and tested in the test period of dataset n. Whereas if $n = 1$, DDT-A equals DDT-B. Therefore, even if a new stock is added, DDT-B infers independently between the stocks. These are shown in Table 4.

We used DDPG [27], DQN [28] as Deep RL baselines and EW (Equal Weight) as a traditional baseline in this experiment. Therefore, we confirmed the scalability of our model by comparing the performance difference between these baselines and DDT-A, which was trained with data from all stocks in the portfolio, and DDT-B, which was not trained with data on new stocks.

Table 4. Combinations of datasets used for the scalability experiment.

Test Dataset	Model	Train	Test	Scalability
Dataset n	DDPG	Dataset n	Dataset n	x
	DQN	Dataset n	Dataset n	x
	DDT-A (ours)	Dataset n	Dataset n	x
	DDT-B (ours)	Dataset 1	Dataset n	o

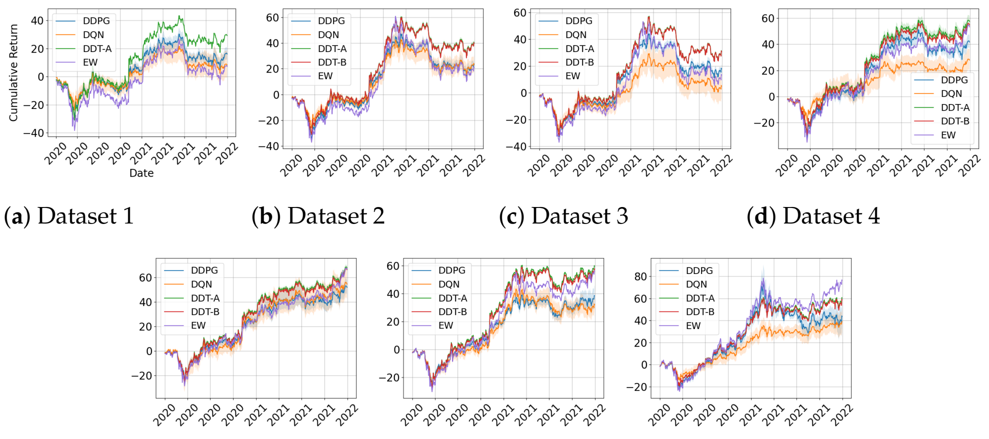
5.3.2. Experiment Results

The results of this experiment are shown in Table 5. As shown in Figure 5, except for dataset 7, our model, DDT-A, obtained the highest CR compared to other baseline algorithms. When DDT-B and DDT-A are compared, DDT-B's CR is almost the same as DDT-A's CR, and dataset 4 showed the largest difference of 3%. Therefore, DDT-B is scalable as it has almost the same cumulative return as DDT-A, which has been trained with all stocks.

In the AR results, except for datasets 5 and 7, DDT-A obtained the highest AR, and in datasets 5 and 7, EW obtained the highest AR. In the results of SR, DDT-A achieved the highest SR in all datasets. It means that our model produces good returns against the risk of stock price volatility. In the results of MDD, except for dataset 5, DQN obtained the best results. This is because the action of DQN is discrete, so it acted conservatively during the maximum fall period.

Table 5. Test results of the scalability experiment. The best results are marked as bold font.

Dataset	Algorithm	CR	SR	AR	MDD
1	DDPG	16.068	0.3997	0.0486	−27.660
	DQN	8.295	0.2556	0.0269	−**23.343**
	EW	7.206	−0.1396	0.0441	−36.425
	DDT-A	**29.164**	**0.6563**	**0.0730**	−29.389
2	DDPG	23.068	0.5636	0.0651	−33.042
	DQN	23.737	0.6265	0.0571	−**24.587**
	EW	20.955	0.4147	0.0663	−35.781
	DDT-A	**40.136**	**1.0340**	**0.0878**	−29.973
	DDT-B	39.591	0.7730	0.0872	−29.836
3	DDPG	17.863	0.4779	0.0538	−32.097
	DQN	4.996	0.1519	0.0242	−**28.368**
	EW	13.141	0.3133	0.0519	−36.341
	DDT-A	**30.819**	**0.6763**	**0.0745**	−31.589
	DDT-B	30.569	0.6683	0.0743	−31.535
4	DDPG	41.697	0.8876	0.0902	−30.741
	DQN	27.920	1.0265	0.0601	−**17.566**
	EW	53.268	0.8349	0.1087	-34.390
	DDT-A	**57.365**	**1.0563**	**0.1091**	-28.038
	DDT-B	54.796	1.0265	0.1061	−28.721
5	DDPG	51.901	1.0571	0.0983	−**24.440**
	DQN	54.583	0.9782	0.1018	−27.540
	EW	66.709	0.9723	**0.1200**	−28.548
	DDT-A	**67.749**	**1.1385**	0.1189	−25.417
	DDT-B	66.418	1.1211	0.1170	−25.197
6	DDPG	38.469	0.8697	0.0792	−26.235
	DQN	32.077	0.7910	0.0684	−**25.533**
	EW	55.809	0.8845	0.1075	−30.791
	DDT-A	**59.475**	**1.0217**	**0.1095**	−27.796
	DDT-B	57.314	1.0008	0.1069	−27.980
7	DDPG	43.459	1.0587	0.0902	-25.952
	DQN	38.824	1.0701	0.0727	−**16.071**
	EW	**76.419**	1.1116	**0.1329**	−27.299
	DDT-A	60.323	**1.1582**	0.1073	−22.073
	DDT-B	58.675	1.1574	0.1051	−21.864



(**a**) Dataset 1 (**b**) Dataset 2 (**c**) Dataset 3 (**d**) Dataset 4

(**e**) Dataset 5 (**f**) Dataset 6 (**g**) Dataset 7

Figure 5. Cumulative returns for the seven datasets of algorithms for the scalability test.

5.4. On-Policy and Off-Policy Experiment

5.4.1. Experiment Setup

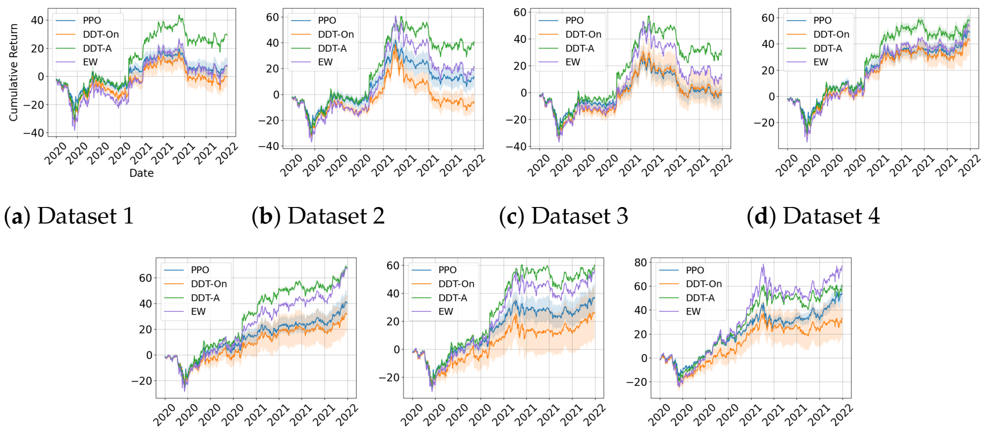
We used DDT-A as the off-policy algorithm, PPO as the on-policy algorithm, and DDT-On, which was a trained DDT with an on-policy algorithm. In this experiment, we compare the results of the on-policy algorithm that uses samples more inefficiently than the off-policy algorithm in each dataset with different trends by comparing the off-policy algorithm and the on-policy algorithm. All algorithms used for this experiment are summarized in Table 6.

Table 6. Test results of On-Off experiment. The best results are marked as bold font.

Dataset	Algorithm	CR	SR	AR	MDD
1	PPO	6.954	0.2284	0.0265	**−22.405**
	DDT-On	−0.515	−0.3363	0.0202	−30.413
	DDT-A	**29.164**	**0.6563**	**0.0730**	−29.389
2	PPO	13.304	0.4921	0.0492	**−25.494**
	DDT-On	−6.004	−0.2535	0.0114	−36.946
	DDT-A	**40.136**	**1.0340**	**0.0878**	−29.973
3	PPO	−0.393	0.0010	0.0140	**−27.132**
	DDT-On	1.632	−0.0914	0.0266	−33.825
	DDT-A	**30.819**	**0.6763**	**0.0745**	−31.589
4	PPO	48.695	**1.0582**	0.0939	**−23.922**
	DDT-On	42.606	0.8645	0.0892	−28.606
	DDT-A	**57.365**	1.0563	**0.1091**	−28.038
5	PPO	40.444	0.9481	0.0798	**−22.438**
	DDT-On	31.534	0.6592	0.0688	−22.849
	DDT-A	**67.749**	**1.1385**	**0.1189**	−25.417
6	PPO	36.987	0.8522	0.0762	**−25.423**
	DDT-On	26.294	0.3282	0.0636	−25.696
	DDT-A	**59.475**	**1.0217**	**0.1095**	−27.796
7	PPO	53.896	**1.1777**	0.0977	**−18.112**
	DDT-On	32.839	0.7657	0.0701	−25.379
	DDT-A	**60.323**	1.1582	**0.1073**	−22.073

5.4.2. Experiment Results

The results of this experiment are shown in Table 6. As shown in Figure 6, in the results of CR, DDT-A obtained the highest CR in all datasets. In particular, in datasets 1, 2, and 3, which are datasets that do not include AAPL or COST, where stock prices increase monotonically, the on-policy algorithms obtained lower CR, SR, and AR than DQN and DDPG, which are the off-policy algorithms of the previous experiment. In the results of SR, Except for datasets 4 and 7, DDT-A obtained the best SR, and in datasets 4 and 7, PPO obtained the best SR with a slight difference from DDT-A. In the results of AR, DDT-A obtained the best AR in all datasets, and in the results of MDD, PPO obtained the best MDD in all datasets. As a result, if stocks with the same trend in both the train and test data are not included in the portfolio, the performance difference between the on-policy algorithm and the off-policy is wider than in the case where it is not.



(**a**) Dataset 1 (**b**) Dataset 2 (**c**) Dataset 3 (**d**) Dataset 4

(**e**) Dataset 5 (**f**) Dataset 6 (**g**) Dataset 7

Figure 6. Cumulative returns for the seven datasets of algorithms for the On-Off test.

5.5. *Risk-Aware Portfolio Management Experiment*

5.5.1. Experiment Setup

We use three samples of DDT-A whose risk level is High, Mid, and Low, calculated according to Equation (22) and these are denoted by z_t^{low}, z_t^{mid}, and z_t^{high} in Section 4.3. These three samples are determined from among the ten low-cost samples of any ten thousand samples of the Dirichlet distribution. During the test period, an action is created by selecting only one sample of the same risk level among these three samples.

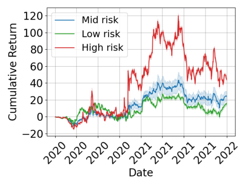
5.5.2. Experiment Results

The results of this experiment are shown in Table 7. In dataset 1, high-risk trading obtained the highest CR and AR, but low-risk trading obtained the best values for SR and MDD. In datasets 2 and 3, which are down-trend datasets, low-risk trading obtained the highest CR and AR. For SR, mid-risk trading and low-risk trading obtained the best values, respectively. However, looking at Figure 7, low-risk trading reached a high peak near 100%, but it formed a large drawdown after that, and mid-risk trading obtained the best MDD. In datasets 4 and 5, which are up-trend datasets, high-risk trading obtained the highest CR and AR. For SR, mid-risk trading obtained the best results, and for MDD, low-risk trading got the best results. In dataset 6, mid-risk trading achieved the best results in all metrics, in dataset 7, low-risk trading obtained the best results in all metrics.

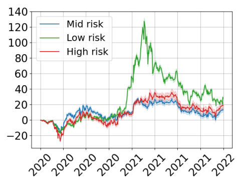
Specifically, in MDD, except for dataset 6, the low-risk or mid-risk result is better than the MDD of PPO in the previous experiment, and the best MDD of dataset 6, −25.899 of low risk, is also not much different from the −25.424 of PPO.

This experiment shows the characteristics of trading according to risk level. In general, high-risk investments expect high returns. These expectations are met in the up-trend dataset but not in the down-trend dataset. Further, in the downtrend dataset, low-risk has a SR similar to or much higher than that of mid-risk. On the other hand, mid-risk has better MDD than low-risk. Therefore, if SR is the most important criterion, low-risk investment can be considered, and if MDD is the most important criterion, mid-risk investment can be considered.

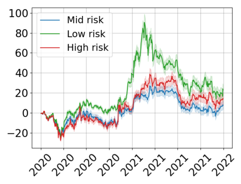
That is, we have the advantage of being able to select samples that reflect these characteristics according to the level of risk.



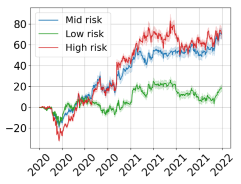
(**a**) Dataset 1



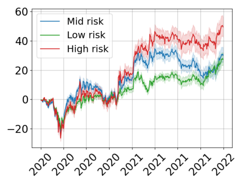
(**b**) Dataset 2



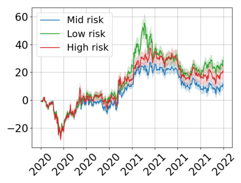
(**c**) Dataset 3



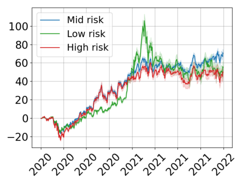
(**d**) Dataset 4



(**e**) Dataset 5



(**f**) Dataset 6



(**g**) Dataset 7

Figure 7. Cumulative returns for the seven datasets of algorithms for the sampling test.

Table 7. Test results of the sampling-by-risk experiment. The best results are marked as bold font.

Dataset	Risk Level	CR	SR	AR	MDD
1	High risk	**44.431**	1.0549	**0.1294**	−38.349
	Mid risk	24.296	0.9722	0.0536	−22.952
	Low risk	15.307	**1.0584**	0.0399	**−21.772**
2	High risk	18.572	0.7216	0.0511	−27.134
	Mid risk	12.9616	**1.0555**	0.0361	**−21.806**
	Low risk	**28.033**	0.8753	**0.0754**	−48.019
3	High risk	13.955	0.4027	0.0474	−28.803
	Mid risk	7.572	0.2529	0.0303	**−25.104**
	Low risk	**22.825**	**0.9276**	**0.0618**	−40.221
4	High risk	**70.525**	1.1355	**0.1318**	−33.110
	Mid risk	70.145	**1.2844**	0.1171	−19.616
	Low risk	18.194	0.7712	0.0474	**−18.317**
5	High risk	**49.416**	0.9672	**0.0968**	−27.293
	Mid risk	30.142	**1.0279**	0.0664	−24.416
	Low risk	27.385	0.6949	0.0575	**−20.636**
6	High risk	20.524	0.7498	0.0547	−29.589
	Mid risk	11.041	0.5010	0.0355	−27.886
	Low risk	**28.095**	**0.8146**	**0.0651**	**−25.899**
7	High risk	49.786	1.2745	0.0970	−25.348
	Mid risk	**70.340**	**1.3422**	**0.1178**	**−18.030**
	Low risk	55.105	1.0813	0.1060	−32.327

6. Conclusions

We proposed a Dirichlet Distribution Trader (DDT) that is a scalable DRL model for selectively managing portfolios according to risk. Its policy has a Dirichlet sistribution in order for an agent to generate multiple portfolio samples. Therefore, our algorithm can selectively manage the portfolio according to the level of risk after selecting 10 portfolios with low transaction costs. In the Risk-Aware Portfolio Management Experiment, we showed that the cumulative returns of portfolios corresponding to each of the three risk levels had distinct characteristics according to the trend of the dataset and showed the need for selective portfolio management. In addition, since the value $\pi(a|s)$ can be obtained from the distribution, efficient training is possible through off-policy learning by importance sampling, and we showed in the on-policy and off-policy experiment that it has better performance than on-policy learning. Our model is not limited to the number of portfolio stocks and has the scalability to adjust the weight of new stocks added to the portfolio even if only three stocks in the base dataset are learned. Furthermore, in the scalability experiment, DDT-A, which is trained on only three stocks, showed almost the same performance as DDT-B, which is trained for all stocks in the portfolio. Based on these advantages, comparative experiments show that DDT is superior to other algorithms in risk metrics and return metrics.

Author Contributions: Conceptualization, H.Y. and K.L.; methodology, H.Y.; software, H.Y.; validation, H.Y., H.P. and K.L.; formal analysis, H.Y. and K.L.; investigation, H.Y.; resources, H.Y.; data curation, H.Y.; writing—original draft preparation, K.L.; writing—review and editing, H.P.; visualization, H.Y.; supervision, K.L.; project administration, K.L.; funding acquisition, K.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the IITP grant funded by the Korea government (MSIT) (No. 2021-0-01341, AI Graduate School Program, CAU).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

IITP	Institute for Information and Communications Technology Promotion
MSIT	Ministry of Science and ICT

References

1. Markowitz, H. Portfolio Selection. *J. Financ.* **1952**, *7*, 77–91. [CrossRef]
2. Park, H.; Sim, M.K.; Choi, D.G. An intelligent financial portfolio trading strategy using deep Q-learning. *Expert Syst. Appl.* **2020**, *158*, 113573. [CrossRef]
3. Betancourt, C.; Chen, W.H. Deep reinforcement learning for portfolio management of markets with a dynamic number of assets. *Expert Syst. Appl.* **2021**, *164*, 114002. [CrossRef]
4. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347v2. [CrossRef]
5. Jiang, Z.; Xu, D.; Liang, J. A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem. *arXiv* **2017**, arXiv:1706.10059
6. Grinblatt, M.; Titman, S.; Wermers, R. Momentum Investment Strategies, Portfolio Performance, and Herding: A Study of Mutual Fund Behavior. *Am. Econ. Rev.* **1995**, *85*, 1088–1105.
7. Malkiel, B.G. Efficient Market Hypothesis. In *Finance*; Eatwell, J., Milgate, M., Newman, P., Eds.; Palgrave Macmillan UK: London, UK, 1989; pp. 127–134. [CrossRef]
8. Jegadeesh, N.; Titman, S. Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency. *J. Financ.* **1993**, *48*, 65–91. [CrossRef]

9. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
10. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
11. Takeuchi, L.; Lee, Y.Y.A. *Applying Deep Learning to Enhance Momentum Trading Strategies in Stocks*; Technical Report; Stanford University: Stanford, CA, USA, 2013.
12. Krauss, C.; Do, X.A.; Huck, N. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. *Eur. J. Oper. Res.* **2017**, *259*, 689–702. [CrossRef]
13. Fischer, T.; Krauss, C. Deep learning with long short-term memory networks for financial market predictions. *Eur. J. Oper. Res.* **2018**, *270*, 654–669. [CrossRef]
14. Zhang, Z.; Zohren, S.; Stephen, R. Deep Reinforcement Learning for Trading. *J. Financ. Data Sci.* **2020**, *2*, 25–40. [CrossRef]
15. Liang, Z.; Chen, H.; Zhu, J.; Jiang, K.; Li, Y. Adversarial Deep Reinforcement Learning in Portfolio Management. *arXiv* **2018**, arXiv:1808.09940.
16. Lee, J.; Kim, R.; Koh, Y.; Kang, J. Global Stock Market Prediction Based on Stock Chart Images Using Deep Q-Network. *IEEE Access* **2019**, *7*, 167260–167277. [CrossRef]
17. Taghian, M.; Asadi, A.; Safabakhsh, R. Learning financial asset-specific trading rules via deep reinforcement learning. *Expert Syst. Appl.* **2022**, *195*, 116523. [CrossRef]
18. Lee, K.; Choi, S.; Oh, S. Sparse Markov Decision Processes With Causal Sparse Tsallis Entropy Regularization for Reinforcement Learning. *IEEE Robotics Autom. Lett.* **2018**, *3*, 1466–1473. [CrossRef]
19. Lee, K.; Kim, S.; Lim, S.; Choi, S.; Hong, M.; Kim, J.I.; Park, Y.; Oh, S. Generalized Tsallis Entropy Reinforcement Learning and Its Application to Soft Mobile Robots. In Proceedings of the Robotics: Science and Systems XVI, Virtual Event/Corvalis, OR, USA, 12–16 July 2020.
20. Cetin, E.; Çeliktutan, O. Learning Routines for Effective Off-Policy Reinforcement Learning. In Proceedings of the 38th International Conference on Machine Learning, ICML 2021, Virtual, 18–24 July 2021, Volume 139, pp. 1384–1394.
21. Wang, Y.; He, P.; Tan, X. Greedy Multi-step Off-Policy Reinforcement Learning. *arXiv* **2021**, arXiv:2102.11717 .
22. Kong, S.; Nahrendra, I.M.A.; Paek, D. Enhanced Off-Policy Reinforcement Learning With Focused Experience Replay. *IEEE Access* **2021**, *9*, 93152–93164. [CrossRef]
23. Vijayan, N.; Prashanth, L.A. Smoothed functional-based gradient algorithms for off-policy reinforcement learning: A non-asymptotic viewpoint. *Syst. Control. Lett.* **2021**, *155*, 104988. [CrossRef]
24. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]
25. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.P.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. *arXiv* **2016**, arXiv:1602.01783.
26. Ingersoll, J.; Ingersoll, J. *Theory of Financial Decision Making*; G-Reference, Information and Interdisciplinary Subjects Series; Rowman & Littlefield: Totowa, NJ, USA, 1987.
27. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**. arXiv:1509.02971.
28. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
29. Rummery, G.A.; Niranjan, M. *On-Line Q-Learning Using Connectionist Systems*; Technical Report; Cambridge University Engineering Department: Cambridge, UK, 1994. .