applied sciences

MDPI

Article

Hybrid Deep Reinforcement Learning for Pairs Trading

Sang-Ho Kim, Deog-Yeong Park and Ki-Hoon Lee *

School of Computer and Information Engineering, Kwangwoon University, 20 Kwangwoon-ro, Nowon-gu, Seoul 01897, Korea; ksh3247610@naver.com (S.-H.K.); spdla6124@naver.com (D.-Y.P.)
* Correspondence: kihoonlee@kw.ac.kr

Abstract: Pairs trading is an investment strategy that exploits the short-term price difference (spread) between two co-moving stocks. Recently, pairs trading methods based on deep reinforcement learning have yielded promising results. These methods can be classified into two approaches: (1) indirectly determining trading actions based on trading and stop-loss boundaries and (2) directly determining trading actions based on the spread. In the former approach, the trading boundary is completely dependent on the stop-loss boundary, which is certainly not optimal. In the latter approach, there is a risk of significant loss because of the absence of a stop-loss boundary. To overcome the disadvantages of the two approaches, we propose a hybrid deep reinforcement learning method for pairs trading called HDRL-Trader, which employs two independent reinforcement learning networks; one for determining trading actions and the other for determining stop-loss boundaries. Furthermore, HDRL-Trader incorporates novel techniques, such as dimensionality reduction, clustering, regression, behavior cloning, prioritized experience replay, and dynamic delay, into its architecture. The performance of HDRL-Trader is compared with the state-of-the-art reinforcement learning methods for pairs trading (P-DDQN, PTDQN, and P-Trader). The experimental results for twenty stock pairs in the Standard & Poor's 500 index show that HDRL-Trader achieves an average return rate of 82.4%, which is 25.7%P higher than that of the second-best method, and yields significantly positive return rates for all stock pairs.

Keywords: algorithmic trading; pairs trading; deep learning; reinforcement learning

Citation: Kim, S.-H.; Park, D.-Y.; Lee, K.-H. Hybrid Deep Reinforcement Learning for Pairs Trading. *Appl. Sci.* **2022**, *12*, 944. https://doi.org/10.3390/app12030944

Academic Editors: Jinho Kim and Young-ho Park

Received: 29 November 2021
Accepted: 14 January 2022
Published: 18 January 2022

1. Introduction

Pairs trading is a statistical arbitrage strategy that exploits short-term price divergences between a pair of assets that have historically moved together. In the stock market, pairs traders simultaneously open a short position in an overvalued stock and a long position in an undervalued stock. When the prices of the two stocks converge, the opened positions are closed by taking the opposite positions. Figure 1 illustrates the similar price movements of a stock pair of A. O. Smith Corp. (AOS, Milwaukee, WI, USA) and Carnival Corp. (CCL, Miami, FL, USA) in the Standard & Poor's (S&P) 500 index, which is a collection of the stocks of 500 large companies in the United States. As shown in the figure, pairs trading exploits divergence and convergence movements using both long and short positions.

To find stock pairs with arbitrage opportunities, we check whether the spread, which is the difference between the prices of two stocks, forms a mean-reverting stationary process. Figure 2 illustrates the normalized spread of the stock pair shown in Figure 1. Traditionally, pairs trading methods open positions when the spread touches a trading boundary (e.g., T_1 in Figure 2), which is predetermined as a constant multiple of the standard deviation of the spread. The constant value may vary for different pairs-trading methods. When the spread returns to the mean (e.g., T_2 in Figure 2) or the specified trading window ends, the positions are closed. The spread may diverge too far from the mean (e.g., T_3 in Figure 2), possibly resulting in a great loss. To limit potential losses in such situations, pairs trading methods set stop-loss boundaries to close the positions by force.

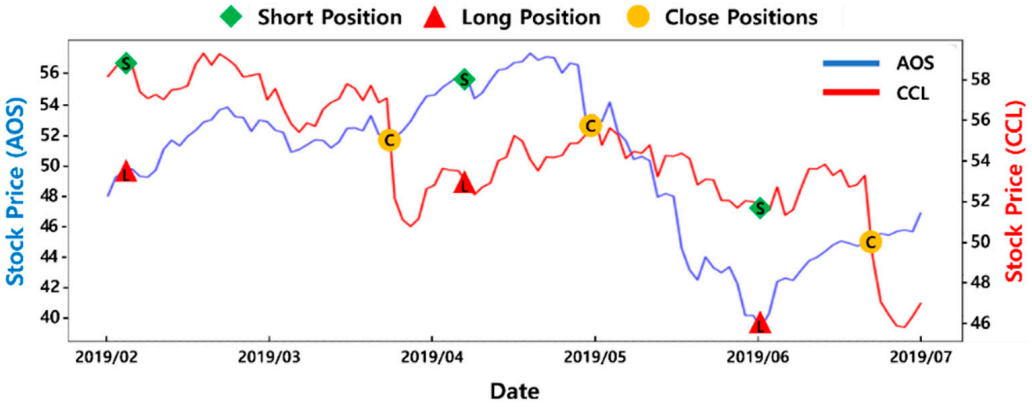

Figure 1. Similar price movements of a stock pair (AOS and CCL).

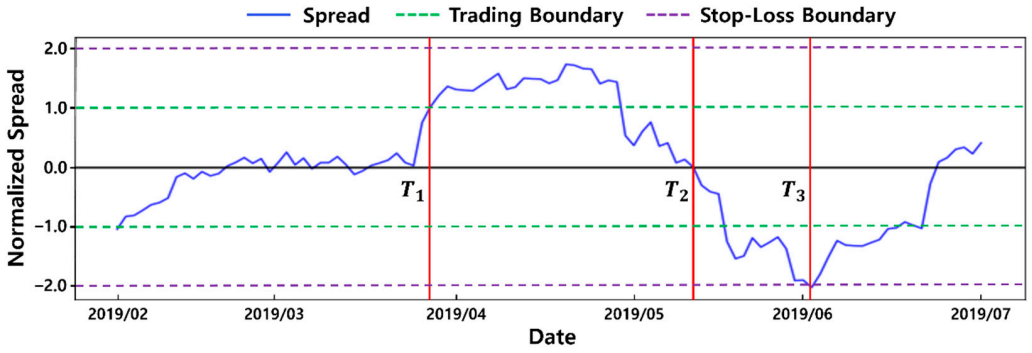

Figure 2. Normalized spread of the stock pair in Figure 1 (AOS and CCL).

Algorithmic trading enables pairs traders to execute complex trading strategies without human intervention. In recent years, deep reinforcement learning has received considerable attention in the field of algorithmic trading because it has both the perception ability of deep learning and the decision-making ability of reinforcement learning. Pairs trading methods based on deep reinforcement learning can be classified into two approaches: (1) indirectly determining trading actions, such as long and short, based on the trading and stop-loss boundaries [1,2] and (2) directly determining trading actions based on the spread [3–5]. The former approach has a fixed gap between the trading and stop-loss boundaries, as shown in Figure 2, which is certainly not optimal because the trading boundary is completely dependent on the stop-loss boundary. The latter approach has no stop-loss boundaries, which presents the risk of a great loss.

In this paper, we propose a hybrid deep reinforcement learning method for pairs trading called HDRL-Trader, which overcomes the disadvantages of the two aforementioned approaches. HDRL-Trader uses two independent reinforcement learning networks: one for determining trading actions and the other for determining stop-loss boundaries. First, it extracts robust features from observations by applying dimensionality reduction and clustering. Second, an accurate state representation is derived from the preprocessed features using state representation learning (SRL). This SRL method co-trains a regression model that predicts the next spread alongside a reinforcement learning model. Third, reinforcement learning and behavior cloning are combined to learn the behavior of a prophetic expert who sees future spread movements. Fourth, the twin delayed deep deterministic policy gradient (TD3) algorithm [6] is extended to determine trading actions for pairs trading. The SRL method, behavior cloning, prioritized experience replay (PER), and dynamic delay are all incorporated into the extended TD3 algorithm. Fifth, the double deep *Q*-network (DDQN) algorithm [7] is extended to determine stop-loss boundaries and combined with the extended TD3 algorithm. Compared with state-of-the-art pairs trading

methods, HDRL-Trader yields higher profits and has superior generalization ability for different stock pairs in the S&P 500 index.

The remainder of this paper is organized as follows. Section 2 introduces pairs trading and reinforcement learning methods, and Section 3 reviews the existing work. Sections 4 and 5 present HDRL-Trader and the experimental procedure and results, respectively. Finally, Section 6 presents our conclusions. For ease of reading, Table A2 in Appendix B lists the abbreviations used in this paper.

2. Background

2.1. Pairs Trading

Pairs trading is a popular market-neutral trading strategy that uses highly correlated and cointegrated stock pairs. If two non-stationary time series (i.e., stock prices) are cointegrated, we can combine them into one stationary time series (i.e., a spread) according to Equation (1). The augmented Dickey–Fuller test [8] is widely used to analyze whether a time series is stationary. In Equation (1), P_A and P_B are the prices of stocks A and B, respectively, and β is the cointegration factor (or hedge ratio).

$$Spread = \log P_A - \beta * \log P_B \tag{1}$$

The hedge ratio β indicates that pairs traders longs (or shorts) one share of stock A and shorts (or longs) β shares of stock B for hedging. Linear regression methods, such as ordinary least squares (OLS) and total least squares (TLS), can be used to calculate β. OLS minimizes the sum of squared errors, while TLS minimizes the sum of the squared orthogonal distances from data points to the regression line [1].

2.2. Reinforcement Learning

In reinforcement learning, an agent is a learner and decision maker that autonomously learns how to maximize the total reward by interacting with its environment through sensors that observe the environment and effectors that execute the action selected by the agent and provide the reward to the agent [9–14]. The agent develops a deterministic policy $\mu(s)$ that maps a state to an action, or a stochastic policy $\pi(a|s)$ that maps a state to a probability distribution over actions. At each time step t, the agent observes a state s_t, selects an action a_t, receives a reward r_t, and then transitions to a new state s_{t+1}. This interaction is modeled as a Markov decision process (MDP) represented by $\langle S,\ A,\ P, R, \gamma \rangle$, where S is the state space, A is the action space, $P(s_t, a_t, s_{t+1})$ is the state transition probability, $R(s, a)$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor, which determines the present value of future rewards. The sum of the discounted rewards is defined as the discounted return G_t as follows:

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i} = r_t + \gamma G_{t+1} \tag{2}$$

In algorithmic trading, states are not directly provided; rather, they must be constructed from a history of observations. We can model this by extending the MDP model with observation space O and observation probability $Z(o|s, a)$. This extended model is referred to as a partially observable MDP model [15].

2.2.1. Deep Q-Network

In value-based reinforcement learning, the agent estimates the expected discounted return, or value, for each state according to Equation (3) or for each state and action pair according to Equation (4). The ε-greedy strategy is widely used to derive a new policy π' from $Q^\pi(s, a)$. With probability $(1 - \varepsilon)$, the agent performs the action with the maximum Q-value (i.e., greedy action), that is, $\pi'(s) = \text{argmax}_{a \in A} Q^\pi(s, a)$. With probability ε, the agent performs a random action for exploration.

$$V^\pi(s) = \mathbb{E}_{a \sim \pi}[G_t | S_t = s] \tag{3}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{a\sim\pi}[G_t|S_t = s,\ A_t = a] \tag{4}$$

For a large state and action space, a deep Q-network (DQN) [16] employs deep neural networks to approximate Q-values. The techniques of experience replay and, using a target network, enable stable learning. An experience replay buffer stores transitions as tuples of $\langle s_t, a_t, r_t, s_{t+1}\rangle$, and the agent learns by sampling batches from the buffer. The online network Q_θ is periodically copied to the target network $Q_{\theta'}$ during learning. The mean squared error (MSE) between the target value Y_{DQN} in Equation (5) and $Q_\theta(s_t, a_t)$ is the loss function of the DQN (Equation (6)), which uses the Bellman equation [17].

$$Y_{\mathrm{DQN}} = r + \gamma \max_a Q_{\theta'}(s_{t+1}, a) \tag{5}$$

$$Loss_{\mathrm{DQN}} = \mathbb{E}\left[\left(Y_{\mathrm{DQN}} - Q_\theta(s_t, a_t)\right)^2\right] \tag{6}$$

The DDQN solves the overestimation problem of the DQN by selecting the greedy action using Q_θ and estimating the Q-value of the action using $Q_{\theta'}$, as shown in Equation (7).

$$Y_{\mathrm{DDQN}} = r + \gamma Q_{\theta'}\left(s_{t+1},\ \operatorname*{argmax}_a Q_\theta(s_{t+1}, a)\right) \tag{7}$$

The PER [18] samples important transitions more frequently to learn more efficiently. The probability $P(i)$ of transition i being sampled is defined in Equation (8). The p_i in Equation (8) is the priority of transition i, which is computed using the temporal difference error $\delta_i = Y_{\mathrm{DDQN}} - Q_\theta(s,a)$. There are two variants of prioritization: proportional and rank-based prioritization. In proportional prioritization, $p_i = |\delta_i| + \xi$, where ξ is a small positive number used to ensure a non-zero probability for transitions with $\delta_i = 0$. In rank-based prioritization, $p_i = \frac{1}{rank(i)}$, where $rank(i)$ is the rank of transition i when the replay buffer is sorted by $|\delta_i|$. To guarantee that every transition is sampled at least once, new transitions have the maximum priority. The α in Equation (8) determines the extent to which the probability is affected by priority. By adjusting α, we can interpolate between greedy prioritization and uniform sampling ($\alpha = 0$).

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{8}$$

Non-uniform sampling in the PER results in a biased estimate of Q-values. To correct this bias, the PER employs importance sampling weights according to Equation (9), where B is the size of the replay buffer, β is the bias-annealing factor for annealing the amount of importance sampling correction over time, and $\max_k \omega_k$ is the maximum weight for normalizing weights. The weights are incorporated into the Q-learning updates by using $\omega_i \delta_i$ instead of δ_i.

$$\omega_i = \left(B \cdot P(i)\right)^{-\beta} / \max_k \omega_k \tag{9}$$

Rainbow DQN [19] is a comprehensive improvement of the DQN that combines several techniques, such as the DDQN, PER, dueling networks [20], multistep learning [21], distributional reinforcement learning [22], and noisy networks [23].

2.2.2. Deep Deterministic Policy Gradient

In policy-based reinforcement learning, the agent directly learns a stochastic or deterministic policy. Actor-critic methods combine the advantages of value- and policy-based methods, where the critic network estimates the state value or Q-value, and the actor network updates the policy in the direction suggested by the critic. In stochastic actor-critic methods, the loss function of the critic network (V_θ) is defined using the Bellman equation in Equation (11), where Y is the target value defined in Equation (10). The loss function

of the actor network (π_ϕ) is defined using the stochastic policy gradient theorem [24] according to Equation (12).

$$Y = r + \gamma V_\theta(s_{t+1}) \tag{10}$$

$$Loss_{critic} = \mathbb{E}[(Y - V_\theta(s_t))^2] \tag{11}$$

$$Loss_{actor} = \mathbb{E}\big[-\log \pi_\phi(Y - V_\theta(s_t))\big] \tag{12}$$

The deterministic policy gradient (DPG) [25] is an actor-critic method that aims to find an optimal deterministic policy $\mu_\phi(s)$. The deep DPG (DDPG) [26] combines the DPG and DQN. It employs the experience replay and target network of the DQN, but it does not use the ϵ-greedy strategy. For the exploration, random noise $\mathcal{N}$ is added to the policy, as shown in Equation (13). The loss function of the critic network is defined using the Bellman equation, as in Equation (15), where Y is the target value defined in Equation (14). The loss function of the actor network is defined using the DPG theorem [25], as shown in Equation (16). The target networks are soft updated, as shown in Equation (17).

$$a_t = \mu_\phi(s_t) + \mathcal{N} \tag{13}$$

$$Y = r + \gamma Q_{\theta'}\Big(s_{t+1}, \mu_{\phi'}(s_{t+1})\Big) \tag{14}$$

$$Loss_{critic} = \mathbb{E}[(Y - Q_\theta(s_t, a_t))^2] \tag{15}$$

$$Loss_{actor} = \mathbb{E}\big[-Q_\theta\big(s_t, \mu_\phi(s_t)\big)\big] \tag{16}$$

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta',\ \phi' \leftarrow \tau\phi + (1-\tau)\phi' \tag{17}$$

2.2.3. Twin Delayed DDPG

TD3 enhances the DDPG using three techniques. The first technique is known as target policy smoothing, which is a regularization strategy. To avoid overfitting, random noise is added to the target action, as in Equation (18), where clipping is used to limit the impact of the noise. The target action is used to compute the target value Y in Equation (19).

$$\widetilde{a}_{t+1} = \mu_{\phi'}(s_{t+1}) + \epsilon,\ \ \epsilon \ \sim\ clip(\mathcal{N}, -c, c) \tag{18}$$

The second technique is known as clipped double Q-learning. To solve the overestimation problem of the DDPG, two critic networks (and two target critic networks) are used. For computing the target value Y, the minimum Q-value of the two target critic networks is used, as shown in Equation (19).

$$Y = r + \gamma \min_{j=1,2} Q_{\theta'_j}(s_{t+1}, \widetilde{a}_{t+1}) \tag{19}$$

The third technique is known as delayed policy updates, which updates the actor (and target) networks less frequently than the critic networks to obtain more accurate Q-values before updating the actor network.

3. Related Work

3.1. Algorithmic Trading

Considerable studies have been conducted on algorithmic trading, including on supervised learning-based methods [27–32] and reinforcement learning-based methods [33–36]. Supervised learning-based methods predict stock prices but do not decide trading actions. Reinforcement learning-based methods can decide trading actions by learning a profitable trading policy. Fengqian and Chao [33] proposed a deep reinforcement learning method that uses a candlestick (or K-line) as a summary of price movements. In this method, a candlestick is decomposed into the lengths of the upper shadow line, lower shadow line, and body. After applying clustering to each component, the cluster centers and body color are used to represent the input state for reinforcement learning. Lei et al. [34] pro-

posed a time-driven feature-aware jointly deep reinforcement learning algorithm, which uses a gate structure, gated recurrent unit (GRU) [37] network, temporal attention mechanism, and auto-encoder that predicts the next closing price, where the encoding part of the auto-encoder is used for state representation. Liu et al. [35] proposed an imitative deep reinforcement learning method that uses a demonstration buffer and behavior cloning for imitation learning. Park and Lee [36] proposed a practical algorithmic trading method called SIRL-Trader, which achieves good profit using only long positions. SIRL-Trader uses offline/online SRL, imitative reinforcement learning, multistep learning, and dynamic delay.

In this work, we study a pairs-trading problem based on hybrid reinforcement learning, which is a fundamentally different problem from those of the above studies. This fundamental difference raises new challenges, such as how to define action spaces and reward functions for pairs trading, how to clone the behavior of a prophetic pairs-trading expert, and how to hybridize two reinforcement learning algorithms, as described in Section 4.

3.2. Pairs Trading

Deep reinforcement learning methods for pairs trading can be classified into two approaches. The first approach [1,2] indirectly determines trading actions based on trading and stop-loss boundaries. Kim and Kim [1] proposed the pairs trading DQN (PTDQN) algorithm, which dynamically optimizes the boundaries for daily stock data. The action space consists of six predetermined boundary pairs; for example, the action $A0$ is defined as (trading boundary = ± 0.5, stop-loss boundary = ± 2.5). The gap between the trading and stop-loss boundaries is fixed at 2.0 for all actions. Lu et al. [2] focused on intraday trading, where the cointegration relationship is much weaker than that of interday trading. To detect structural breaks in which the cointegration relationship vanishes, the authors proposed a spread wavelet-aware hybrid network that combines a continuous wavelet convolutional neural network [38] for frequency-domain features and a long short-term memory (LSTM) network [39] for time-domain features. The authors also proposed a structural break-aware DQN algorithm to determine the trading and stop-loss boundaries. The action space consists of six predetermined boundary pairs and a hold action, and the gap between the trading and stop-loss boundaries is fixed at 2.0. In the above methods, the trading boundary is completely dependent on the stop-loss boundary, which is certainly not optimal.

The second approach [3–5] directly determines trading actions based on the spread. Brim [3] used the DDQN to determine trading actions based on the technical indicators of the spread, and to reduce trading actions with negative rewards, it multiplies negative rewards by a large constant value. Wang, Sandås, and Beling [4] used the deuling DQN algorithm and proposed a reward shaping method that employs a baseline policy with a fixed trading boundary as a guidance to learn a robust policy and reduce overfitting. Kim, Park, and Lee [5] proposed a DDQN-based pairs trading method called P-Trader that uses technical indicators for the spread, the candlestick clustering technique used in [33], a gate structure, a GRU network, a temporal attention mechanism, and the auto-encoder technique used in [34]. The above methods do not have stop-loss boundaries, which leads to high risk.

In this paper, we propose a hybrid reinforcement learning method that combines the above two approach types, which is the first one to do so to the best of our knowledge. First, we extend the TD3 algorithm to directly determine trading actions by incorporating the auto-encoder technique, behavior cloning, PER, and dynamic delay. Second, we extend the DDQN algorithm to determine stop-loss boundaries and combine it with the extended TD3 algorithm.

4. Hybrid Deep Reinforcement Learning for Pairs Trading

In this section, we propose the novel pairs trading method called HDRL-Trader, which uses hybrid deep reinforcement learning.

4.1. Architecture of HDRL-Trader

Figure 3 illustrates the architecture of HDRL-Trader for pairs trading in a stock market environment. For data preprocessing, we apply dimensionality reduction and clustering to extract robust features. For SRL, we use a gate structure, LSTM layer, and regression network to generate states from the features. For hybrid reinforcement learning, we combine extended versions of the TD3 and DDQN algorithms. Each component is explained in detail in the following subsections.

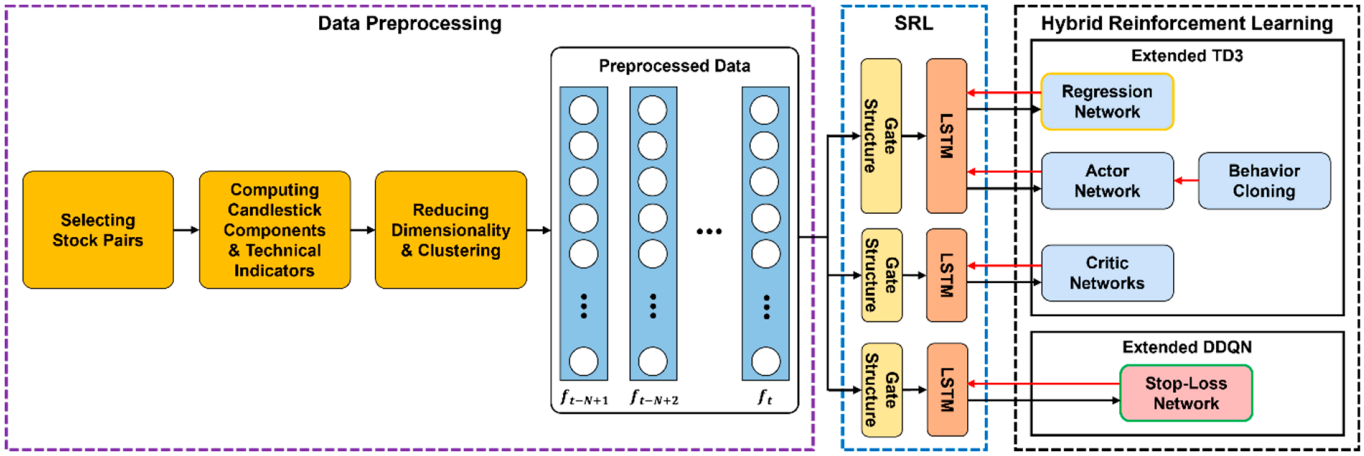

Figure 3. Architecture of the proposed HDRL-Trader.

4.2. Data Preprocessing

We select stock pairs through correlation and cointegration tests. To test the cointegration relationship, the Engle-Granger two-step test [40] is widely used [41–43]. According to the Engle-Granger test, we calculate the spread for closing prices of stocks using the OLS regression method and performs a stationarity test using the augmented Dickey-Fuller test. For large datasets, pre-selection based on Pearson's correlation coefficient is a widely used technique to reduce computationally expensive cointegration tests [43–46]. We apply this technique using normalized closing prices.

For data preprocessing, we extract a low-dimensional feature vector from the spread data, as shown in Figure 4. First, we compute the candlestick components and technical indicators listed in Table 1 from the spread data, which consist of opening, high, low, closing, and volume spreads. Second, we normalize the input features using the z-score standardization method. Third, we apply dimensionality reduction to each feature group in Table 1. We reduce the dimensionality to the threshold F in Figure 4 using principal component analysis (PCA). Fourth, we cluster each feature generated by PCA using fuzzy c-means clustering [47]. After clustering, we represent each feature value, except the body color, as the cluster center to which it belongs.

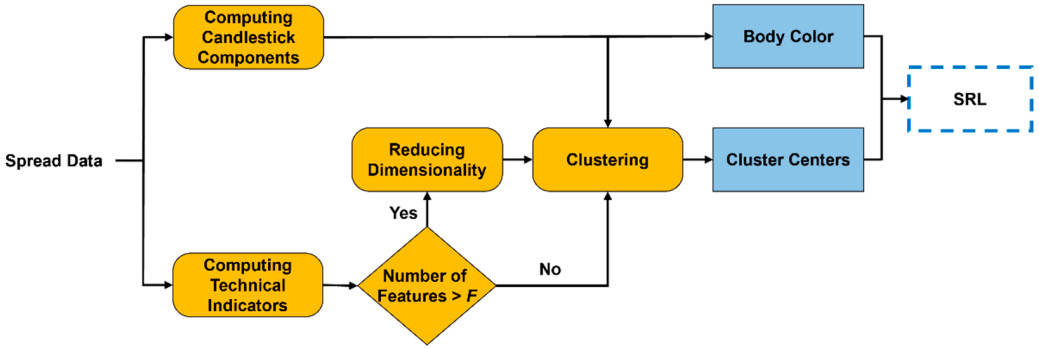

Figure 4. Data preprocessing.

Table 1. Input features.

Feature Group	Features
Candlestick components [33]	Lengths of the upper shadow line, lower shadow line, and body; body color
Overlap studies [48]	BBANDS, DEMA, EMA, HT-TRENDLINE, KAMA, MA, MAMA, MIDPOINT, MIDPRICE, SAR, SAREXT, SMA, T3, TEMA, TRIMA, WMA
Momentum indicators [48]	ADX, ADXR, APO, AROON, AROONOSC, BOP, CCI, CMO, DX, MACD, MACDEXT, MACDFIX, MFI, MINUS_DI, MINUS_DM, MOM, PLUS_DI, PLUS_DM, PPO, ROC, ROCP, ROCR, RSI, STOCH, STOCHF, STOCHRSI, TRIX, ULTOSC, WILLR
Volume indicators [48]	AD, ADOSC, OBV
Volatility indicators [48]	ATR, NATR, TRANGE

4.3. State Representation Learning

State representation is crucial for reinforcement learning. To observe historical spread movements, our SRL model takes a sliding window of preprocessed data, as shown in Figure 3. To adaptively select features, we assign a weight to each feature using the gate structure [34]. The gate g shown in Figure 5a is defined as in Equation (20) where f is the input feature vector, W and b are parameters learned by end-to-end training, and σ is a sigmoid activation function. Using gate g, we generate the weighted feature vector f' from the element-wise multiplication ($\odot$) of vectors g and f in Equation (21).

$$g = \sigma(W \cdot f + b) \tag{20}$$

$$f' = g \odot f \tag{21}$$

The LSTM network layer shown in Figure 5b takes a sliding window of weighted feature vectors as the input and outputs the hidden state of the last time step. We employ the LSTM network because it can effectively capture the long-term dependency in the time series and has shown superior performance in stock price prediction over other types of deep neural networks and traditional machine learning algorithms [49–51]. We refer to the network up to the LSTM layer as the SRL network. The output of the LSTM layer is used in reinforcement learning, as well as in a regression network that predicts the next closing spread to provide accurate state information. The predicted spread does not participate in reinforcement learning, but the underlying SRL network does because it is shared with the actor network, as explained in the next section.

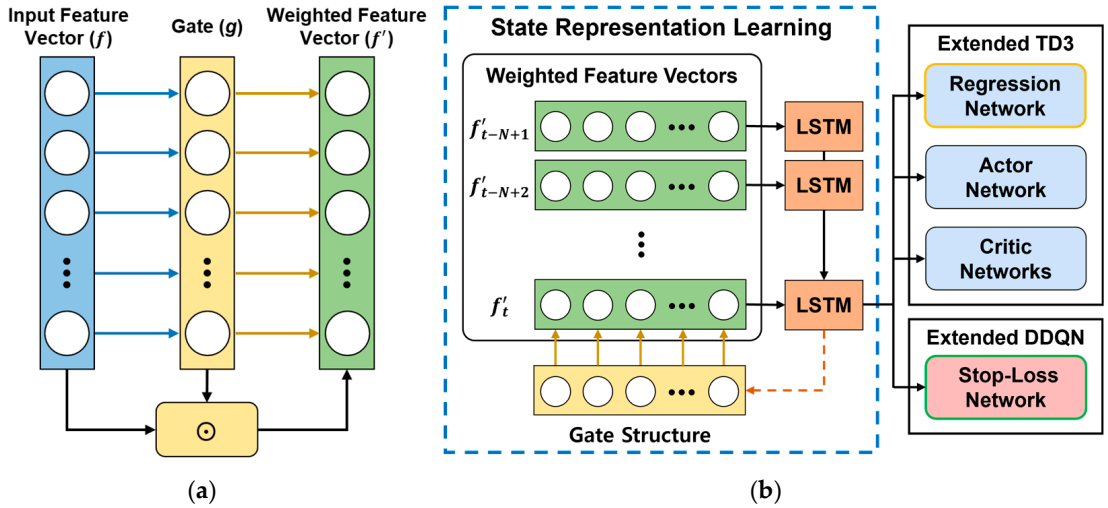

Figure 5. State representation learning. (**a**) Gate structure. (**b**) Architecture of the SRL.

4.4. Hybrid Reinforcement Learning

We extend the TD3 algorithm to directly determine trading actions. We incorporate the proposed SRL model, behavior cloning, PER, and dynamic delay into this extended TD3 algorithm. To determine stop-loss boundaries, we extend the DDQN algorithm and combine it with the extended TD3 algorithm.

4.4.1. Action Spaces for Pairs Trading

The agent starts with a certain amount of money and allocates half of the total to each of the two stocks in a pair. When the positions are closed and cashed out, the agent reallocates half of the total money to each stock. For each trading day t, the extended TD3 algorithm determines the trading action $a_t^{ta} \in \{long_A/short_B, hold, short_A/long_B\}$. The $long_A/short_B$ action longs an undervalued stock A and shorts an overvalued stock B. The $short_A/long_B$ action is the opposite of $long_A/short_B$, and the $hold$ action takes no position. For the sake of simplicity, the agent longs/shorts the maximum number of shares of a stock at the closing price. The extended DDQN algorithm determines the stop-loss boundary $a_t^{sl} \in \{\pm 1.5, \pm 2.0, \pm 2.5, \pm 3.0, \pm 3.5\}$. When the stop-loss boundary is met by the spread, the trading action a_t^{ta} is ignored and overridden. Opened positions are closed when (1) the spread returns to the mean (referred to as normal close), (2) the spread does not return to the mean during the trading window (referred to as exit), or (3) the spread reaches the stop-loss boundary (referred to as stop-loss close).

4.4.2. Reward Functions for Pairs Trading

The reward function for the trading actions is designed to consider both the risk of exit and transaction cost. An exit usually results in a loss because the positions are closed by force at the end of the trading window (or the exit time), even if the spread has not returned to the mean. In each trading window, the ratio RT_t^{exit} of time interval between the exit time t_{exit} and position closing time t_{close} is computed by Equation (22). If the positions are closed near the exit time (i.e., RT_t^{exit} is small), we consider that the risk of exit is high.

$$RT_t^{exit} = \frac{t_{exit} - t_{close}}{|trading\ window|} \tag{22}$$

In a real trading environment, there are transaction costs, such as fees, taxes, and trading slippages. Here, slippage is the difference between the expected price and actual price at which a trade is executed. The transaction cost is factored into the stock price,

as in Equation (23), where ζ is the transaction cost rate used to discount the price. The discounted price $\breve{price}_t$ is applied when the stocks are sold.

$$\breve{price}_t = price_t \times (1 - \zeta) \tag{23}$$

We use the returns of the long and short positions to define the reward function. The return R_t^{long} for the long position is defined in Equation (24), where n^{long} is the number of shares, and t_{close} and t are the position closing time and current time, respectively. The difference in stock prices is normalized by the stock price at t. The stock price at t_{close} is discounted by the transaction cost rate because the stock is sold at t_{close}. Similarly, the return R_t^{short} for the short position is defined in Equation (25).

$$R_t^{long} = n^{long} \times \left(\breve{price}_{t_{close}}^{long} - price_t^{long} \right) / price_t^{long} \tag{24}$$

$$R_t^{short} = n^{short} \times \left(\breve{price}_t^{short} - price_{t_{close}}^{short} \right) / \breve{price}_t^{short} \tag{25}$$

We define the reward r_t^{ta} for trading actions in Equation (26). For a normal close, the reward is the portfolio return $(R_t^{long} + R_t^{short})$ multiplied by RT_t^{exit}, which is designed to reduce the risk of exit. For an exit or stop-loss close, the reward is the portfolio return itself, which is usually negative.

$$r_t^{ta} = \begin{cases} \left(R_t^{long} + R_t^{short} \right) \times RT_t^{exit} & \text{if a normal close occurs} \\ R_t^{long} + R_t^{short} & \text{if an exit or stop-loss close occurs} \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

We define the reward r_t^{sl} for the stop-loss boundary in Equation (27). When the current spread S_t touches or transgresses the stop-loss boundary a_t^{sl} (i.e., $S_t / a_t^{sl} \geq 1$), the reward is computed as the rate of change between the current spread S_t and next spread S_{t+1}. If S_{t+1} diverges more than S_t (i.e., $S_{t+1}/S_t > 1$), the reward is positive, which indicates that the stop-loss boundary is correctly determined. If $S_{t+1}/S_t < 1$, then the reward is negative. When the current spread S_t diverges more than the spread S_o at the position opening (i.e., $S_t/S_o > 1$), the reward is computed as the rate of change between S_o and S_t. In this case, the reward is always negative, which indicates that the stop-loss boundary is incorrectly determined.

$$r_t^{sl} = \begin{cases} \frac{S_{t+1} - S_t}{S_t} & \text{if } \frac{S_t}{a_t^{sl}} \geq 1 \\ \frac{S_o - S_t}{S_o} & \text{if } \frac{S_t}{S_o} > 1 \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

4.4.3. Behavior Cloning

We employ a behavior cloning technique for actor network training, which learns the actions of a prophetic trading expert. The expert determines an action on day_t using information about future spread movements. The expert sets a trading boundary tb for opening positions, which is a hyperparameter. If the current spread S_t touches or transgresses tb (i.e., $S_t/tb \geq 1$), and the future spread returns to the mean within the trading window, TW, a normal close is guaranteed. Thus, the expert selects $short_A/long_B$ or $long_A/short_B$ on day_t depending on the sign of S_t. Otherwise, the expert selects the *hold* action. Figure 6 illustrates how the expert acts. The expert's action at t_2 is $short_A/long_B$ because the spread returns to the mean within $t_2 + |TW|$. However, at t_1, the spread does not return to the mean within $t_1 + |TW|$, so the *hold* action is selected. Similarly, the selected actions at t_3 and t_4 are *hold* and $long_A/short_B$, respectively.

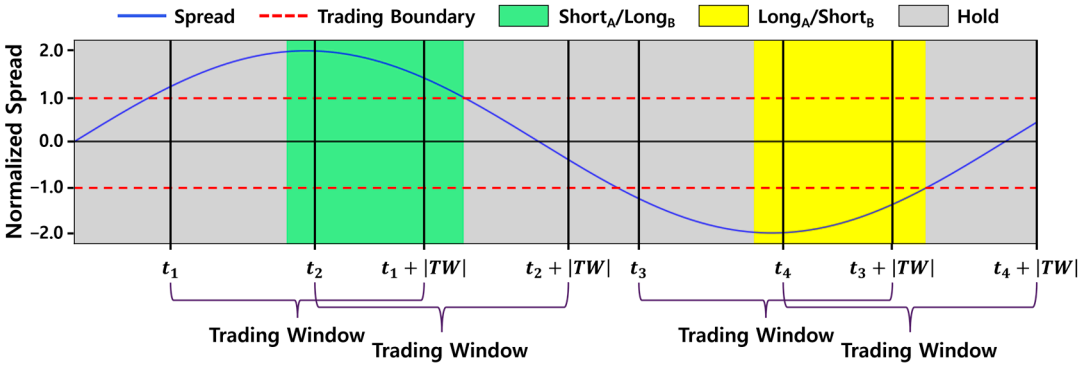

Figure 6. An example of the expert's actions.

4.4.4. Hybrid Reinforcement Learning Algorithm

Figure 7 illustrates the proposed architecture for hybrid reinforcement learning, which combines the extended TD3 and DDQN algorithms. The extended TD3 algorithm uses an actor network μ_ϕ, SRL network λ_υ for μ_ϕ, regression network that shares λ_υ with the actor network, two critic networks Q_{θ_1} and Q_{θ_2}, and two SRL networks o_{η_1} and o_{η_2} for Q_{θ_1} and Q_{θ_2}, respectively. The actor network is trained with behavior cloning. The extended DDQN algorithm uses a Q-network Q_ψ for the stop-loss boundary (called the stop-loss network) and the SRL network κ_ι for Q_ψ. All networks, except the regression network, have corresponding target networks. In Figure 7, the SRL networks correspond to the sensors, and the trading system corresponds to the effector. Figure 8 shows the structure of each network.

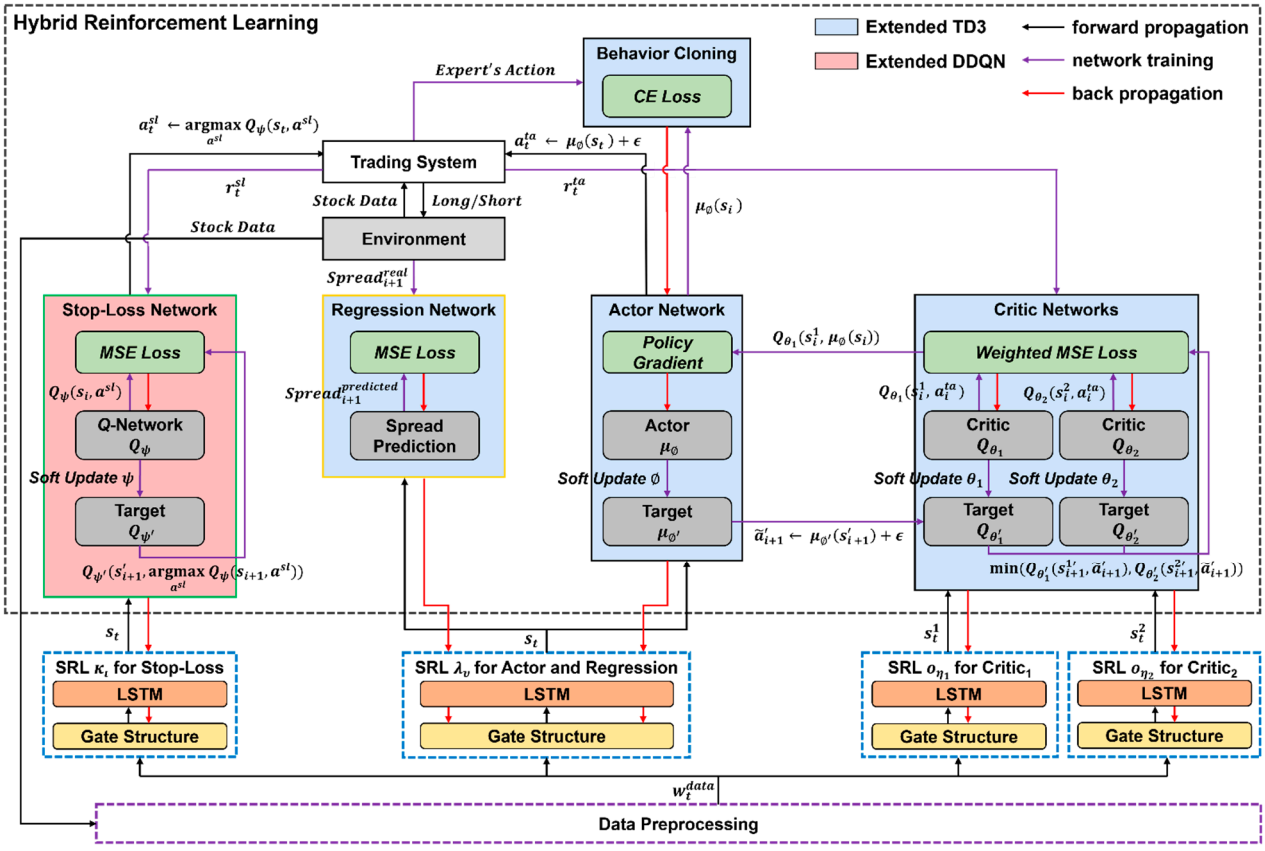

Figure 7. Architecture of the hybrid deep reinforcement learning.

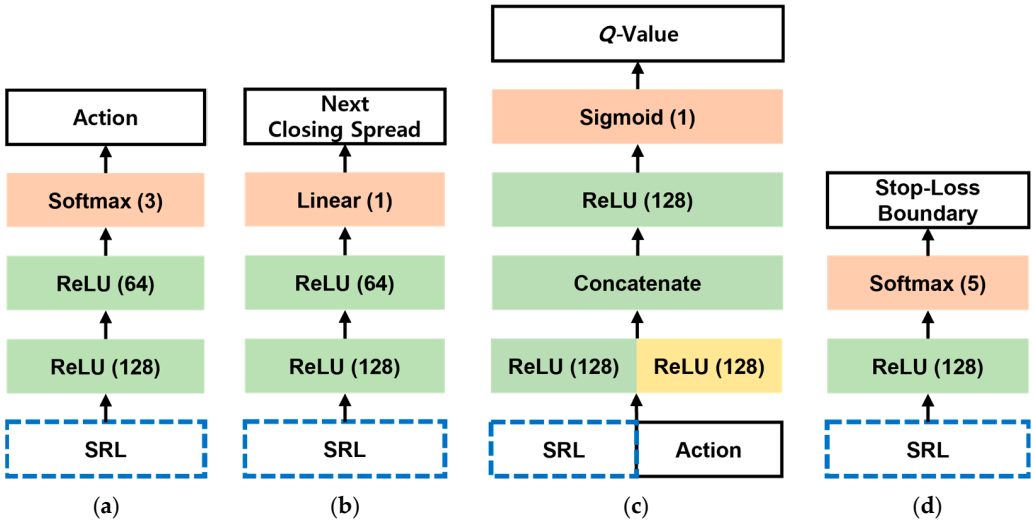

Figure 8. Neural network structures. (**a**) Actor. (**b**) Regression. (**c**) Critics. (**d**) Stop-Loss.

Algorithm 1 presents the proposed hybrid reinforcement learning algorithm. The input for the SRL networks is a sliding window w_t^{data} of weighted feature vectors $\{x_{t-N_w+1}, \ldots, x_{t-1}, x_t\}$ obtained from data preprocessing. After interacting with the environment, a transition of $\left\langle w_t^{data}, a_t^{ta}, a_t^{sl}, r_t^{ta}, r_t^{sl}, w_{t+1}^{data}, p_t \right\rangle$ is stored to $\mathcal{B}$ where the priority p_t is set to the maximum (line 13). Using the sampling probability in Equation (8), a minibatch of transitions is sampled (line 14).

In the extended TD3 algorithm, the actor network μ_ϕ is combined with the SRL network λ_υ, and updates on μ_ϕ are back-propagated to λ_υ. In the actor network shown in Figure 8a, a softmax layer is used as the output layer to support discrete actions. To select an action with exploration noise, a different amount of noise is added to each output of the softmax layer (line 9), and then argmax is applied to the outputs. To avoid overfitting, we employ the target policy smoothing technique of TD3 (line 15). To provide accurate state information to the actor network, we train the regression network, whose structure is shown in Figure 8b, by minimizing the MSE between the real and predicted closing spreads (line 24). The actor network is indirectly affected by this training through the underlying SRL network λ_υ, which is shared with the regression network. To learn the expert's action a_{expert}^{ta}, we train the actor network by minimizing the cross-entropy (CE) loss between the softmax output vector of a_i^{ta} and the expert's action a_{expert}^{ta}, which is represented as a one-hot vector (line 25).

The critic networks Q_{θ_1} and Q_{θ_2}, whose structures are shown in Figure 8c, are combined with the corresponding SRL networks o_{η_1} and o_{η_2}, respectively. To avoid overestimation, we employ the clipped double Q-learning technique of TD3 when computing the target value Y_{critic} (line 16). Y_{critic} is used in computing the temporal difference errors (line 17). The critic networks are updated by minimizing the MSE loss with the importance sampling weight ω_i in Equation (9) (line 18). The updates on each critic network are back-propagated to the corresponding SRL network. The transition priority p_i is updated using the minimum of the temporal difference errors (line 19).

In contrast to the static delay used in the delayed policy updates technique of TD3, the dynamic delay technique [36] is applied when updating the actor and target networks for more stable and efficient training. At each epoch, the delay value d is computed using Equation (28), where c and b are parameters for controlling the variance and minimum delay value, respectively (line 7).

$$d = (e \bmod c) + b \tag{28}$$

In the extended DDQN algorithm (red parts in Algorithm 1), the stop-loss network Q_ψ, whose structure is shown in Figure 8d, is combined with the SRL network κ_ι shown in Figure 7. To support discrete stop-loss boundaries, a softmax layer is used as the output layer of the stop-loss network. An action a_t^{sl} is selected using the ε-greedy strategy (lines 10 and 11). The target value Y_{sl} is computed as in the DDQN, and then, the stop-loss network is updated using Y_{sl} (lines 20 and 21). When the stop-loss network is updated, the SRL network κ_ι is also updated by backpropagation.

Algorithm 1. The hybrid reinforcement learning algorithm of HDRL-Trader

Input: Sliding-window data $w_t^{data} \leftarrow \{x_{t-N_w+1}, \ldots, x_{t-1}, x_t\}$ obtained from the data preprocessing
Output: actor network μ_ϕ, SRL network λ_v for μ_ϕ, stop-loss network Q_ψ, SRL network κ_ι for Q_ψ

1. Initialize an actor network μ_ϕ, an SRL network λ_v for μ_ϕ, and a regression network
2. Initialize critic networks Q_{θ_1}, Q_{θ_2}, SRL networks o_{η_1}, o_{η_2} for Q_{θ_1}, Q_{θ_2}
3. Initialize a stop-loss network Q_ψ, an SRL network κ_ι for Q_ψ
4. Initialize target networks : $\phi' \leftarrow \phi$, $v' \leftarrow v$, $\theta'_{1,2} \leftarrow \theta_{1,2}$, $\eta'_{1,2} \leftarrow \eta_{1,2}$, $\psi' \leftarrow \psi$, $\iota' \leftarrow \iota$
5. Initialize a prioritized replay buffer $\mathcal{B}$

6. **for** $e = 1$ to N_{epochs} **do**
7. Compute the delay value for each epoch: $d \leftarrow (e \bmod c) + b$

8. **for** $t = 1$ to $T - 1$ **do**
9. Select a trading action a_t^{ta} with exploration noise $\epsilon \backsim \mathcal{N}(0, \sigma)$: $a_t^{ta} \leftarrow \mu_\phi(s_t) + \epsilon$ where $s_t = \lambda_v\left(w_t^{data}\right)$
10. With probability ε, select a random stop-loss boundary a_t^{sl}
11. Otherwise, select $a_t^{sl} \leftarrow \underset{a^{sl}}{\text{argmax}} Q_\psi\left(s_t, a^{sl}\right)$ where $s_t = \kappa_\iota\left(w_t^{data}\right)$
12. Observe rewards r_t^{ta}, r_t^{sl} and the next input w_{t+1}^{data}
13. Store a transition $\langle w_t^{data}, a_t^{ta}, a_t^{sl}, r_t^{ta}, r_t^{sl}, w_{t+1}^{data}, p_t\rangle$ to $\mathcal{B}$ where $p_t = \max_{i<t} p_i$

14. Sample a minibatch of B transitions $\langle w_i^{data}, a_i^{ta}, a_i^{sl}, r_i^{ta}, r_i^{sl}, w_{i+1}^{data}, p_i\rangle$ from $\mathcal{B}$ with $P(i)$ in Equation (8)
15. Smooth the target policy with $\epsilon \backsim \text{clip}(\mathcal{N}(0, \sigma'), -c, c)$: $\widetilde{a}'_{i+1} \leftarrow \mu_{\phi'}\left(s'_{i+1}\right) + \epsilon$ where $s'_{i+1} = \lambda_{v'}\left(w_{i+1}^{data}\right)$
16. $Y_{critic} \leftarrow r_i^{ta} + \gamma \min_{j=1,2} Q_{\theta'_j}\left(s_{i+1}^{j'}, \widetilde{a}'_{i+1}\right)$ where $s_{i+1}^{j'} = o_{\eta'_j}\left(w_{i+1}^{data}\right)$
17. Compute temporal difference errors: $\delta_i^j \leftarrow Y_{critic} - Q_{\theta_j}\left(s_i^j, a_i^{ta}\right)$ where $s_i^j = o_{\eta_j}\left(w_i^{data}\right)$
18. Update the critics θ_j by the MSE loss $\frac{1}{B}\sum \left(\delta_i^j\right)^2$ with the importance sampling weight ω_i in Equation (9)
19. Update the transition priority: $p_i \leftarrow \min_{j=1,2}\left(\left|\delta_i^j\right| + \xi\right)$
20. $Y_{sl} \leftarrow r_i^{sl} + \gamma Q_{\psi'}(s'_{i+1}, \underset{a^{sl}}{\text{argmax}} Q_\psi\left(s_{i+1}, a^{sl}\right))$ where $s'_{i+1} = \kappa_{\iota'}\left(w_{i+1}^{data}\right)$, $s_{i+1} = \kappa_\iota\left(w_{i+1}^{data}\right)$
21. Update the stop-loss network ψ by the MSE loss: $\frac{1}{B}\sum \left(Y_{sl} - Q_\psi\left(s_i, a_i^{sl}\right)\right)^2$ where $s_i = \kappa_\iota\left(w_i^{data}\right)$

22. **if** t mod d **then**
23. Update the actor ϕ by the deterministic policy gradient:
$\frac{1}{B}\sum \nabla Q_{\theta_1}(s_i^1, a_i^{ta})$ where $s_i^1 = o_{\eta_1}\left(w_i^{data}\right)$, $a_i^{ta} = \mu_\phi\left(\lambda_v\left(w_i^{data}\right)\right)$
24. Update the regression network by the MSE loss: $\frac{1}{B}\sum \left(spread_{i+1}^{real} - spread_{i+1}^{predicted}\right)^2$
25. Update the actor ϕ by the CE loss for the behavior cloning: $\frac{1}{B}\sum \nabla CE\left(a_i^{ta}, a_{expert}^{ta}\right)$ where $a_i^{ta} = \mu_\phi\left(\lambda_v\left(w_i^{data}\right)\right)$
26. Soft-update the target networks: $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$, $v' \leftarrow \tau v + (1-\tau)v'$,
$\theta'_{1,2} \leftarrow \tau\theta_{1,2} + (1-\tau)\theta'_{1,2}$, $\eta'_{1,2} \leftarrow \tau\eta_{1,2} + (1-\tau)\eta'_{1,2}$,
$\psi' \leftarrow \tau\psi + (1-\tau)\psi'$, $\iota' \leftarrow \tau\iota + (1-\tau)\iota'$
27. **end if**
28. **end for**
29. **end for**

4.5. Discussion

Table 2 summarizes the pairs trading methods in terms of their data preprocessing, SRL, and reinforcement learning abilities. The notation "○" in Table 2 indicates that the corresponding technique is used, and "×" indicates that it is not used. HDRL-Trader is the only method that integrates all novel techniques (dimensionality reduction, clustering, gate structure, regression model, behavior cloning, PER, dynamic delay, and hybrid reinforcement learning).

Table 2. Comparison of pairs trading methods.

Methods	Data Preprocessing		State Representation Learning			Reinforcement Learning		
	Dim. Reduction	Clustering	Gating	Regression	Behavior Cloning	PER	Dynamic Delay	Hybrid
HDRL-Trader	○	○	○	○	○	○	○	○
P-Trader [5]	×	○	○	○	×	×	×	×
PTDQN [1]	×	×	×	×	×	×	×	×
P-DDQN [3]	×	×	×	×	×	×	×	×

5. Experiments and Results

5.1. Experimental Setup

5.1.1. Datasets for Training and Testing

We evaluate pairs trading methods using two datasets with different numbers of stock pairs in the S&P 500 index. The stock data are obtained from Yahoo Finance [52]. Stock pairs are selected if their absolute Pearson's correlation coefficient is greater than or equal to 0.85, and the *p*-value of their augmented Dickey-Fuller test is less than or equal to 0.05 for the training period. A smaller dataset with diverse price trends is used to compare HDRL-Trader with other methods in detail. The smaller dataset consists of six stock pairs with three different price trends in the test period (upward, sideways, and downward), as shown in Figures 9–11, respectively. A larger dataset of 20 stock pairs (Table A1 in Appendix A) is used to verify the generalization ability of the methods. For the two datasets, the period of the training data is from January 2013 to December 2018 (1510 days), and the period of the test data is from January 2019 to December 2020 (504 days).

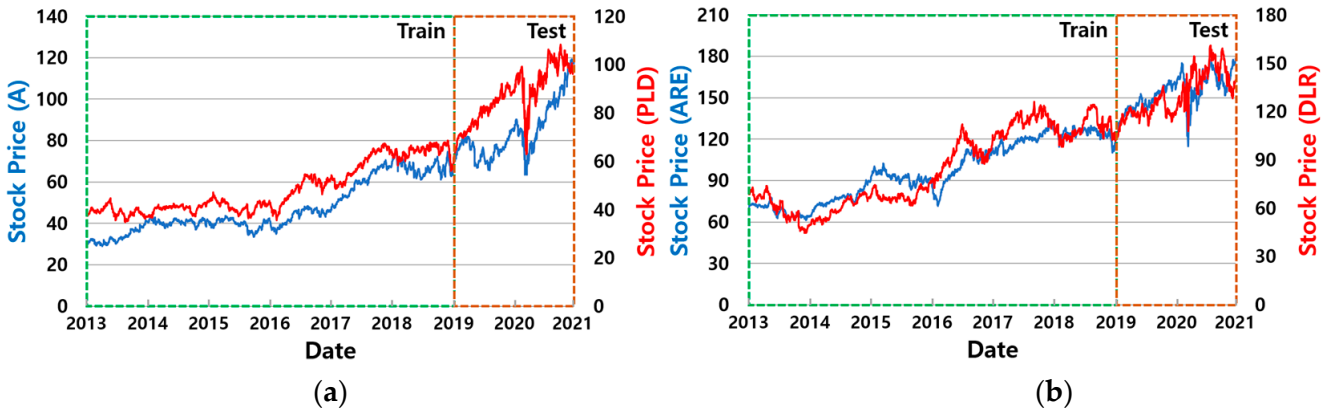

Figure 9. Stock pairs trending upward. (**a**) A, PLD. (**b**) ARE, DRL.

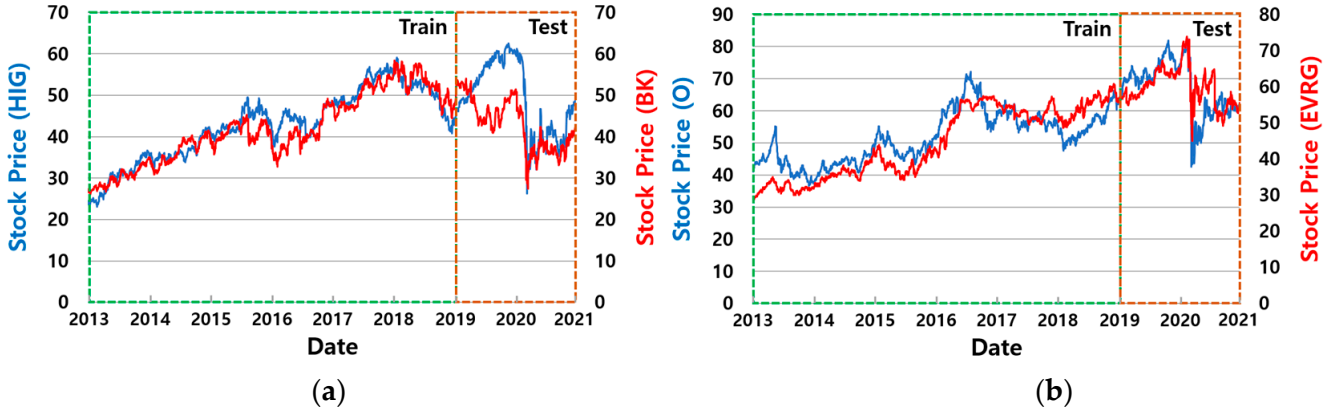

Figure 10. Stock pairs trending sideways. (**a**) HIG, BK. (**b**) O, EVRG.

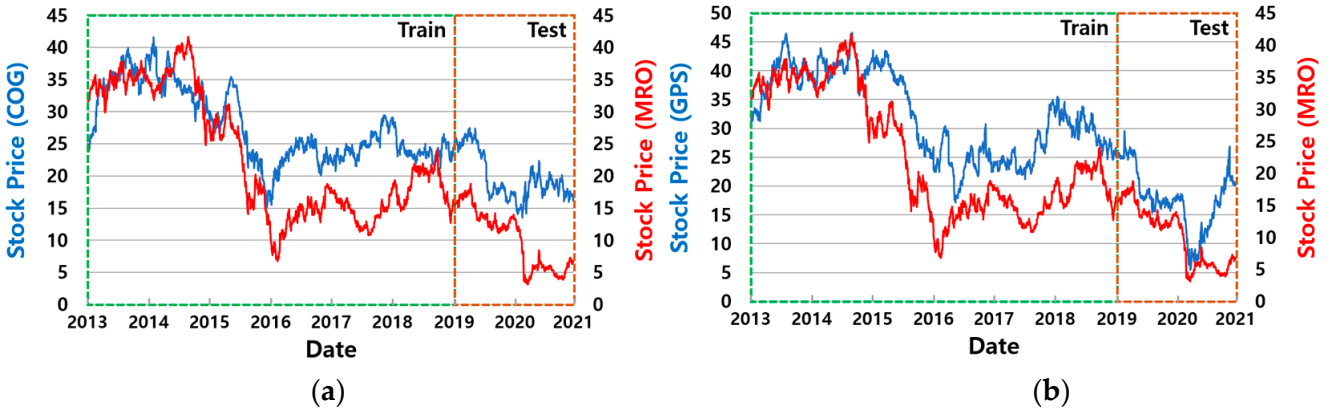

Figure 11. Stock pairs trending downward. (**a**) COG, MRO. (**b**) GPS, MRO.

5.1.2. Evaluation Metrics

We evaluate the methods in terms of the rate of return and risk indicators, such as the Sharpe ratio (SR) [53] and maximum drawdown (MDD) [2]. The rate of return is computed by Equation (29), where PV_{start} is the initial money of \$10,000, and PV_{end} is the portfolio value at the end of the test period, which is the sum of the values of long and short positions and the remaining money.

$$(PV_{end} - PV_{start})/PV_{start} \tag{29}$$

The SR in Equation (30) measures the return of an investment compared to its risk. In Equation (30), $\mathbb{E}[R]$ is the expected return, and $\sigma[R]$ is the standard deviation of the return, which measures fluctuation (i.e., risk). A higher SR indicates a higher risk-adjusted return. We use the change rate of the portfolio value at each day as the return for the day.

$$\text{SR} = \frac{\mathbb{E}[R]}{\sigma[R]} \tag{30}$$

The MDD in Equation (31) measures the maximum loss rate from the peak to the trough of a portfolio over a specified period T. In Equation (31), the inner max term computes the drawdown for time τ. A lower MDD indicates a lower risk.

$$\text{MDD}(T) = \max_{\tau \in (0,\, T)} \left[\max_{t \in (0,\, \tau)} \frac{PV_t - PV_\tau}{PV_t} \right] \tag{31}$$

5.1.3. Baseline Methods

HDRL-Trader is compared with the state-of-the-art methods listed below. For a fair comparison, we optimize the network structures and hyperparameters for each method, as described below. The Adam optimizer is used for all methods with momentum parameters β_1 of 0.9 and β_2 of 0.999, an epsilon of 10^{-7}, and a decay of 0.99.

- Buy and Hold (B&H) buys two stocks in a pair on the first day of the test period, holds them, and then sells them on the last day of the test period.
- PTDQN [1] determines trading and stop-loss boundaries using a DQN. The Q-network comprises two ReLU dense layers with 15 units and a softmax output layer. We set the sliding window size to 30, mini-batch size to 32, learning rate to 0.001, and ε to 0.5 with a decay of 0.95.
- P-DDQN [3] determines trading actions using a DDQN with a negative reward multiplier. The Q-network comprises two ReLU dense layers with 50 units and a softmax output layer. We set the mini-batch size to 32, learning rate to 0.0001, and ε to 0.3.
- P-Trader [5] determines trading actions using a DDQN and employs techniques such as clustering, a gate structure, a temporal attention mechanism, and a regression network. The Q-network comprises two ReLU dense layers with 128 and 64 units and a softmax output layer. The regression network comprises a ReLU dense layer with 128 units and a linear output layer. We set the sliding window size to 15, mini-batch size to 32, learning rate to 0.0001, and ε to 0.5 with a decay of 0.95.

5.1.4. Implementation Details of HDRL-Trader

In the data preprocessing, we set the tumbling window size for the z-score normalization to 20, the dimensionality threshold F in Figure 4 to 8, and the number of clusters to 10. In the SRL, we set the sliding window size for input to 10 and the number of units of the LSTM layer to 128. In the hybrid reinforcement learning, we set the trading window size in Equation (22) to 60; transaction cost rate ζ in Equation (23) to 0.3%; trading boundary tb for the behavior cloning to 1.0; c and b in Equation (28) for the dynamic delay to 4 and 2, respectively; α, β, and ξ for the PER to 0.6, 0.4, and 0.0001, respectively; probability ε to 0.3; noise size σ for exploration to 0.7; noise size σ' for regularization to 0.7; clipping size c to 1; mini-batch size to 32; and learning rate to 0.0001.

5.2. Experimental Results

5.2.1. Comparison with Other Methods

To compare HDRL-Trader with the other methods in detail, the smaller dataset with various price trends is used. Table 3 shows that the pairs trading methods (P-DDQN, PTDQN, P-Trader, and HDRL-Trader) achieve good profits for all price trends because pairs trading is a market-neutral trading strategy. HDRL-Trader has the highest return rate, highest SR, and lowest MDD for all price trends because it integrates all of the novel techniques in Table 2. Compared with the second-best method, P-Trader, which directly determines trading actions without a stop-loss boundary, HDRL-Trader shows significantly higher performance because of the hybrid reinforcement learning with the stop-loss network. For the downward trend, the MDDs are higher than those for the other trends for all methods because they suffer from exits with a great loss due to the large divergence, as shown in Figure 11. Even in this situation, HDRL-Trader has the lowest MDD because it reduces significant losses using hybrid reinforcement learning with the stop-loss network.

To verify the generalization ability of the methods, the larger dataset is used. As shown in Figure 12 and Table A1 in Appendix A, HDRL-Trader outperforms all other methods in terms of the minimum, maximum, and average return rate, SR, and MDD. HDRL-Trader archives an average return rate of 82.4%, which is 25.7%P higher than that of the second-best method. The average SR of HDRL-Trader is 1.24, which is 0.26 higher than the second-best method. The average MDD of HDRL-Trader is 0.36, which is 0.08 lower than the second-best method. These results indicate that the hybrid reinforcement learning

algorithm of HDRL-Trader with its novel techniques is very effective for generalization. The experimental results can be statistically analyzed using the metrics for measuring machine intelligence [54], and we leave this as future work.

Table 3. Experimental results on the smaller dataset.

Rate of Return (the Higher the Better)						
Stock Pairs		**B&H**	**P-DDQN**	**PTDQN**	**P-Trader**	**HDRL-Trader**
Upward Trending	A, PLD	76.8%	92.9%	115.3%	109.2%	**151.7%**
	ARE, DRL	46.5%	42.9%	75.6%	91.8%	**129.0%**
Sideways Trending	HIG, BK	−0.4%	27.8%	51.8%	49.6%	**79.8%**
	O, EVRG	0.8%	59.2%	65.8%	73.5%	**104.0%**
Downward Trending	COG, MRO	−42.8%	107.0%	104.5%	99.1%	**132.3%**
	GPS, MRO	−38.2%	37.8%	13.7%	37.5%	**64.0%**
Minimum		−42.8%	27.8%	13.7%	37.5%	**64.0%**
Maximum		76.8%	107.0%	115.3%	109.2%	**151.7%**
Average		7.1%	61.2%	71.1%	76.8%	**110.1%**
Sharpe Ratio (the Higher the Better)						
Stock Pairs		**B&H**	**P-DDQN**	**PTDQN**	**P-Trader**	**HDRL-Trader**
Upward Trending	A, PLD	1.57	1.88	1.82	2.06	**2.25**
	ARE, DRL	1.13	1.26	1.72	1.49	**1.74**
Sideways Trending	HIG, BK	0.25	0.82	0.98	0.86	**1.25**
	O, EVRG	0.30	0.66	0.84	1.20	**1.64**
Downward Trending	COG, MRO	−0.62	1.49	1.16	1.33	**1.61**
	GPS, MRO	−0.12	0.54	0.48	0.63	**0.93**
Minimum		−0.62	0.54	0.48	0.63	**0.93**
Maximum		1.57	1.88	1.82	2.06	**2.25**
Average		0.42	1.11	1.17	1.26	**1.57**
Maximum Drawdown (the Lower the Better)						
Stock Pairs		**B&H**	**P-DDQN**	**PTDQN**	**P-Trader**	**HDRL-Trader**
Upward Trending	A, PLD	0.45	0.38	0.40	0.27	**0.24**
	ARE, DRL	0.36	0.30	0.35	0.33	**0.22**
Sideways Trending	HIG, BK	0.43	0.25	0.39	0.21	**0.16**
	O, EVRG	0.51	0.49	0.35	0.36	**0.21**
Downward Trending	COG, MRO	0.65	0.70	0.59	0.46	**0.43**
	GPS, MRO	0.81	0.51	0.53	0.52	**0.48**
Minimum		0.36	0.25	0.35	0.21	**0.16**
Maximum		0.81	0.70	0.59	0.52	**0.48**
Average		0.53	0.44	0.43	0.36	**0.29**

5.2.2. Ablation Studies

To evaluate the contribution of each technique used in HDRL-Trader, ablation studies are conducted using the smaller dataset. The techniques excluded one-by-one are the stop-loss network (SL), PER, behavior cloning (BC), clustering (Clu), dimensionality reduction (Dim), regression network (Reg), and gate structure (Gat). Figure 13 shows the results, where "All" denotes HDRL-Trader with all techniques. All the techniques contribute significantly to the performance improvement. In particular, the hybrid with the stop-

loss network is very crucial because it reduces significant losses. For the dynamic delay technique, we compare its performance with those of static delays ranging from two to five. Figure 14 shows that the dynamic delay outperforms all the static delays, demonstrating its contribution.

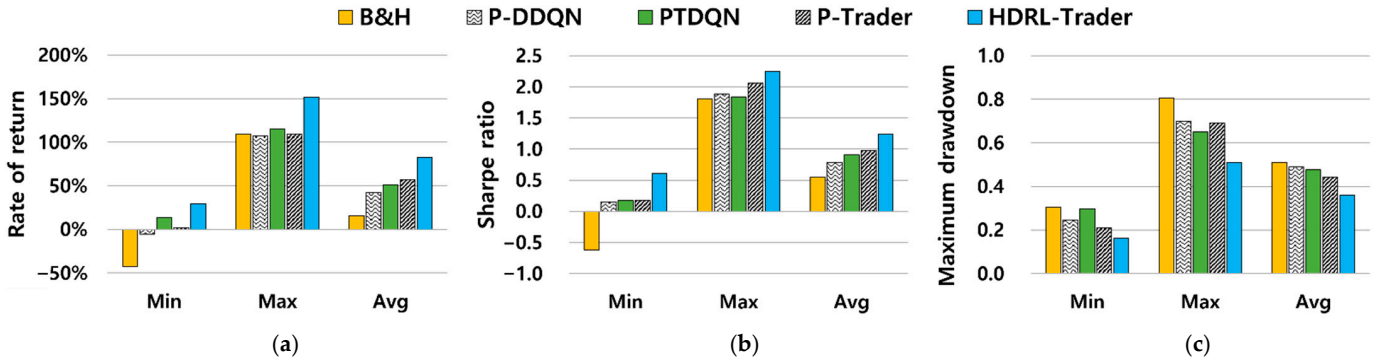

Figure 12. Experimental results on the larger dataset. (**a**) Rate of return. (**b**) Sharpe ratio. (**c**) Maximum drawdown.

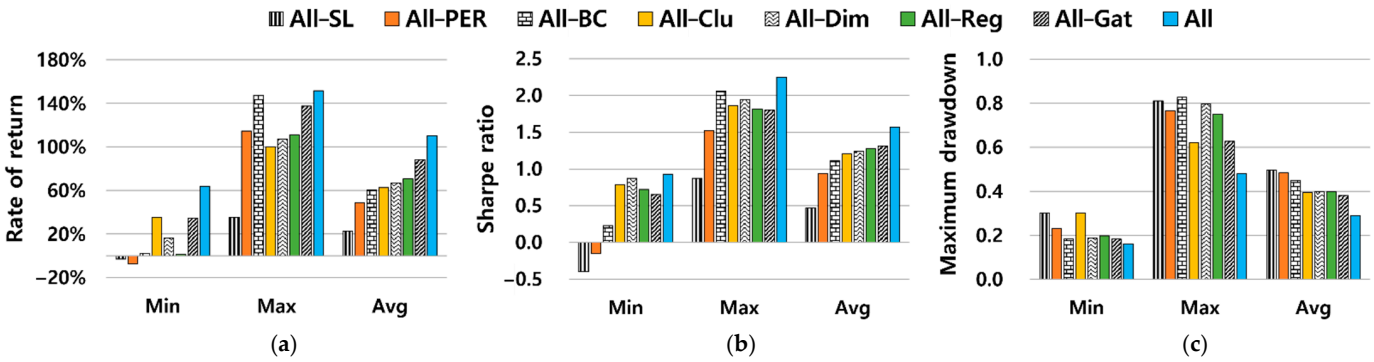

Figure 13. Experimental results of the ablation studies. (**a**) Rate of return. (**b**) Sharpe ratio. (**c**) Maximum drawdown.

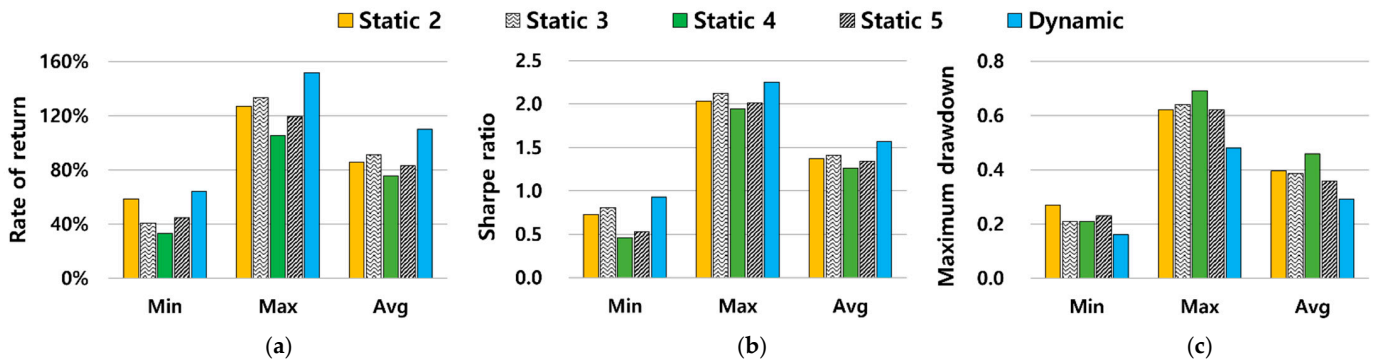

Figure 14. Comparison of static and dynamic delays. (**a**) Rate of return. (**b**) Sharpe ratio. (**c**) Maximum drawdown.

5.2.3. Comparison with Other Hyperparameter Values

Figure 15 shows the effect of the major hyperparameters used in HDRL-Trader, i.e., the dimensionality reduction threshold, number of clusters, sliding window size, and noise size for exploration. The average return rate is evaluated on the smaller dataset. As shown in Figure 15a–c, the performance is degraded if the dimensionality threshold, number of clusters, or sliding window size is set too small or too large. This is because there is

information underload (overload) if they are set too small (large). Figure 15d shows that if the noise size for exploration is set too small or too large, the performance is degraded because of the exploration-exploitation trade-off.

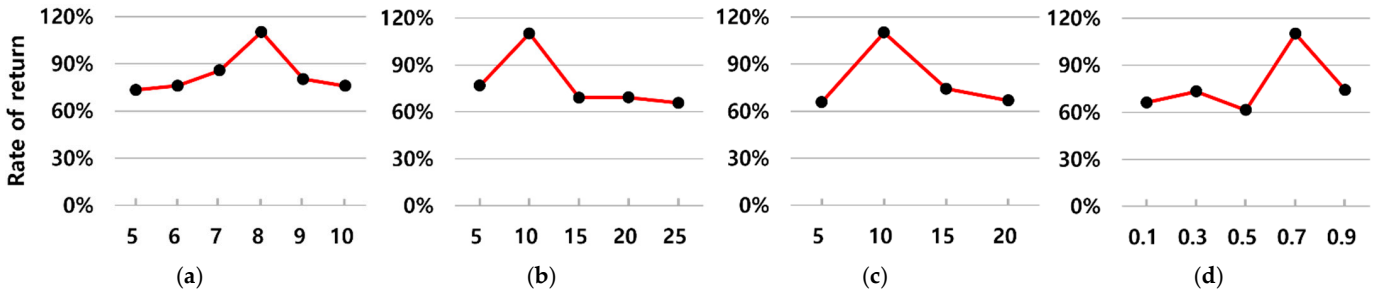

Figure 15. Comparison with other hyperparameter values. (**a**) Dimensionality. (**b**) Number of Clusters. (**c**) Sliding Window Size. (**d**) Noise Size.

5.2.4. Robustness Study

Figure 16 shows the effect of the transaction cost rate ζ, which demonstrates the robustness of HDRL-Trader. The average return rate is evaluated on the smaller dataset. The results show that the profit decreases as the transaction cost rate increases for all methods. However, HDRL-Trader achieves the best profit compared with the other methods regardless of the transaction cost rate. Furthermore, HDRL-Trader achieves a good profit even for a transaction cost rate of 0.5%, which is much higher than that of a real trading environment.

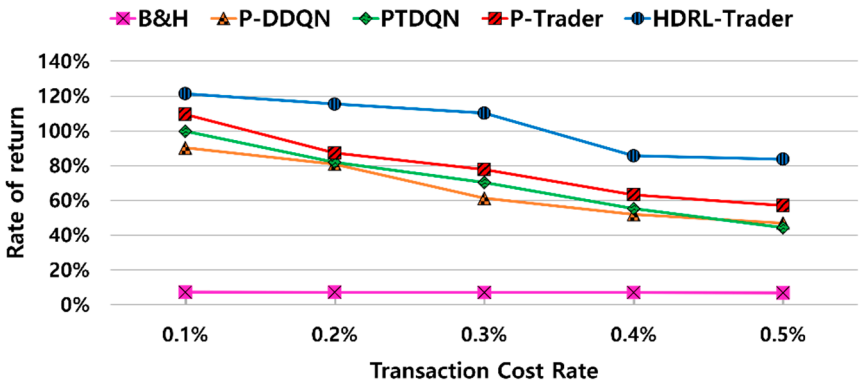

Figure 16. Experimental results of robustness study.

6. Conclusions and Future Work

In this study, we proposed a novel hybrid reinforcement learning method for pairs trading called HDRL-Trader that solves the dependency problem between the stop-loss and trading boundaries and the problem of the absence of the stop-loss boundary. We extended the twin delayed deep deterministic policy gradient algorithm to determine the trading action and the double deep Q-network algorithm to determine the stop-loss boundary. We then proposed a hybrid algorithm that combines the extended algorithms and incorporated novel techniques, such as dimensionality reduction, clustering, regression, behavior cloning, prioritized experience replay, and dynamic delay, into the hybrid algorithm. We compared the performance of HDRL-Trader with the state-of-the-art reinforcement learning methods for pairs trading (P-DDQN, PTDQN, and P-Trader). The experimental results for the twenty stock pairs showed that HDRL-Trader achieves an average return rate of 82.4%, which is 25.7%P higher than that of the second-best method, and yielded significantly positive return rates for all the stock pairs. Future work can overcome limitations of the present study. First, we plan to extend HDRL-Trader for continuous action spaces. Second, we

plan to investigate the effect of various methods of selecting stock pairs. Last, we plan to statistically analyze the experimental results using the intelligence metrics.

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

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1D1A1B07043727). The work reported in this paper was conducted during the sabbatical year of Kwangwoon University in 2019.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Stock data used in this study are available at https://finance.yahoo.com/ (accessed on 22 November 2021).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Table A1. Experimental results on the larger dataset.

Stock Pairs	Rate of Return					Sharpe Ratio					Maximum Drawdown				
	B&H	P-DDQN	PT-DQN	P-Trader	HDRL-Trader	B&H	P-DDQN	PT-DQN	P-Trader	HDRL-Trader	B&H	P-DDQN	PT-DQN	P-Trader	HDRL-Trader
AAPL, TXT	109.3%	88.8%	109.6%	99.8%	**125.2%**	1.81	1.51	1.74	1.72	**2.12**	0.58	0.63	0.57	0.54	**0.47**
AOS, CCL	−15.4%	51.6%	13.4%	50.0%	**68.4%**	−0.04	0.37	0.17	0.54	**0.86**	0.59	**0.49**	0.64	0.69	0.51
FE, RE	53.8%	20.7%	43.3%	57.9%	**65.3%**	**1.20**	0.55	1.07	0.98	1.05	0.41	0.42	0.36	0.39	**0.27**
MMM, RE	0.3%	11.3%	23.1%	10.6%	**29.7%**	0.22	0.48	0.64	0.54	**0.66**	0.37	0.47	0.43	0.45	**0.33**
NIKE, FTNT	100.7%	43.6%	66.9%	70.9%	**117.2%**	**1.72**	0.57	1.02	1.42	1.69	0.52	0.58	0.55	0.59	**0.51**
SRE, MDT	25.6%	21.8%	40.4%	60.3%	**76.4%**	0.76	0.99	1.17	1.28	**1.37**	0.40	0.36	0.43	0.37	**0.28**
APA, HES	−11.5%	54.8%	48.7%	73.5%	**98.1%**	0.37	0.77	0.84	0.99	**1.13**	0.71	0.63	0.65	0.52	**0.49**
OXY, HES	−23.5%	57.3%	29.3%	44.3%	**60.1%**	0.16	0.95	0.65	0.81	**1.06**	0.68	0.51	0.47	0.55	**0.46**
CMA, ADI	25.7%	11.1%	16.8%	23.3%	**51.4%**	0.69	0.55	0.56	0.64	**0.94**	0.46	0.61	0.56	0.45	**0.34**
LHX, MTB	14.4%	33.1%	21.0%	40.0%	**53.2%**	0.53	0.63	0.54	0.65	**0.84**	0.40	0.33	0.30	0.32	**0.21**
PEP, ATO	20.3%	1.1%	15.4%	37.4%	**51.8%**	0.68	0.32	0.43	0.75	**0.86**	0.30	0.40	0.48	0.36	**0.30**
DXC, ALL	−9.1%	−5.5%	28.3%	2.0%	**29.0%**	0.09	0.15	0.49	0.17	**0.61**	0.54	0.60	0.53	0.52	**0.47**
VLO, NTRS	−7.4%	31.7%	65.9%	59.5%	**70.3%**	0.19	0.58	0.78	0.83	**1.06**	0.55	0.54	0.49	0.53	**0.48**
MPC, CNC	−12.8%	56.9%	66.1%	44.2%	**91.2%**	0.07	0.72	0.82	0.64	**1.10**	0.52	0.58	0.47	0.43	**0.34**
A, PLD	76.8%	92.9%	115.3%	109.2%	**151.7%**	1.57	1.88	1.82	2.06	**2.25**	0.45	0.38	0.40	0.27	**0.24**
ARE, DLR	46.5%	42.9%	75.6%	91.8%	**129.0%**	1.13	1.26	1.72	1.49	**1.74**	0.36	0.30	0.35	0.33	**0.22**
O, EVRG	−0.4%	27.8%	51.8%	49.6%	**79.8%**	0.25	0.82	0.98	0.86	**1.25**	0.43	0.25	0.39	0.21	**0.16**
HIG, BK	0.8%	59.2%	65.8%	73.5%	**104.0%**	0.30	0.66	0.84	1.20	**1.64**	0.51	0.49	0.35	0.36	**0.21**
COG, MRO	−42.8%	107.0%	104.5%	99.1%	**132.3%**	−0.62	1.49	1.16	1.33	**1.61**	0.65	0.70	0.59	0.46	**0.43**
GPS, MRO	−38.2%	37.8%	13.7%	37.5%	**64.0%**	−0.12	0.54	0.48	0.63	**0.93**	0.81	0.51	0.53	0.52	**0.48**
Minimum	−42.8%	−5.5%	13.4%	2.0%	**29.0%**	−0.62	0.15	0.17	0.17	**0.61**	0.30	0.25	0.30	0.21	**0.16**
Maximum	109.3%	107.0%	115.3%	109.2%	**151.7%**	1.81	1.88	1.82	2.06	**2.25**	0.81	0.70	0.65	0.69	**0.51**
Average	15.6%	42.3%	50.7%	56.7%	**82.4%**	0.55	0.79	0.90	0.98	**1.24**	0.51	0.49	0.48	0.44	**0.36**

Appendix B

Table A2. List of abbreviations.

Abbreviation	Description
B&H	Buy and Hold
CE	Cross Entropy
DPG	Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
DDQN	Double Deep Q-Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
MSE	Mean Squared Error
SRL	State Representation Learning
HDRL	Hybrid Deep Reinforcement Learning
TD3	Twin-Delayed Deep Deterministic policy gradient
TW	Trading Window

References

1. Kim, T.; Kim, H.Y. Optimizing the pairs-trading strategy using deep reinforcement learning with trading and stop-loss boundaries. *Complexity* **2019**, *2019*, 1–20. [CrossRef]
2. Lu, J.Y.; Lai, H.C.; Shih, W.Y.; Chen, Y.F.; Huang, S.H.; Chang, H.H.; Wang, J.Z.; Huang, J.L.; Dai, T.S. Structural break-aware pairs trading strategy using deep reinforcement learning. *J. Supercomput.* **2021**, 1–40. [CrossRef]
3. Brim, A. Deep reinforcement learning pairs trading with a double deep Q-network. In Proceedings of the 2020 10th Annual Computing and Communication Workshop and Conference, CCWC, Las Vegas, NV, USA, 6–8 January 2020; pp. 222–227.
4. Wang, C.; Sandås, P.; Beling, P. Improving pairs trading strategies via reinforcement learning. In Proceedings of the 2021 International Conference on Applied Artificial Intelligence, ICAPAI, Halden, Norway, 19–21 May 2021; pp. 1–7.
5. Kim, S.H.; Park, D.Y.; Lee, K.H. A practical pairs-trading method using deep reinforcement learning. *Database Res.* **2021**, *37*, 65–80.
6. Fujimoto, S.; Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. In Proceedings of the 35th International Conference on Machine Learning, ICML, Stockholm, Sweden, 10–15 July 2018; Volume 80, pp. 1582–1591.
7. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double Q-learning. In Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI, Phoenix, AZ, USA, 12–17 February 2016; Volume 30, pp. 2094–2100.
8. Dickey, D.A.; Fuller, W.A. Distribution of the estimators for autoregressive time series with a unit root. *J. Am. Stat. Assoc.* **1979**, *74*, 427–431.
9. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
10. Kendall, E.A.; Malkoun, M.T.; Jiang, C.H. A methodology for developing agent based systems for enterprise integration. In *Modelling and Methodologies for Enterprise Integration*; Bernus, P., Nemes, L., Eds.; Springer: Boston, MA, USA, 1996; pp. 333–344.
11. Slušný, S.; Neruda, R.; Vidnerová, P. Comparison of RBF network learning and reinforcement learning on the maze exploration problem. In Proceedings of the 18th International Conference on Artificial Neural Networks, ICANN, Prague, Czech Republic, 3–6 September 2008; pp. 720–729.
12. Wang, B.N.; Gao, Y.; Chen, J.Y.; Chen, S.F. A two-layered multi-agent reinforcement learning model and algorithm. *J. Netw. Comput. Appl.* **2017**, *30*, 1366–1376. [CrossRef]
13. Gershman, S.J.; Pesaran, B.; Daw, N.D. Human reinforcement learning subdivides structured action spaces by learning effector-specific values. *J. Neurosci.* **2009**, *29*, 13524–13531. [CrossRef] [PubMed]
14. Kendall, E.A.; Malkoun, M.T.; Jiang, C.H. The application of object-oriented analysis to agent based systems. *J. Occup. Organ. Psychol.* **1997**, *9*, 56–62.
15. Kaelbling, L.P.; Littman, M.L.; Cassandra, A.R. Planning and acting in partially observable stochastic domains. *Artif. Intell.* **1998**, *101*, 99–134. [CrossRef]
16. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.A.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
17. Bellman, R. On the theory of dynamic programming. *Proc. Natl. Acad. Sci. USA* **1952**, *38*, 716–719. [CrossRef]

18. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. In Proceedings of the 4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico, 2–4 May 2016.
19. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, LA, USA, 2–7 February 2018; pp. 3215–3222.
20. Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; Freitas, N. Dueling network architectures for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning, ICML, New York, NY, USA, 19–24 June 2016; Volume 48, pp. 1995–2003.
21. Sutton, R.S.; Barto, A.G. *Introduction to Reinforcement Learning*; MIT Press: Cambridge, MA, USA, 1998; Volume 135.
22. Bellemare, M.G.; Dabney, W.; Munos, R. A distributional perspective on reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning, ICML, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 449–458.
23. Fortunato, M.; Azar, M.G.; Piot, B.; Menick, J.; Hessel, M.; Osband, I.; Graves, A.; Mnih, V.; Munos, R.; Hassabis, D.; et al. Noisy networks for exploration. In Proceedings of the 6th International Conference on Learning Representations, ICLR, Vancouver, BC, Canada, 30 April–3 May 2018.
24. Sutton, R.S.; McAllester, D.A.; Singh, S.P.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Proceedings of the Advanced in Neural Information Processing Systems, NIPS, Denver, CO, USA, 29 November–4 December 1999; pp. 1057–1063.
25. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the 31st International Conference on Machine Learning, ICML, Beijing, China, 21–26 June 2014; Volume 32, pp. 387–395.
26. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. In Proceedings of the 4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico, 2–4 May 2016.
27. Ding, X.; Zhang, Y.; Liu, T.; Duan, J. Deep learning for event-driven stock prediction. In Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI, Buenos Aires, Argentina, 25–31 July 2015; pp. 2327–2333.
28. Tsantekidis, A.; Passalis, N.; Tefas, A.; Kanniainen, J.; Gabbouj, M.; Iosifidis, A. Forecasting stock prices from the limit order book using convolutional neural networks. In Proceedings of the 2017 IEEE 19th Conference on Business Informatics (CBI), Thessaloniki, Greece, 24–27 July 2017; Volume 1, pp. 7–12.
29. Chong, E.; Han, C.; Park, F.C. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Syst. Appl.* **2017**, *83*, 187–205. [CrossRef]
30. Zhang, L.; Aggarwal, C.; Qi, G.J. Stock price prediction via discovering multi-frequency trading patterns. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD, New York, NY, USA, 13–17 August 2017; pp. 2141–2149.
31. Tran, D.T.; Iosifidis, A.; Kanniainen, J.; Gabbouj, M. Temporal attention-augmented bilinear network for financial time-series data analysis. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *30*, 1407–1418. [CrossRef] [PubMed]
32. Feng, F.; He, X.; Wang, X.; Luo, C.; Liu, Y.; Chua, T.S. Temporal relational ranking for stock prediction. *ACM Trans. Inf. Syst.* **2019**, *37*, 1–30. [CrossRef]
33. Fengqian, D.; Chao, L. An adaptive financial trading system using deep reinforcement learning with candlestick decomposing features. *IEEE Access* **2020**, *8*, 63666–63678. [CrossRef]
34. Lei, K.; Zhang, B.; Li, Y.; Yang, M.; Shen, Y. Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading. *Expert Syst. Appl.* **2020**, *140*, 112872. [CrossRef]
35. Liu, Y.; Liu, Q.; Zhao, H.; Pan, Z.; Liu, C. Adaptive quantitative trading: An imitative deep reinforcement learning approach. In Proceedings of the 34th AAAI Conference on Artificial Intelligence, AAAI, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 2128–2135.
36. Park, D.Y.; Lee, K.H. Practical algorithmic trading using state representation learning and imitative reinforcement learning. *IEEE Access* **2021**, *9*, 152310–152321. [CrossRef]
37. Cho, K.; van Merrienboer, B.; Gülçehre, Ç.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, Doha, Qatar, 25–29 October 2014; pp. 1724–1734.
38. Li, T.; Zhao, Z.; Sun, C.; Cheng, L.; Chen, X.; Yan, R.; Gao, R.X. Waveletkernelnet: An interpretable deep neural network for industrial intelligent diagnosis. In *IEEE Transactions on Systems, Man, and Cybernetics: Systems*; IEEE: Piscataway, NJ, USA, 2021; pp. 1–11.
39. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
40. Engle, R.F.; Granger, C.W.J. Co-integration and error correction: Representation, estimation, and testing. *Econometrica* **1987**, *55*, 251–276. [CrossRef]
41. Liang, S.; Lu, S.; Lin, J.; Wang, Z. Low-latency hardware accelerator for improved Engle-Granger cointegration in pairs trading. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2021**, *68*, 2911–2924. [CrossRef]
42. Krauss, C. Statistical arbitrage pairs trading strategies: Review and outlook. *J. Econ. Surv.* **2016**, *31*, 513–545. [CrossRef]
43. Brunetti, M.; Luca, R.D. *Pre-Selection in Cointegration-Based Pairs Trading*; Vergata Press: Italy, Rome, 2021.

44. Miao, G.J. High frequency and dynamic pairs trading based on statistical arbitrage using a two-stage correlation and cointegration approach. *Int. J. Econ. Financ. Issues* **2014**, *6*, 96–110. [CrossRef]
45. Chen, H.; Chen, S.; Chen, Z.; Li, F. Empirical investigation of an equity pairs trading strategy. *Manag. Sci.* **2017**, *65*, 370–389. [CrossRef]
46. Erdem, O.; Ceyhan, E.; Varli, Y. A new correlation coefficient for bivariate time-series data. *Phys. A Stat. Mech. Appl.* **2014**, *414*, 274–284. [CrossRef]
47. Bezdek, J.C.; Ehrlich, R.; Full, W. FCM: The fuzzy c-means clustering algorithm. *Comput. Geosci.* **1984**, *10*, 191–203. [CrossRef]
48. TA-Lib: Technical Analysis Library. Available online: http://ta-lib.org/ (accessed on 22 November 2021).
49. Li, W.; Liao, J. A comparative study on trend forecasting approach for stock price time series. In Proceedings of the 2017 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification, ASID, Xiamen, China, 27–29 October 2017; pp. 74–78.
50. Nabipour, M.; Nayyeri, P.; Jabani, H.; Mosavi, A.; Salwana, E.; Shahab, S. Deep learning for stock market prediction. *Entropy* **2020**, *22*, 840. [CrossRef] [PubMed]
51. Banik, S.; Sharma, N.; Mangla, M.; Mohanty, S.N.; Shitharth, S. LSTM based decision support system for swing trading in stock market. *Knowl.-Based Syst.* **2021**, *239*, 107994. [CrossRef]
52. Yahoo Finance. Available online: https://finance.yahoo.com/ (accessed on 22 November 2021).
53. Sharpe, W.F. The Sharpe ratio. *J. Portf. Manag.* **1994**, *21*, 49–58. [CrossRef]
54. Iantivics, L.B.; Iakovidis, D.K.; Nechita, E. II-Learn-A novel metric for measuring the intelligence increase and evolution of artificial learning systems. *Int. J. Comput. Intell. Syst.* **2019**, *12*, 1323–1338. [CrossRef]