Optimal Action Space Search: an Effective Deep Reinforcement Learning Method for Algorithmic Trading

Zhongjie Duan
East China Normal University
Shanghai, China
zjduan@stu.ecnu.edu.cn

Cen Chen*
East China Normal University
Shanghai, China
cenchen@dase.ecnu.edu.cn

Dawei Cheng*
Tongji University
Shanghai, China
dcheng@tongji.edu.cn

Yuqi Liang
Seek Data Group, Emoney Inc.
Shanghai, China
roly.liang@seek-data.com

Weining Qian
East China Normal University
Shanghai, China
wnqian@dase.ecnu.edu.cn

ABSTRACT

Algorithmic trading is a crucial yet challenging task in the financial domain, where trading decisions are made sequentially from milliseconds to days based on the historical price movements and trading frequency. To model such a sequential decision making process in the dynamic financial markets, Deep Reinforcement Learning (DRL) based methods have been applied and demonstrated their success in finding trading strategies that achieve profitable returns. However, the financial markets are complex imperfect information games with high-level of noise and uncertainties which usually make the exploration policy of DRL less effective. In this paper, we propose an end-to-end DRL method that explores solutions on the whole graph via a probabilistic dynamic programming algorithm. Specifically, we separate the state into environment state and position state, and model the position state transition as a directed acyclic graph. To obtain reliable gradients for model training, we adopt a probabilistic dynamic programming algorithm to explore solutions over the whole graph instead of sampling a path. By avoiding the sampling procedure, we propose an efficient training algorithm and overcome the efficiency problem in most existing DRL methods. Furthermore, our method is compatible with most recurrent neural network architecture, which makes our method easy to implement and very effective in practice. Extensive experiments have been conducted on two real-world stock datasets. Experimental results demonstrate that our method can generate stable trading strategies for both high-frequency and low-frequency trading, significantly outperforming the baseline DRL methods on annualized return and Sharpe ratio.

CCS CONCEPTS

• **Information systems → Data mining**.

*Corresponding authors.

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557412

KEYWORDS

Reinforcement learning, Algorithmic trading

ACM Reference Format:
Zhongjie Duan, Cen Chen, Dawei Cheng, Yuqi Liang, and Weining Qian. 2022. Optimal Action Space Search: an Effective Deep Reinforcement Learning Method for Algorithmic Trading. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557412

1 INTRODUCTION

Financial trading is a challenging task that aims to maximize profits through repetitive trading orders. Traditional investors used to trade financial products based on their domain knowledge and expertise, but they can hardly fully track the frequent changes in the volatile market and make timely decisions. To address this challenge, various automated algorithmic trading techniques were developed and have been widely applied to many problems related to trading and investment that covers diverse financial products such as stocks, convertible bonds, cryptocurrencies, etc.

Most of the early works in algorithmic trading are rules-based algorithms that exploit financial theories, such as the classical trend-following, delta-neutral, and mean-reversion strategies [33, 37, 43]. Although the assumptions in these works are often oversimplified, the theories behind them can provide important technical indicators [31] that inspire the subsequent Deep Learning (DL) based trading methods [45]. The majority of them regard the algorithmic trading problem as a forecasting or classification task. For example, Arévalo et. al. [1] proposed a DNN-based method that predicts the future price and makes decisions accordingly. Dixon et. al. [10] utilizes recurrent neural networks for price classification. Although DL based methods have been demonstrated to be effective for algorithmic trading [32, 50], several limitations are faced. First, DL based methods usually adopt mean square error or cross-entropy loss as the training target, which is different from the buy-sell signals provided in the trading process. Second, the training and backtesting targets are also inconsistent, where profit is evaluated by a simulated trading program in the backtesting phase. These inconsistencies often lead to suboptimal model performance. In general, DL based methods cannot directly mimic the buy-sell behavior and optimize the trading profit, which prevents them from pursuing high returns.

Thus, effective methods that can better capture real-world trading behaviors need to be explored.

To overcome the above-mentioned limitations, existing works [12] leverage Reinforcement Learning (RL)[42] to model the sequential decision process in real-world trading. In this process, the trading agent interacts with the environment (i.e., the stock market or a simulated trading program), with the goal to maximize the expected return. More recently, a few works investigate Deep Reinforcement Learning (DRL) which uses deep neural networks as function approximators, as a promising direction for algorithmic trading [5, 49]. DRL based methods have shown huge applicability, potentially rivaling professional human investors [34].

However, directly applying DRL methods in algorithmic trading inevitably faces several challenges. First, the financial data gathered from volatile markets are highly noisy, thus it may make the DRL strategies unstable. Meanwhile, most financial markets are incomplete information games [7]. Traders do not possess full information about their opponents, which makes financial markets full of uncertainty. To mitigate data noise and uncertainty, technical indicators are used as input data to represent the market conditions, but their effectiveness relies on expertise. Second, the reward design is handcraft, which highly depends on the domain expertise. Some existing studies focus on designing a reward mechanism based on financial theory [48, 51]. For example, Wu et. al. [48] uses the Sortino ratio as a reward to better handle risk. These reward mechanisms also rely on expertise and are poor at generalization.

To address these challenges, we propose a novel DRL method named *Optimal Action Space Search (OASS)* that explores the optimal decisions on the whole graph via a probabilistic dynamic programming algorithm. Specifically, we separate the state into *environment state* and *position state*. The position states form a directed acyclic graph and every path in the graph represents a sequence of decisions. To avoid the adverse effects of data noise and uncertainty on the model, we use a probabilistic dynamic programming algorithm that explores solutions on the whole graph and calculates the expected trading profit. This algorithm provides reliable and stable policy gradients for parameter updating. Leveraging the probabilistic dynamic programming algorithm, OASS does not require a well-designed reward mechanism. A simple reward function that represents the change of capital is enough. Additionally, we implement OASS mainly on GPU and greatly improve the efficiency of the algorithm. To demonstrate the effectiveness of OASS, we compared it with 7 state-of-the-art DRL baselines on real-world high-frequency trading and low-frequency trading datasets. The results strongly demonstrate the superior performance of our proposed method. The main contributions of our paper include:

- We propose a novel end-to-end DRL method (OASS) for algorithmic trading that can search optimal decisions in the whole action space. This method is compatible with most recurrent neural network architecture, which is easy to implement and very effective in practice.
- We improve the model stability and training efficiency in algorithmic trading problem. By leveraging a probabilistic dynamic programming algorithm, we obtain reliable gradients for parameter updating, which significantly accelerate the convergence speed.
- We demonstrate that our proposed method significantly outperforms other baseline DRL methods in the experiments. We have released the source code of our proposed method[1] and a benchmark library for algorithmic trading[2].

2 RELATED WORK

In this section, we briefly review the related studies.

2.1 Deep Reinforcement Learning

Reinforcement learning [42] is proposed for tackling complex sequential decision problems. Its theory is rooted in the neuroscientific field of behavior control [39]. Using neural networks as function approximators, DRL has reached state-of-the-art performance in some fields [20, 40]. In most existing DRL methods, the problem is modeled as a Markov Decision Process [35]. The model attempts to get high rewards by exploration, which is implemented by the interactions of an agent and an environment. DRL methods form three main techniques [34]: critic-only, actor-only, and actor-critic learning models. Critic-only models use only the action-value function to make decisions. DQN [28] is a typical critic-only method using ϵ-greedy policy for exploration. In 2015, a study [28] proved that DQN is able to reach human-level control in Atari games. Actor-only models output a probabilistic distribution for decision-making. REINFORCE [47] was proposed as an actor-only policy gradient method in 1992. Different from the above two kinds of methods, actor-critic methods combine two model architectures, where the actor model makes decisions and the critic model evaluates them. A2C [22] proposes an advantage function to accelerate the training process. DDPG [25] is the first DRL method designed for continuous control. However, these deep reinforcement learning methods still face challenges in stability. To address the stability problem, PPO [38] was proposed in 2017. It directly controls the size of the gradient ascent step by optimizing a "surrogate" objective function. Compared to these existing methods, OASS expands the explored area from a path to the whole graph and gives reliable gradients for model training.

2.2 Algorithmic Trading

Algorithmic trading is defined as buy-sell decisions made solely by algorithmic models [34]. With the rapid development of online training platforms and systems, trading algorithm based on deep learning and deep reinforcement learning becomes a popular topic in the field of finance. In deep learning, some studies regard the trading process as a price prediction [14] or classification [10] problem, which results in that the buy-sell decisions are not made by models directly. To implement an end-to-end training algorithm, Deng et. al. [8] proposed a method using differential profit function as loss function. It is the first study to build a trading system using DRL. DRL is getting increasing attention over the years and some open-source DRL projects [27] are released to stimulate research. Many studies construct algorithmic trading framework based on PG [18, 19], DQN [5, 17], A2C [51], DDPG [2, 49] and PPO [24]. Some improved DRL methods are designed specifically for algorithmic trading. In 2018, Action Augmentation is proposed by Huang et.

[1] https://github.com/ECNU-CILAB/OASS
[2] https://github.com/ECNU-CILAB/TriangleStrategy

al. [16] as an exploration policy to replace the ϵ-greedy policy in DQN. Action Augmentation calculates the rewards while choosing other actions, which makes the model learn more information than before. A study in 2019 [6] has demonstrated that DRL methods have the potential to outperform supervised deep learning methods. Most studies focus on applying existing general DRL methods in algorithmic trading. We propose a novel DRL algorithm and further improve its performance in algorithmic trading.

3 PRELIMINARY

We model the algorithmic trading problem as a Markov Decision Process. In most existing reinforcement learning methods, the simulated trading procedure is implemented by the interactions of an agent (trader) and an environment (market). At each time step i, the agent observes the state $s_i \in \mathcal{S}$ (market condition) from the environment and chooses an action $a_i \in \mathcal{A}$ (e.g., buy, sell or do nothing). Then the environment receives the action a_i and calculates the reward $r_i \in \mathbb{R}$ (return). The training objective of reinforcement learning is the value function:

$$G(s_i) = r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} + \cdots, \tag{1}$$

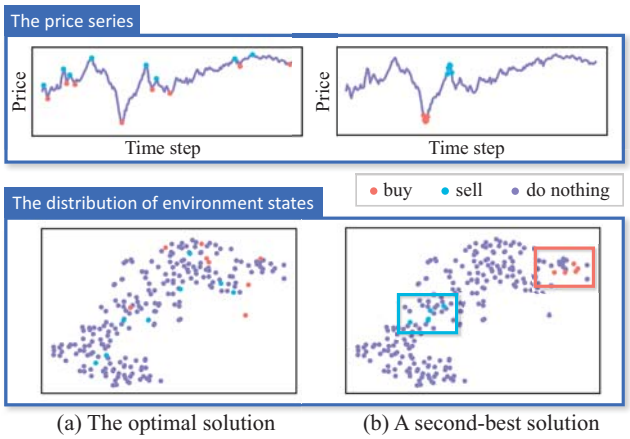
where γ is the discount factor and usually $\gamma \in [0, 1]$.

State Space. $\mathcal{S} = \mathcal{S}^e \times \mathcal{S}^a$. The state $s_i \in \mathcal{S}$ is a vector input to the model. As mentioned in many existing studies [12, 32], it is difficult to accurately measure the influence of traders on market conditions. In this problem, assuming the trading actions taken by the agent have no influence on the market, we separate the state into $s_i = [s_i^e, s_i^a]$. $s_i^e \in S^e$ is the *environment state*, which contains price, volume, and other technical indicators to represent the market condition. Environment states are provided by the environment completely and would not be changed by whatever actions the agent chose. $s_i^a \in S^a$ is the *position state*, which represents the asset position. Considering trading on a single finance product, $\mathcal{S}^a = \{-1, 0, 1\}$ is a discrete space. The three position states correspond to three different asset position states.

- $s_i^a = 1$ represents holding one unit of this finance product. The agent can get returns from the increasement of price.
- $s_i^a = 0$ represents no holding. The agent can mitigate risks in the financial market.
- $s_i^a = -1$ represents that the agent has sold one unit of this finance product and may buy it back in the future. The agent can get returns from the decreasement of price.

Action Space. $\mathcal{A}$. The action space is a discrete space $\{-1, 0, 1\}$. The three actions correspond to maximizing short position, no holding and maximizing long position. This action space design has been proved efficient in existing works [5, 18, 51]. The position state is the representation of the influence caused by actions. In this problem, actions change position states directly, i.e., if $a_i = 1$, then $s_{i+1}^a = 1$ at the next time step.

Reward Function. $r_i = R(s_i^a, s_{i+1}^a)$. In the simulated trading system, at each time step i, buy price p_i^{buy} and sell price p_i^{sell} are provided by the environment. The environment will decrease the money when the agent chooses to buy (i.e., $s_{i+1}^a > s_i^a$) and increase the money when the agent chooses to sell (i.e., $s_{i+1}^a < s_i^a$). The transaction cost is also calculated at the same time, which is calculated by the product of transaction amounts and cost ratio. To guide the model to gain a high expected return, we design a simple reward function representing the change of capital. Note that the reward does not include the value of financial products held by the agent, thus can avoid the adverse impact of short-term sharp price volatility on rewards. The reward function is formulated as follows:



(a) The optimal solution (b) A second-best solution

Figure 1: An example of a financial product in one day. The left one is the optimal solution and the right one is a second-best solution. We reduce the dimensionality of environment states to a 2D plane. The optimal solution can achieve a high profit but it is weakly correlated to environment states. A second-best solution that is strongly correlated to environment states is desirable.

$$R(s_i^a, s_{i+1}^a) = \begin{cases} -|s_{i+1}^a - s_i^a|(1+\alpha^{\text{buy}})p_i^{\text{buy}}, & \text{if } s_{i+1}^a > s_i^a, \\ |s_{i+1}^a - s_i^a|(1-\alpha^{\text{sell}})p_i^{\text{sell}}, & \text{otherwise,} \end{cases} \tag{2}$$

where α^{buy} and α^{sell} are the cost ratio. For example, in China stock market, the cost ratio is thousandth. With the reward function defined above. equation 1 is reformulated as:

$$G(s_i^e, s_i^a) = R(s_i^a, s_{i+1}^a) + \gamma R(s_{i+1}^a, s_{i+2}^a) + \gamma^2 R(s_{i+2}^a, s_{i+3}^a) + \cdots \tag{3}$$

When $\gamma = 1$, the cumulative reward value equals the transaction return, which is the optimization objective of the algorithmic trading problem. Moreover, to avoid that the agent only decides to sell, the actions at the beginning and the end of each sequence are forced to be 0 (i.e., $s_1 = s_{n+1} = 0$). In the following part, we regard γ as 1 by default.

In addition, to describe the relation between actions and position states, we use $P(s_i^a, s_{i+1}^a | s_i^e, a_i)$ to denote the probability that position state s_i^a will be changed to s_{i+1}^a when the agent chooses action a_i. As we described above, the action a_i changes the position state directly, thus $P(s_i^a, \cdot | s_i^e, \cdot)$ is an identity matrix:

$$P(s_i^a, s_{i+1}^a | s_i^e, a_i) = \begin{cases} 1, & \text{if } a_i = s_{i+1}^a, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Based on the above problem definition, we can calculate the optimal solution to this problem using a dynamic programming algorithm. For the ease of explanations, we use $a^*(s_i^e, s_i^a)$ to denote the theoretical optimal action at time step i, and use $G^*(s_i^e, s_i^a)$ to denote the maximized value of $G(s_i^e, s_i^a)$. Obviously, the state

transformation equation is

$$G^*(s_i^e, s_i^a) = \max_{a_i \in \mathcal{A}} U(s_i^e, s_i^a, a_i), \tag{5}$$

where

$$U(s_i^e, s_i^a, a_i) = \sum_{s_{i+1}^e \in \mathcal{S}^e} P(s_i^a, s_{i+1}^a | s_i^e, a_i) \Big(R(s_i^a, s_{i+1}^a) + \gamma G^*(s_{i+1}^e, s_{i+1}^a) \Big). \tag{6}$$

It's easy to see that the optimal action $a^*(s_i^e, s_i^a)$ is

$$a^*(s_i^e, s_i^a) = \arg\max_{a_i \in \mathcal{A}} U(s_i^e, s_i^a, a_i). \tag{7}$$

In most complete information game [13] (e.g., Go), the primary problem is to find the optimal solution, which is strongly correlated to the state. However, the algorithmic trading problem is an incomplete information game. The optimal solution is weakly correlated to the state. Letting the model directly fit the optimal solution described above using supervised learning methods is conceptually elegant but practically unreasonable. Figure 1 shows an example of a financial product in one day. Ideally, if we chose the optimal decision, the cumulative return would be very high. The high profit is obviously not commonsensible and it is impossible to make these accurate high-frequency decisions. Leveraging TSNE algorithm [44], we reduce the dimensionality of environment states to a 2D plane. We can easily see that the distribution of environment states corresponding to buy and sell action is chaotic in Figure 1(a) but regular in Figure 1(b). Actually, a second-best solution that is strongly correlated to environment states is what we want, but it is completely different from the optimal solution. That is why we cannot use supervised learning methods to train the model.

4 METHODOLOGY

4.1 Model Architecture

Algorithmic trading can be viewed as a sequential decision process that can be solved by reinforcement learning. The goal of reinforcement learning is to find an optimal mapping π between states and a probability distribution over the action space $\mathcal{A}$. The mapping is usually implemented by a neural network model. We use $\pi(a_i|\theta, s_i^e, s_i^a)$ to denote the probability of choosing action a_i at state (s_i^e, s_i^a), where θ is the learnable parameters of π.

In recent years, many model architectures are used in algorithmic trading, including MLP, CNN, LSTM, and others [48, 50]. LSTM [15] is well-suited for modelling time series data. LSTM has become the most popular model architecture in algorithmic trading and has shown its advantages [6, 18, 36], as it is able to sense the dynamic market condition in the historical data. Therefore, we decide to employ a k-layer LSTM as our model backbone, the outputs of LSTM cells are calculated as follows:

$$\left(h_1^j, h_2^j, \ldots, h_n^j\right) = \begin{cases} \left(s_1^e, s_2^e, \ldots, s_n^e\right), & j = 0, \\ \text{LSTM}\left(h_1^{j-1}, h_2^{j-1}, \ldots, h_n^{j-1} | \theta^j\right), & 1 \le j \le k, \end{cases} \tag{8}$$

where θ^j is the trainable parameters in the j-th LSTM cell. In the training phase, dropout [41] is utilized for regularization. After that, we map the output to $\mathbb{R}^{|\mathcal{S}^a| \times |\mathcal{A}|}$ using a fully connected layer, then

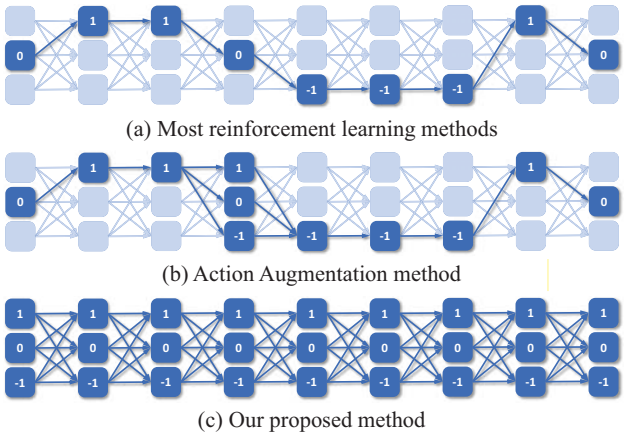


Figure 2: The explored area of three kinds of methods. (a) Most DRL methods only obtain a path. (b) Action Augmentation method expands the path by replacing the action with other actions. (c) Our proposed method is able to explore the whole directed acyclic graph.

reshape it to a matrix with $|\mathcal{S}^a|$ rows and $|\mathcal{A}|$ columns.

$$o_i = \text{reshape}\left(W_o h_i^k + b_o\right), 1 \le i \le n. \tag{9}$$

Finally, we calculate the probability at each time step i. i.e., applying Softmax function on each row of o_i.

$$\pi(a_i|\theta, s_i^e, s_i^a) = \frac{\exp\left(o_i(s_i^a, a_i)\right)}{\sum_{a \in \mathcal{A}} \exp\left(o_i(s_i^a, a)\right)}, \tag{10}$$

where $\theta = \left(\theta^1, \theta^2, \ldots, \theta^k, W_o, b_o\right)$ refers to the entire trainable parameters of this neural network model. The Softmax function guarantees that $\sum_{a_i \in \mathcal{A}} \pi(a_i|\theta, s_i^e, s_i^a) = 1$ for every state $(s_i^e, s_i^a) \in \mathcal{S}^e \times \mathcal{S}^a$.

4.2 Training Algorithm

In model-free DRL methods, sequences of (s_i, a_i, r_i) are sampled from the interactions between the agent and environment. The sampled data is stored in replay memory [26] for updating the neural network models. This framework ensures the generality of the methods but results in high resource requirements. As shown in Figure 2, the position states at each time step construct a directed acyclic graph, and a sampled sequence corresponds with a path. While updating the neural network model, it can only gain information from the path but not the whole graph. In order to expand the explored area, AlphaGo [40] utilizes Monte-Carlo Tree Search (MCTS) [4] and it beats many competitors in Go. However, MCTS is also limited on resource requirements. Action Augmentation method [16] was proposed in 2018. Instead of ϵ-greedy exploration strategy, Action Augmentation method calculates reward values the agent will gain if it chooses other actions. Action Augmentation is designed for algorithmic trading specifically because the action space $\mathcal{A}$ only contains three actions and the reward value is easy to calculate if an action is changed. We are interested in designing a better high-performance exploration method than these methods.

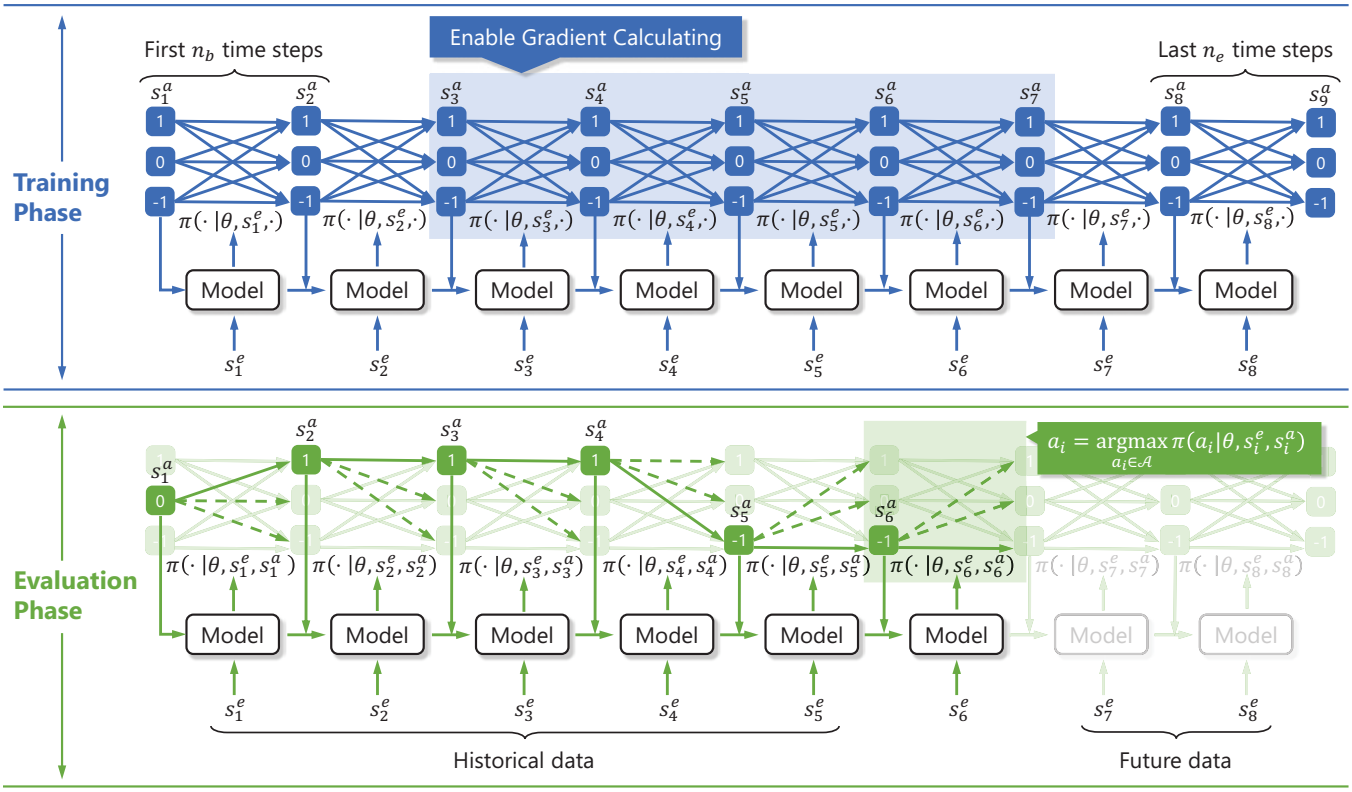


Figure 3: The training and evaluation phase of OASS. In the training phase, the probabilities at the first n_b and the last n_e time steps are ignored. Only the probabilities at the middle $n - n_b - n_e$ time steps are used in gradient calculation. In the evaluation phase, the model is updated incrementally and chooses the action with the biggest probability.

The recurrent form of equation 3 is

$$G(s_i^e, s_i^a) = R(s_i^a, s_{i+1}^a) + \gamma G(s_{i+1}^e, s_{i+1}^a). \tag{11}$$

At time step i, assuming the agent chooses action a_i with probability $\pi(a_i|\theta, s_i^e, s_i^a)$, where $\pi(a_i|\theta, s_i^e, s_i^a)$ is the output value of the neural network model. Now we can calculate the expectation of $G(s_i^e, s_i^a)$ using a probabilistic dynamic programming algorithm.

$$\begin{aligned} E\big(G(s_i^e, s_i^a)|\pi, \theta\big) &= E\big(R(s_i^a, s_{i+1}^a) + \gamma G(s_{i+1}^e, s_{i+1}^a)|\pi, \theta\big) \\ &= \sum_{a_i \in \mathcal{A}} \pi(a_i|\theta, s_i^e, s_i^a) D(s_i^a, a_i), \end{aligned} \tag{12}$$

where

$$D(s_i^a, a_i) = \sum_{s_{i+1}^a \in \mathcal{S}^a} P(s_i^a, s_{i+1}^a|s_i^e, a_i) Q(s_i^a, s_{i+1}^a), \tag{13}$$

$$Q(s_i^a, s_{i+1}^a) = R(s_i^a, s_{i+1}^a) + \gamma E\big(G(s_{i+1}^e, s_{i+1}^a)|\pi, \theta\big). \tag{14}$$

It is easy to prove that $E\big(G(s_i^e, s_i^a)|\pi, \theta\big) = G^*(s_i^e, s_i^a)$ when

$$\pi(a_i|\theta, s_i^e, s_i^a) = \begin{cases} 1, & \text{if } a_i = a^*(s_i^e, s_i^a), \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

As we described in Section 3, it is almost impossible to find a perfect model that satisfies the above formula because of the weak correlation. Considering only time step i, we simplify the optimization problem as

$$\arg\max_{\theta} E\big(G(s_i^e, s_i^a)|\pi, \theta\big). \tag{16}$$

Our main idea is to select θ randomly and then adjust θ to increase $E\big(G(s_i^e, s_i^a)|\pi, \theta\big)$ in each iteration.

Equation 12 is a variant of Bellman equation [9]. Based on this, we can calculate $E\big(G(s_i^e, s_i^a)|\pi, \theta\big)$ at each time step recurrently. The time complexity is $O(n|\mathcal{A}||\mathcal{S}^a|^2)$, where n is the length of the sequence. Fortunately, $|\mathcal{A}|$ and $|\mathcal{S}^a|$ are both very small. Compared to the neural network model, the time complexity of this probabilistic dynamic programming algorithm is negligible. We want that $E\big(G(s_i^e, s_i^a)|\pi, \theta\big)$ is as bigger as possible because it equals to the expected trading return after time step i. However, $E\big(G(s_i^e, s_i^a)|\pi, \theta\big)$ cannot be utilized as a loss function directly because it relies on $E\big(G(s_{i+1}^e, s_{i+1}^a)|\pi, \theta\big)$, which makes the gradient back propagation procedure complex and slow. If we ignore the relation between $E\big(G(s_{i+1}^e, s_{i+1}^a)|\pi, \theta\big)$ and (π, θ) (i.e., regard $E\big(G(s_{i+1}^e, s_{i+1}^a)|\pi, \theta\big)$ as a constant), the gradient will be simple. We name this adjustment as *Detached Step*.

$$\nabla_\theta E\big(G(s_i^e, s_i^a)|\pi, \theta\big) = \sum_{a_i \in \mathcal{A}} D(s_i^a, a_i) \nabla_\theta \pi(a_i|\theta, s_i^e, s_i^a). \tag{17}$$

Then construct a policy gradient equation for gradient-based optimization algorithms.

$$\nabla_\theta \mathcal{L}_i(\theta, s_i^e, s_i^a) = \frac{1}{|\mathcal{A}|} \sum_{a_i \in \mathcal{A}} D(s_i^a, a_i) \nabla_\theta \pi(a_i|\theta, s_i^e, s_i^a). \quad (18)$$

Some existing study [46] has proved that a baseline is essential for gradient-based DRL. The policy gradient equation with a baseline b is formulated as

$$\nabla_\theta \mathcal{L}_i(\theta, s_i^e, s_i^a) = \frac{1}{|\mathcal{A}|} \sum_{a_i \in \mathcal{A}} \Big(D(s_i^a, a_i) - b\Big) \nabla_\theta \pi(a_i|\theta, s_i^e, s_i^a). \quad (19)$$

The coefficient $D(s_i^a, a_i) - b$ is often called advantage function. $D(s_i^a, a_i) - b > 0$ means that increasing $\pi(a_i|\theta, s_i^e, s_i^a)$ leads to a higher expected return than before, and vice versa. In actor-critic method, the baseline is an estimation of $G(s_i^e, s_i^a)$ given by another model. In OASS, there is no need to utilize another function approximator because $E\big(G(s_i^e, s_i^a)|\pi, \theta\big)$ has been calculated before. We can use $E\big(G(s_i^e, s_i^a)|\pi, \theta\big)$ as the baseline value directly.

$$b = E\big(G(s_i^e, s_i^a)|\pi, \theta\big). \quad (20)$$

Note that $D(s_i^a, a_i) - E\big(G(s_i^e, s_i^a)|\pi, \theta\big) \approx 0$ when $\pi(a_i|\theta, s_i^e, s_i^a) \approx 1$. This phenomenon results in vanishing gradient. Thus, we use another robust way to calculate b:

$$b = \frac{1}{|\mathcal{A}|} \sum_{a_i \in \mathcal{A}} D(s_i^a, a_i). \quad (21)$$

Considering the architecture of LSTM, the probabilities at the beginning of each sequence are not stable, because there is little historical information provided to the model. The coefficients $D(s_i^a, a_i)$ at the end of each sequence are also unreliable because there are not enough reward values in the future. To overcome this, we delete the gradients of the first n_b and last n_e time steps. That is to say, the gradient of a sequence $\{s_i^e\}_{i=1}^n$ is

$$\nabla_\theta \mathcal{L} = \frac{1}{(n - n_b - n_e)|\mathcal{S}^a|} \sum_{i=n_b+1}^{n-n_e} \sum_{s_i^a \in \mathcal{S}^a} \nabla_\theta \mathcal{L}_i(\theta, s_i^e, s_i^a). \quad (22)$$

The pseudo-code of OASS algorithm is shown in algorithm 1. First, we calculate the probability $\pi(a_i|\theta, s_i^e, s_i^a)$ for every $1 \le i \le n$, $a_i \in \mathcal{A}$ and $s_i^a \in \mathcal{S}^a$. The model takes $\{s_i^e\}_{i=1}^n$ as input and takes $\{\pi(\cdot|\theta, s_i^e, \cdot) \in \mathbb{R}^{|\mathcal{S}^a| \times |\mathcal{A}|}\}_{i=1}^n$ as output. Second, we calculate the gradients at each time step. Finally, update the parameters θ using gradient-based optimizer algorithms.

Algorithm 1 Optimal Action Space Search algorithm

Input $\{s_i^e\}_{i=1}^n$: a sequence of environment states.
for $i = 1, 2, \ldots, n$ **do**
  Calculate $\pi(a_i|\theta, s_i^e, s_i^a), s_i^a \in \mathcal{S}^a, a_i \in \mathcal{A}$
end for
for $i = n, n-1, \ldots, 1$ **do**
  for $(s_i^a, s_{i+1}^a) \in \mathcal{S}^a \times \mathcal{S}^a$ **do**
    $R(s_i^a, s_{i+1}^a) = \begin{cases} -|s_{i+1}^a - s_i^a|(1+\alpha^{\text{buy}})p_i^{\text{buy}}, & \text{if } s_{i+1}^a > s_i^a, \\ |s_{i+1}^a - s_i^a|(1-\alpha^{\text{sell}})p_i^{\text{sell}}, & \text{otherwise.} \end{cases}$
    $Q(s_i^a, s_{i+1}^a) = R(s_i^a, s_{i+1}^a) + \gamma E\big(G(s_{i+1}^e, s_{i+1}^a)|\pi, \theta\big)$
  end for
  for $s_i^a \in \mathcal{S}^a$ **do**
    for $a_i \in \mathcal{A}$ **do**
      $D(s_i^a, a_i) = \sum_{s_{i+1}^a \in \mathcal{S}^a} P(s_i^a, s_{i+1}^a | s_i^e, a_i) Q(s_i^a, s_{i+1}^a)$
    end for
    $E\big(G(s_i^e, s_i^a)|\pi, \theta\big) = \sum_{a_i \in \mathcal{A}} \pi(a_i|\theta, s_i^e, s_i^a) D(s_i^a, a_i)$
    $b = \frac{1}{|\mathcal{A}|} \sum_{a_i \in \mathcal{A}} D(s_i^a, a_i)$
    $\nabla_\theta \mathcal{L}_i(\theta, s_i^e, s_i^a) = \frac{1}{|\mathcal{A}|} \sum_{a_i \in \mathcal{A}} \Big(D(s_i^a, a_i) - b\Big) \nabla_\theta \pi(a_i|\theta, s_i^e, s_i^a)$
  end for
end for
$\nabla_\theta \mathcal{L} = \frac{1}{(n-n_b-n_e)|\mathcal{S}^a|} \sum_{i=n_b+1}^{n-n_e} \sum_{s_i^a \in \mathcal{S}^a} \nabla_\theta \mathcal{L}_i(\theta, s_i^e, s_i^a)$
Return $\nabla_\theta \mathcal{L}$

4.3 Implementation Optimization

Note that the gradient value $D(s_i^a, a_i)$ relies on the actions happened after time step i. It will mislead the model if the actions happened after time step i are unreliable. Our OASS framework cannot overcome this pitfall for all sequential decision making problems with a general solution. But in algorithmic trading, we can leverage the idea of curriculum learning [29]. At the beginning of the training phase, we set the transaction cost coefficient α^{buy} and α^{sell} to 0, thus the model is encouraged to transact frequently. α^{buy} and α^{sell} are increased linearly to guide the model to construct a robust trading strategy.

Another weakness of DRL methods is training efficiency. Due to the interactions in DRL, training a DRL model completely in GPU is intractable and most DRL methods need plenty of time for sampling. But in OASS, we use a probabilistic dynamic programming algorithm instead of sampling. The probabilistic dynamic programming algorithm runs in parallel for a batch of sequences. With GPU implantation, the proposed OASS method can get extra speedup and is shown to be more efficient than other DRL methods.

As illustrated in Figure 3, there is no sampling in the training phase, and only the probabilities at the middle $n - n_b - n_e$ time steps are used in gradient calculation. The dropout modules in the model architecture are enabled while training. In the evaluation phase, the dropout modules are disabled. To further reduce uncertainty and make the trading strategy stable, the agent chooses the action with the biggest probability in the evaluation phase.

$$a_i = \arg\max_{a_i \in \mathcal{A}} \pi(a_i|\theta, s_i^e, s_i^a). \quad (23)$$

5 EXPERIMENT

In this section, we present the OASS's performance and compare it with existing state-of-the-art baseline methods.

5.1 Experiment Settings

We collected two datasets from China Stock Market. The two datasets contain 4543 stocks in China. (i) The first dataset is high-frequency limited order book (LOB) data, including the minute-level price and volume information. The ask price and bid price are used as the deal price. The train set is from 2021-01-01 to 2021-04-30, the validation set is from 2021-05-01 to 2021-05-31, and the test set is from 2021-06-01 to 2021-07-31. (ii) The second dataset is low-frequency data, including day-level market information. The close price every day is used as the deal price. Considering that the data

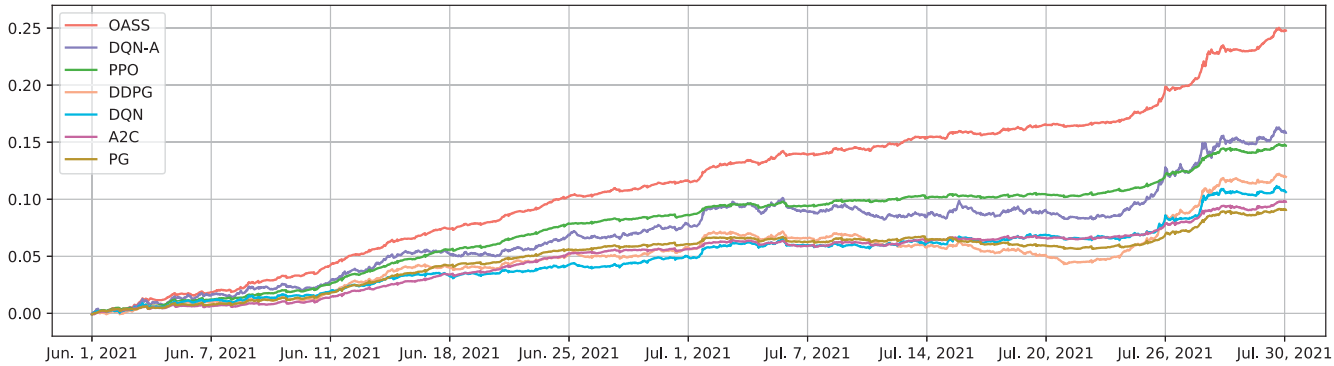


Figure 4: Cumulative return of each method in high-frequency trading.

Table 1: Performance comparison between OASS and other baseline methods

Task	Method	AR↑	V↓	MDD↓	SR↑
High-frequency trading	PG	0.5345	0.9376	0.0105	0.5701
	A2C	0.5663	**0.8454**	0.0057	0.6699
	DQN	0.6412	1.1602	0.0082	0.5527
	DQN-A	0.9315	1.9660	0.0173	0.4738
	DDPG	0.7088	1.8691	0.0269	0.3792
	PPO	0.8683	1.0578	**0.0040**	0.8209
	OASS	**1.4124**	1.4635	0.0046	**0.9650**
Low-frequency trading	PG	0.0403	2.0766	0.0967	0.0194
	A2C	0.0765	2.2366	0.0841	0.0342
	DQN	0.1570	1.3000	0.0263	0.1207
	DQN-A	0.1897	1.7793	0.0481	0.1066
	DDPG	0.0707	2.0885	0.0854	0.0338
	PPO	0.2159	1.5175	0.0308	0.1423
	OASS	**0.2519**	**1.2534**	**0.0186**	**0.2009**

amount of low-frequency trading is much smaller than that of high-frequency trading, we collect low-frequency trading data over a larger time span. The train set is from 2018-01-01 to 2020-06-30, the validation set is from 2020-07-01 to 2020-12-31, and the test set is from 2021-01-01 to 2021-09-30.

With the formulated problem described in Section. 3, many general DRL methods can be employed in algorithmic trading problem. We compare our method with the following baseline methods. These methods have been applied in algorithmic trading problems.

- **Policy Gradients (PG)** [18]: The vanilla policy gradient method is implemented by REINFORCE algorithm. It uses sampled value function to construct gradients.
- **Advanced Actor-Critic (A2C)** [3]: This method contains of an actor model and a critic model. The first one outputs the probability of each action, and the second one outputs an estimation of the value function.
- **Deep Q Network (DQN)** [5]: DQN is a very popular DRL method in many fields. It contains a Q network and uses ϵ-greedy policy to explore a better solution.
- **Deep Q Network with Action Augmentation (DQN-A)** [16]: Based on DQN, Action Augmentation is proposed as an improved exploring policy. It calculates the change of reward while an action is replaced by another action.
- **Deep Deterministic Policy Gradiet (DDPG)** [49]: DDPG is a DRL method for continuous action space.
- **Proximal Policy Optimization (PPO)** [11]: PPO directly controls the size of the gradient ascent step by optimizing a "surrogate" objective function.

To performance a fair comparison, the model architectures of each method are the same except for the output layer. To prevent models from overfitting, the model architecture used is a 3-layer LSTM. The hidden size of each LSTM cell is 32 in high-frequency trading and 8 in low-frequency trading. General DRL methods cannot utilize the separated state, thus position states are encoded as one-hot vectors. Considering that DDPG is designed especially for continuous action space, the model is allowed to output any action values in the range $[-1, 1]$. We use Adam optimizer [21] and set the learning rate to 10^{-4}. We set n_b and n_e to 20. Besides, the input data, the reward design, and the backtest program are the same for all experiments. The curriculum learning module is applied in all methods. The evaluation workflow consists of three steps. (i) Train models on the training set until convergence. (ii) Test models on the validation set and select the top 20% stocks with the maximum return. (iii) Run simulated trading on the test set, where the capital is distributed evenly across each selected stock. Specifically, we use the following metrics for evaluation:

- **Annualized Return (AR)**: The annualized return represents the average return of a trading strategy in a year.
- **Volatility (V)**: Volatility is a measure of risks. It is calculated as the annualized standard deviation of daily returns.
- **Max Drawdown (MDD)**: Max drawdown is the maximum observed loss from a peak to a trough of a portfolio before a new peak is attained.
- **Sharpe Ratio (SR)**: Sharpe ratio is the average return earned in excess of the risk-free rate per unit of volatility. It is calculated as AR divided by V.

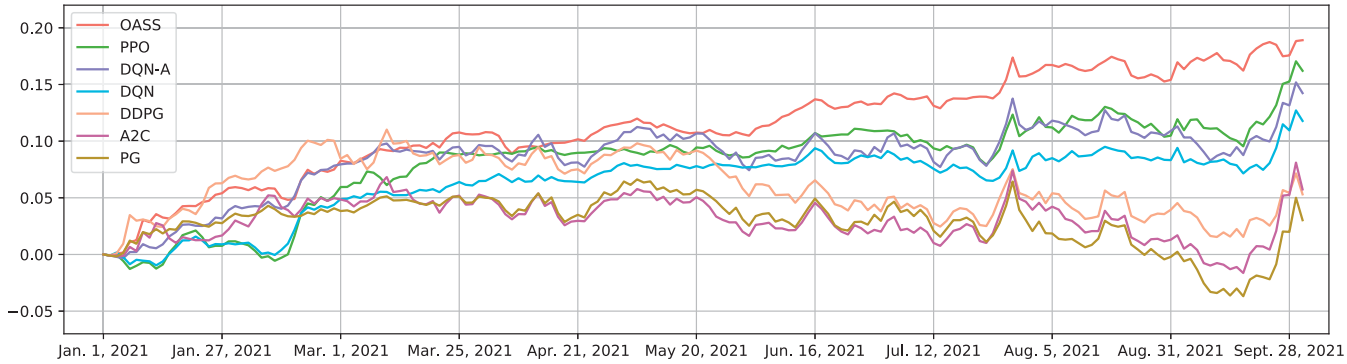


Figure 5: Cumulative return of each method in low-frequency trading.

Table 2: Performance of OASS-P in high-frequency trading

	AR↑	V↓	MDD↓	SR↑
Average	1.4076	1.4673	0.0045	0.9593
Maximum	1.4211	1.4790	0.0046	0.9628
Minimum	1.4002	1.4589	0.0045	0.9490

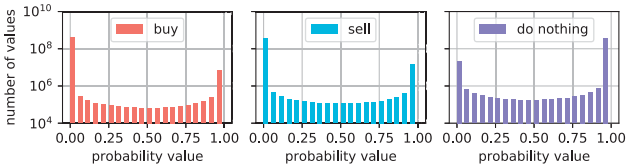


Figure 6: The distribution of three classes of probabilities.

5.2 Experiment Results

Figure 4 shows the cumulative return curve of all methods in high-frequency trading. We have the following findings. Benefiting from the rich information in LOB data, the cumulative returns of all methods are positive. It demonstrates the effectiveness of existing DRL methods. Our proposed method outperforms all baseline methods significantly in return. The cumulative return of OASS from Jun. 1, 2021, to Jul.30, 2021, is up to 25%, while DQN-A, the second-best method, can only get a return lower than it by 9%. DQN-A works better than DQN. DQN-A can improve the model's performance by expanding the explored area, but it still underperforms OASS. A2C, DDPG, and PPO are three methods that contain more than one neural network model. The difference between A2C and PG is minimal even if the first one utilizes another critic model. DDPG faces a great risk. Only PPO can get a stable return because of its prudent updating algorithm. Comparing the performance of PG and A2C, we can infer that a complex model architecture contributes very little to improving model performance because critic models cannot give an accurate estimation of future rewards based on the noisy data.

The metrics are reported in Table 1. Volatility and Max drawdown reflect the risk of investment strategies. High annualized returns and low risks are hard to be obtained simultaneously. We can see that the MDD of PPO is the lowest, but its annualized return is lower than DDPG. Sharpe ratio is a comprehensive evaluation index considering both return and risk, and OASS gets the highest one. OASS can get a stable return within the acceptable risk.

The similar experiment results in low-frequency trading are shown in Figure 5 and Table 1. The correlation between input data and price is weaker than in high-frequency trading. In begin of 2021, the cumulative return of DDPG is the highest, however, its high return is unsustainable. In all, OASS gets the highest cumulative return. Even considering all the four metrics, the overall performance of OASS is still the best.

Another important aspect to examine is the efficiency of training and inference. As OASS is GPU-friendly, in our implementation, it requires no more than a quarter of the training time of the competing methods. While in the inference stage, since the model backbone used by different methods is the same, the inference speed is not much different. Inference takes no more than 10ms, regardless of whether it is high-frequency trading or low-frequency trading.

5.3 Model Stability

At the training phase, we assume that the action a_i is chosen with probability $\pi(a_i|\theta, s_i^e, s_i^a)$, while at the evaluation phase, a_i is chosen with formula 23. From the view of game theory, we want a strategy close to the Nash Equilibrium [30]. In many incomplete information games [7], Nash Equilibrium is usually a mixed strategy, where the player chooses a probability distribution over some possible pure strategies. Therefore, choosing actions with probabilities is conceptually elegant. For convenience, we name this variant of OASS as OASS-P. Using the same well-trained model as in high-frequency trading, we backtest OASS-P 10 times and report the result in Table 2.

The results are surprising, the performance of OASS-P is only a little worse than OASS, which is insignificant. To find out the reasons, we analyze the distribution of three classes of probabilities $\pi(a_i|\theta, s_i^e, s_i^a)$: buy ($a_i > s_i^a$), sell ($a_i < s_i^a$) and do nothing ($a_i = s_i^a$). In Figure 6, obviously most probability values are close to 0 or close to 1. We can claim that a well-trained model in OASS is "confident". To reduce uncertainty in decision making, we suggest choosing action based on formula 23 in practice.

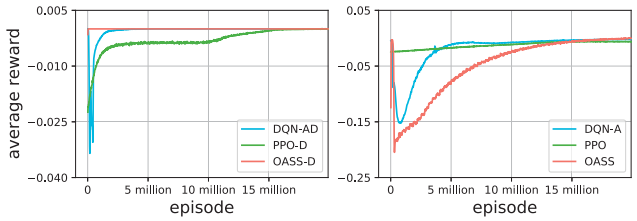


Figure 7: The average rewards of three methods in the training phase. OASS-D, PPO-D and DQN-AD denotes the corresponding variants without the curriculum learning module.

5.4 Ablation Study

In this subsection, we further study how curriculum learning influences the models' performance. We compare the best three methods: OASS, PPO, and DQN-A, and report the average reward for every 2^{14} episode. The results are shown in Figure 7, where OASS-D, PPO-D, and DQN-AD are the corresponding methods without the curriculum learning module. It is clear to see that the three methods without the curriculum learning module reach a zero reward invariably because they find that no transactions lead to no risks. This shows a general flaw in DRL. When the curriculum learning module is applied, the average rewards of the three methods decrease initially and then increase slowly. At the beginning of the training phase, models are trained to trade with no cost, which is completely different from the real task. Then the transaction cost is increased linearly, which encourages models to reduce the trading frequency and find stable trading strategies. The average reward of OASS is lower than the other two methods, but finally, it becomes the highest one. Finally, the average rewards of other methods may still be lower than zero. The main reason is that not all financial products are suitable for algorithmic trading. That is also why we select the top 20% stocks with the maximum return on the validation set in the above experiments. The ablation study results show that curriculum learning can avoid models falling into a naive local optimum. This conclusion applies to the above three DRL methods.

5.5 Case Study

As we described in Section 3, the theoretical optimal decisions could be obtained by a dynamic programming algorithm. To analyze the relation between theoretical optimal decisions and model's decisions, leveraging the TSNE algorithm [44], we visualize the environment states s_i^e and the outputs of each layer $\{h_i^j\}_{j=1}^3$ in OASS. The results are shown in Figure 8. The first four figures show the model's decisions. In Figure 8(a), the distribution of buying points and selling points looks irregular. After three nonlinear transformations implemented by LSTM, the buy points and sell points form two clusters. The LSTM layers construct a mapping $\mathcal{S} \to R^n$ and form a behavior pattern in $\mathbb{R}^n$. The last four figures show the theoretical optimal decisions. Comparing Figure 8(a) and 8(e), we find that the model decides to reduce transaction frequency for avoiding risks, which is consistent with financial common sense. In Figure 8(h), the distribution of buying points and selling points are still irregular even if in the last LSTM layer. This means the model's behavior pattern is completely different from the theoretical optimal

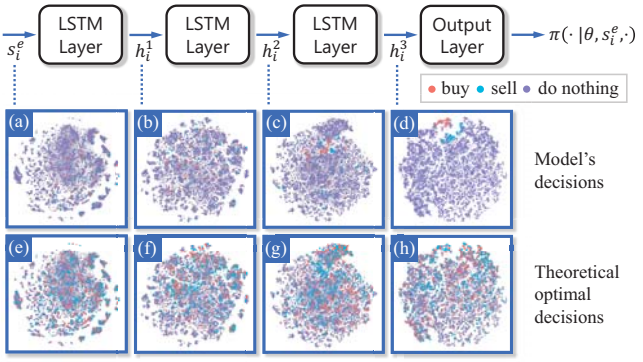
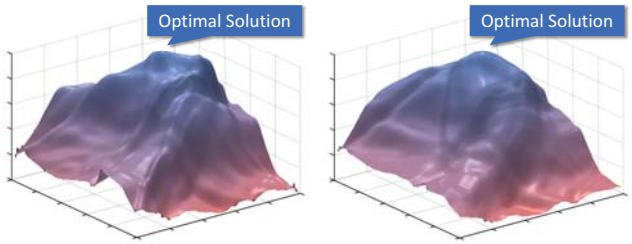


Figure 8: The distribution of environment states and the outputs of each layer in OASS. We reduce the dimensionality of data using TSNE algorithm.



(a) Without Detached Step (b) With Detached Step

Figure 9: The objective function surfaces.

decisions. The correlation between theoretical optimal decisions and states is so weak that our model cannot fit, but OASS can find a more practical behavior pattern. This is the main difference between OASS and other supervised learning methods.

We also study how Detached Step helps the model reach this solution. We select a batch of data randomly and plot the objective function surfaces leveraging the visualization method proposed by Li et. al. [23]. In Figure 9(a), the objective function surface is rugged. This means solving the optimization problem (16) directly is difficult. But when we enable Detached Step, the objective function surface in Figure 9(b) becomes smooth. There are fewer local optimum points in Figure 9(b) than in Figure 9(a). Detached Step makes the optimization problem easier than before and makes gradient-based optimization algorithms more effective.

6 CONCLUSION AND FUTURE WORK

In this paper, we investigate the feasibility of DRL methods in the algorithmic trading problem. Specifically, we propose a new DRL algorithm (OASS) and demonstrate its effectiveness. OASS can expand the explored area to the whole search space using a probabilistic dynamic programming algorithm. Benefit from the exploring algorithm, the gradient values obtained by OASS are more reliable and stable than other DRL methods. Therefore, OASS can gain high reward values rapidly. In the backtest experiments, the strategy based on OASS outperforms all other methods in both high-frequency and low-frequency trading. In future work, we will focus on improving the universality of OASS and explore its application in other fields.

REFERENCES

[1] Andrés Arévalo, Jaime Niño, German Hernández, and Javier Sandoval. 2016. High-frequency trading strategy based on deep neural networks. In *International conference on intelligent computing*. Springer, 424–436.

[2] Akhil Raj Azhikodan, Anvitha GK Bhat, and Mamatha V Jadhav. 2019. Stock trading bot using deep reinforcement learning. In *Innovations in Computer Science and Engineering*. Springer, 41–49.

[3] Stelios D Bekiros. 2010. Heterogeneous trading strategies with adaptive fuzzy actor–critic reinforcement learning: A behavioral approach. *Journal of Economic Dynamics and Control* 34, 6 (2010), 1153–1170.

[4] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 4. 216–217.

[5] Lin Chen and Qiang Gao. 2019. Application of deep reinforcement learning on automated stock trading. In *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 29–33.

[6] Quang-Vinh Dang. 2019. Reinforcement learning in stock trading. In *International conference on computer science, applied mathematics and applications*. Springer, 311–322.

[7] Claude d'Aspremont and Louis-André Gérard-Varet. 1979. Incentives and incomplete information. *Journal of Public economics* 11, 1 (1979), 25–45.

[8] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. 2016. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems* 28, 3 (2016), 653–664.

[9] Avinash K Dixit, John JF Sherrerd, et al. 1990. *Optimization in economic theory*. Oxford University Press on Demand.

[10] Matthew F Dixon, Nicholas G Polson, and Vadim O Sokolov. 2019. Deep learning for spatio-temporal modeling: dynamic traffic flows and high frequency trading. *Applied Stochastic Models in Business and Industry* 35, 3 (2019), 788–807.

[11] Yuchen Fang, Kan Ren, Weiqing Liu, Dong Zhou, Weinan Zhang, Jiang Bian, Yong Yu, and Tie-Yan Liu. 2021. Universal trading for order execution with oracle policy distillation. *arXiv preprint arXiv:2103.10860* (2021).

[12] Thomas G Fischer. 2018. *Reinforcement learning in financial markets-a survey*. Technical Report. FAU Discussion Papers in Economics.

[13] Drew Fudenberg and Jean Tirole. 1991. *Game theory*. MIT press.

[14] Ma Hiransha, E Ab Gopalakrishnan, Vijay Krishna Menon, and KP Soman. 2018. NSE stock market prediction using deep-learning models. *Procedia computer science* 132 (2018), 1351–1362.

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[16] Chien Yi Huang. 2018. Financial trading as a game: A deep reinforcement learning approach. *arXiv preprint arXiv:1807.02787* (2018).

[17] Gyeeun Jeong and Ha Young Kim. 2019. Improving financial trading decisions using deep Q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications* 117 (2019), 125–138.

[18] WU Jia, WANG Chen, Lidong Xiong, and SUN Hongyong. 2019. Quantitative trading on stock market based on deep reinforcement learning. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[19] Zhengyao Jiang and Jinjun Liang. 2017. Cryptocurrency portfolio management with deep reinforcement learning. In *2017 Intelligent Systems Conference (IntelliSys)*. IEEE, 905–913.

[20] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589.

[21] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[22] Vijay Konda and John Tsitsiklis. 1999. Actor-critic algorithms. *Advances in neural information processing systems* 12 (1999).

[23] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems* 31 (2018).

[24] Zhipeng Liang, Hao Chen, Junhao Zhu, Kangkang Jiang, and Yanran Li. 2018. Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940* (2018).

[25] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning.. In *ICLR (Poster)*.

[26] Ruishan Liu and James Zou. 2018. The effects of memory replay in reinforcement learning. In *2018 56th annual allerton conference on communication, control, and computing (Allerton)*. IEEE, 478–485.

[27] Xiao-Yang Liu, Jingyang Rui, Jiechao Gao, Liuqing Yang, Hongyang Yang, Zhaoran Wang, Christina Dan Wang, and Jian Guo. 2021. FinRL-Meta: A Universe of Near-Real Market Environments for Data-Driven Deep Reinforcement Learning in Quantitative Finance. *arXiv preprint arXiv:2112.06753* (2021).

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[29] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. 2020. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *Journal of Machine Learning Research* 21 (2020), 1–50.

[30] John F Nash Jr. 1950. Equilibrium points in n-person games. *Proceedings of the national academy of sciences* 36, 1 (1950), 48–49.

[31] Christopher J Neely, David E Rapach, Jun Tu, and Guofu Zhou. 2014. Forecasting the equity risk premium: the role of technical indicators. *Management science* 60, 7 (2014), 1772–1791.

[32] Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer. 2020. Deep learning for financial applications: A survey. *Applied Soft Computing* 93 (2020), 106384.

[33] James M Poterba and Lawrence H Summers. 1988. Mean reversion in stock prices: Evidence and implications. *Journal of financial economics* 22, 1 (1988), 27–59.

[34] Tidor-Vlad Pricope. 2021. Deep reinforcement learning in quantitative algorithmic trading: A review. *arXiv preprint arXiv:2106.00123* (2021).

[35] Lawrence R Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (1989), 257–286.

[36] Francesco Rundo. 2019. Deep LSTM with reinforcement learning layer for financial trend prediction in FX high frequency trading systems. *Applied Sciences* 9, 20 (2019), 4460.

[37] Christian Schmitt and Jürgen Kaehler. 1996. *Delta-neutral volatility trading with intra-day prices: an application to options on the dax*. Technical Report. ZEW Discussion Papers.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[39] Wolfram Schultz, Peter Dayan, and P Read Montague. 1997. A neural substrate of prediction and reward. *Science* 275, 5306 (1997), 1593–1599.

[40] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

[42] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[43] Andrew C Szakmary, Qian Shen, and Subhash C Sharma. 2010. Trend-following trading strategies in commodity futures: A re-examination. *Journal of Banking & Finance* 34, 2 (2010), 409–426.

[44] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[45] Guifeng Wang, Longbing Cao, Hongke Zhao, Qi Liu, and Enhong Chen. 2021. Coupling macro-sector-micro financial indicators for learning stock representations with less uncertainty. In *The 35th AAAI Conference on Artificial Intelligence*. 4418–4426.

[46] Lex Weaver and Nigel Tao. 2013. The optimal reward baseline for gradient-based reinforcement learning. *arXiv preprint arXiv:1301.2315* (2013).

[47] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.

[48] Xing Wu, Haolei Chen, Jianjia Wang, Luigi Troiano, Vincenzo Loia, and Hamido Fujita. 2020. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences* 538 (2020), 142–158.

[49] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. 2018. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522* (2018).

[50] Zihao Zhang, Stefan Zohren, and Stephen Roberts. 2019. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing* 67, 11 (2019), 3001–3012.

[51] Zihao Zhang, Stefan Zohren, and Stephen Roberts. 2020. Deep reinforcement learning for trading. *The Journal of Financial Data Science* 2, 2 (2020), 25–40.