IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 18, 2021, accepted October 25, 2021, date of publication November 10, 2021,
date of current version November 18, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3127209

Practical Algorithmic Trading Using State Representation Learning and Imitative Reinforcement Learning

DEOG-YEONG PARK AND KI-HOON LEE
School of Computer and Information Engineering, Kwangwoon University, Nowon-gu, Seoul 01897, Republic of Korea

Corresponding author: Ki-Hoon Lee (kihoonlee@kw.ac.kr)

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 present research has been conducted by the Research Grant of Kwangwoon University in 2020.

ABSTRACT Algorithmic trading allows investors to avoid emotional and irrational trading decisions and helps them make profits using modern computer technology. In recent years, reinforcement learning has yielded promising results for algorithmic trading. Two prominent challenges in algorithmic trading with reinforcement learning are (1) extracting robust features and (2) learning a profitable trading policy. Another challenge is that it was previously often assumed that both long and short positions are always possible in stock trading; however, taking a short position is risky or sometimes impossible in practice. We propose a practical algorithmic trading method, *SIRL-Trader*, which achieves good profit using only long positions. SIRL-Trader uses offline/online state representation learning (SRL) and imitative reinforcement learning. In offline SRL, we apply dimensionality reduction and clustering to extract robust features whereas, in online SRL, we co-train a regression model with a reinforcement learning model to provide accurate state information for decision-making. In imitative reinforcement learning, we incorporate a behavior cloning technique with the twin-delayed deep deterministic policy gradient (TD3) algorithm and apply multistep learning and dynamic delay to TD3. The experimental results show that SIRL-Trader yields higher profits and offers superior generalization ability compared with state-of-the-art methods.

INDEX TERMS Algorithmic trading, deep learning, state representation learning, imitation learning, reinforcement learning.

I. INTRODUCTION

Algorithmic trading, which enables investors to trade stocks without human intervention, has started playing an important role in modern stock markets. Algorithmic trading is a subset of quantitative trading, which relies heavily on quantitative analysis and machine-learning methods. In particular, reinforcement learning methods can be used to learn trading strategies in the process of interacting with the stock market environment. The application of reinforcement learning to algorithmic trading is not trivial because financial time-series data are nonstationary and contain considerable noise. Two prominent challenges presented by algorithmic trading with reinforcement learning are (1) extracting robust features and (2) learning a profitable trading policy. To accommodate these challenges, recent methods [1]–[6] use deep reinforcement learning and deliver good performance in terms of profitability.

Previously, an assumption was often made that both long and short positions are always possible in stock trading. These positions represent the direction of the bets that a stock price is expected to either rise or fall. A long position involves buying stocks with the intention of future selling, whereas a short position involves selling stocks borrowed from a stockbroker first and then buying them back to close the position.

We note that taking a short position is risky or sometimes impossible in practice, particularly for individual investors. First, the maximum gain in a short position is 100% when the stock price falls to zero, whereas the potential loss is theoretically infinite as the price has no upper bound. On the

The associate editor coordinating the review of this manuscript and approving it for publication was Khoa Luu.

other hand, the maximum loss in a long position is limited because the price can only decrease to zero, but the possible gain is theoretically infinite. Second, the stockbroker may close a short position immediately, without the investor's consent. The borrowed stocks are essentially loans from stockbrokers. If the stock price rises, the stockbroker requires additional capital, and if the investor cannot provide capital, the stockbroker can close the position, resulting in a loss. Third, the stock market generally increases over time. Fourth, economic regulators can severely restrict or temporarily ban short-selling during an economic crisis. In summary, a short position is not recommended for individual investors. However, obtaining good profits using only long positions is challenging in algorithmic trading.

In this paper, we propose a practical algorithmic trading method named *SIRL-Trader*, which generates good profits using only long positions. First, we devise an offline/online state representation learning (SRL) method. The offline unsupervised SRL reduces the dimensionality and applies clustering to extract robust features from observations. The online supervised SRL co-trains a regression model that predicts the next price with a reinforcement learning model to provide accurate state information for decision-making. Second, we combine imitation learning with reinforcement learning by cloning the behavior of a prophetic expert who has information about subsequent price movements. Third, we extend the twin-delayed deep deterministic policy gradient (TD3) algorithm [7], which is a state-of-the-art reinforcement learning method, to incorporate offline/online SRL, behavior cloning, multistep learning, and dynamic delay. Fourth, compared with state-of-the-art algorithmic trading methods, SIRL-Trader yields higher profit and has superior generalization ability for different stocks.

The remainder of this paper is organized as follows. Section II introduces reinforcement learning methods, and Section III reviews existing work. Sections IV and V present SIRL-Trader and experimental results, respectively. Finally, Section VI presents our conclusions and suggestions for future work. For ease of reading, Table 5 in the Appendix lists the abbreviations used in this paper.

II. BACKGROUND

A. REINFORCEMENT LEARNING

In reinforcement learning, an agent learns to act in an environment to maximize the total reward. At each time step, the environment provides a state s to the agent, the agent selects and takes an action a, and then the environment provides a reward r and the next state s'. This interaction can be formalized as a Markov Decision Process (MDP), which is a tuple $\langle S, A, P, R, \gamma \rangle$, where S is a finite set of states, A is a finite set of actions, $P(s, a, s')$ is a state transition probability, $R(s, a)$ is a reward function, and $\gamma \in [0, 1]$ is the discount factor, a trade-off between immediate and long-term rewards. In reinforcement learning for algorithmic trading, the state is not directly given and needs to be constructed from a history of observations. To accommodate this, the MDP model was extended with an observation probability $P(o|s, a)$. The extended model is referred to as the partially observable MDP (POMDP) model [8].

The agent selects an action using a deterministic policy $\mu(s)$ or stochastic policy $\pi(a|s)$ that defines the probability distribution over actions for each state. The discounted sum of future rewards collected by the agent from the state s_t is defined as the *discounted return* G_t in Equation (1).

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

B. DEEP Q-NETWORK (DQN)

In value-based reinforcement learning, the agent learns an estimate of the expected discounted return, or *value*, for each state (Equation (2)) or for each state and action pair (Equation (3)).

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}[G_t|S_t = s] \tag{2}$$

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

A common way of deriving a new policy π' from $Q^{\pi}(s, a)$ is to act ϵ-greedily with respect to actions. With probability $(1 - \epsilon)$, the agent takes the action with the highest Q-value (the greedy action), that is, $\pi'(s) = \underset{a\in A}{\text{argmax}}\, Q^{\pi}(s, a)$. With probability ϵ, the agent takes a random action to introduce the exploration.

The deep Q-network (DQN) [9] introduces deep neural networks to approximate the Q-value for large state and action spaces. DQN uses a replay buffer to store past experiences as tuples of $\langle s, a, r, s' \rangle$ and learns by sampling batches from the replay buffer. DQN uses two neural networks: online (Q_θ) and target ($Q_{\theta'}$) networks. The parameters θ of the online network are periodically copied to the target network during training. The loss function of DQN in Equation (5) is the mean squared error (MSE) between $Q_\theta(s, a)$ and the target value Y_{DQN} in Equation (4), which uses the Bellman equation [10]. The techniques of experience replay and the target network enable stable learning.

$$Y_{DQN} = r + \gamma \max_{a'} Q_{\theta'}(s', a') \tag{4}$$

$$L_{DQN} = \mathbb{E}[(Y_{DQN} - Q_\theta(s, a))^2] \tag{5}$$

Because we take the maximum value for the target network in Equation (4), we often obtain an overestimated value. Double DQN (DDQN) [11] solves this overestimation problem of DQN by decoupling the selection of the action from its evaluation, as shown in Equation (6).

$$Y_{DDQN} = r + \gamma Q_{\theta'}(s', \underset{a'}{\text{argmax}}\, Q_\theta(s', a')) \tag{6}$$

Rainbow DQN [12] is an improvement on DQN and combines several features including DDQN, prioritized experience replay [13], dueling networks [14], multistep learning [15], distributional reinforcement learning [16], and noisy networks [17].

C. ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC (A3C)

In policy-based reinforcement learning, the agent directly learns a policy function $\pi(a|s)$. Actor-critic methods combine policy-based and value-based reinforcement learning methods. In these methods, the critic network learns the value function, and the actor network learns the policy in the direction suggested by the critic. In general, the loss function of the critic network (V_θ) is defined as in Equation (8) using the target Y in Equation (7), which uses the Bellman equation; the loss function of the actor network (π_ϕ) is defined in Equation (9), which uses the stochastic policy gradient theorem [18].

$$Y = r + \gamma V_\theta(s') \tag{7}$$
$$L_{critic} = \mathbb{E}[(Y - V_\theta(s))^2] \tag{8}$$
$$L_{actor} = \mathbb{E}[-\log \pi_\phi(a|s)(Y - V_\theta(s))] \tag{9}$$

Asynchronous advantage actor-critic (A3C) is an actor-critic method that asynchronously executes multiple agents in parallel instead of using experience replay. In A3C, the critic network estimates the *advantage* of action a in state s, or $A(s, a) = Q(s, a) - V(s)$. A3C uses n-step returns to accelerate convergence and includes the entropy of the policy $\pi(a|s)$ to the loss function of the actor network to introduce the exploration.

D. DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

The deterministic policy gradient (DPG) [19] is an actor-critic method that learns a deterministic policy $\mu(s)$ rather than a stochastic policy $\pi(a|s)$. It is a special case of the stochastic policy gradient when the variance approaches zero. DPG is efficient and effective for high-dimensional continuous action spaces.

The deep DPG (DDPG) [20] is an actor-critic method that combines DPG and DQN. As in DQN, DDPG uses a replay buffer and target networks. For the exploration, DDPG adds noise $\mathcal{N}$ to the policy, as shown in Equation (10). The loss function of the critic network is defined as in Equation (12) using the target Y in Equation (11), which uses the Bellman equation, and that of the actor network in Equation (13), which uses the deterministic policy gradient theorem [19]. After updating the online actor and critic networks, the target actor and critic networks are soft-updated from the online networks, as in Equation (14).

$$a = \mu_\phi(s) + \mathcal{N} \tag{10}$$
$$Y = r + \gamma Q_{\theta'}(s', \mu_{\phi'}(s')) \tag{11}$$
$$L_{critic} = \mathbb{E}[(Y - Q_\theta(s, a))^2] \tag{12}$$
$$L_{actor} = \mathbb{E}[-Q_\theta(s, \mu_\phi(s))] \tag{13}$$
$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta', \quad \phi' \leftarrow \tau\phi + (1 - \tau)\phi' \tag{14}$$

E. TWIN-DELAYED DDPG (TD3)

The twin-delayed DDPG (TD3) improves DDPG in three ways. First, TD3 adds Gaussian noise to the target action, as in Equation (15), which is used to obtain the target value Y in Equation (16). This technique, known as *target policy smoothing*, serves as a regularizer to avoid overfitting to sharp peaks in the Q-value estimate. The noise is clipped to limit its impact.

$$a' = \mu_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}, -c, c) \tag{15}$$

Second, TD3 uses two critics (and two target critics) to solve the overestimation problem of DDPG. The minimum value in the pair of target critics is used to compute the target value, as shown in Equation (16). This technique is known as *clipped double Q-learning*.

$$Y = r + \gamma \min_{j=1,2} Q_{\theta'_j}(s', a') \tag{16}$$

Third, TD3 updates the actor network μ_ϕ (and the target actor $\mu_{\phi'}$ and critic networks $Q_{\theta'_j}$) less frequently than critic networks Q_{θ_j}. This technique, known as *delayed policy updates*, aims to delay policy updates until the Q-value converges.

III. RELATED WORK

Considerable research has been devoted to algorithmic trading, including supervised learning methods [21]–[27] that predict price trends and reinforcement learning methods [1]–[6] that directly learn a profitable trading policy. In recent years, deep reinforcement learning methods have shown promising results in algorithmic trading. There are two challenges in deep reinforcement learning for algorithmic trading: extracting robust features from observations to represent the states (i.e., SRL) and learning a profitable trading policy.

Deng *et al.* [1] used a fuzzy deep neural network for SRL and proposed a policy-based reinforcement learning method with a recurrent neural network. Li *et al.* [2] used a stacked denoising autoencoder for SRL and proposed DDQN-extended and A3C-extended methods with a long short-term memory (LSTM) network [28]. The A3C-extended method yields more profits than the DDQN-extended method. Fengqian and Chao [3] decomposed candlesticks (or K-lines) into components such as the lengths of the upper shadow line, lower shadow line, and body. Each component is then clustered, and the cluster centers and the color of the body are used to represent the state. For deep reinforcement learning, [3] used a policy gradient method with ϵ-greedy exploration. Wu *et al.* [6] used a gated recurrent unit (GRU) network [29] for SRL and proposed DDQN-extended and DDPG-extended methods, GDQN and GDPG, respectively. GDPG provides more stable returns than GDQN does. Lei *et al.* [4] used a gate structure to select features, GRU to capture long-term dependency, and a temporal attention mechanism to weight past states based on the current state. They proposed a policy gradient method, known as time-driven feature-aware jointly deep reinforcement learning (TFJ-DRL), which combines SRL and a policy gradient method using an auto-encoder. The decoding part of the SRL model is used to predict the next closing price, where

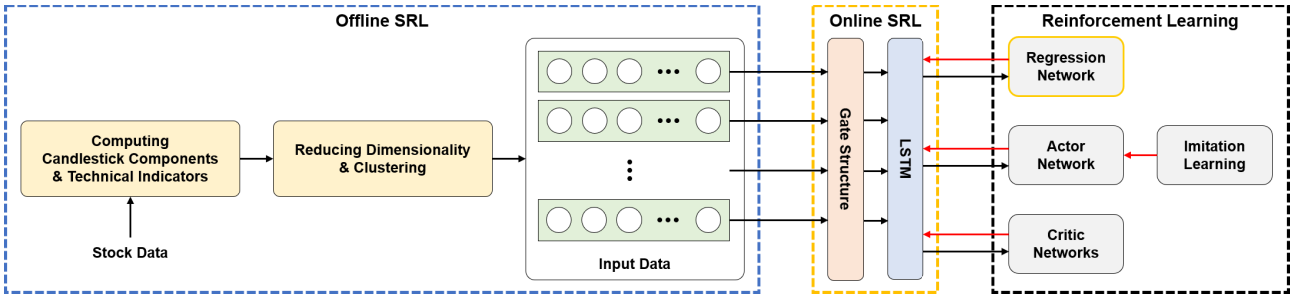

FIGURE 1. Architecture for SIRL-Trader.

the real price is used as the feedback signal. The encoding part of the SRL model is used as the state representation for the reinforcement learning. Liu *et al.* [5] used GRU for SRL and introduced imitation learning techniques, such as a demonstration buffer and behavior cloning, to the DPG algorithm.

The above-mentioned studies have resulted in several significant improvements in algorithm trading. However, it is unclear whether these improvements are complementary and can be combined to obtain positive results. This study resulted in a comprehensive solution that integrates existing improvements with new ideas, as explained in the next section.

IV. SIRL-TRADER

In this section, we propose the novel algorithmic trading method named SIRL-Trader.

A. ARCHITECTURE

We propose an actor-critic reinforcement learning method that extends TD3 to incorporate offline/online SRL, imitation learning, multistep learning, and dynamic delay. Fig. 1 shows the proposed architecture. Each component is explained in detail in the following subsections.

B. STATE REPRESENTATION LEARNING

To help the agent learn a good policy, SRL models learn how to map observations to states. We use the candlestick components and technical indicators in Table 1 as observations. Our SRL method consists of (1) offline unsupervised SRL that occurs before training the reinforcement learning model and (2) online supervised SRL that occurs while the reinforcement learning model is being trained.

The offline SRL extracts a low-dimensional robust representation from high-dimensional observations, as shown in Fig. 2; first, we normalize each input feature using the z-score standardization method. Second, for each feature group in Table 1 with high dimensionality, we reduce the dimensionality of the feature space to the threshold F in Fig. 2 using principal components analysis (PCA). Third, we cluster each feature using fuzzy c-means clustering (FCM) [31]. After clustering, each feature value except the body color is represented by the cluster center to which it belongs.

TABLE 1. Input features.

Feature Group	Features
Candlestick components [3]	the lengths of upper shadow line, lower shadow line, and body; the body color
Overlap studies [30]	BBANDS, DEMA, EMA, HT-TRENDLINE, KAMA, MA, MAMA, MIDPOINT, MIDPRICE, SAR, SAREXT, SMA, T3, TEMA, TRIMA, WMA
Momentum indicators [30]	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 [30]	AD, ADOSC, OBV
Volatility indicators [30]	ATR, NATR, TRANGE

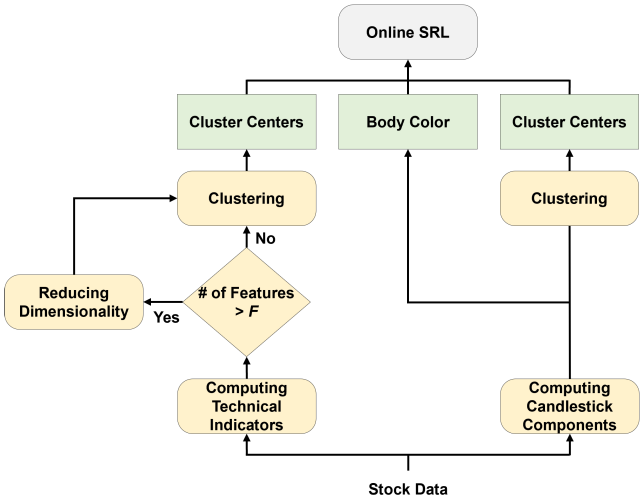

FIGURE 2. Architecture for offline SRL.

The online SRL, of which the architecture is shown in Fig. 3, takes input as a sliding window of outputs from the offline SRL to see historical data. To focus on important features in a window, we assign weights to the features using the gate structure [4]. The gate g shown in Fig. 4 uses a sigmoid activation function σ, as in Equation (17), where f denotes the input feature vector. Parameters W and b are learned using end-to-end training. In subsequent steps, we use the

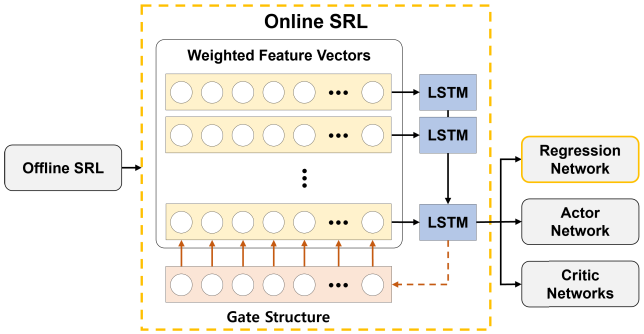

FIGURE 3. Architecture for online SRL.

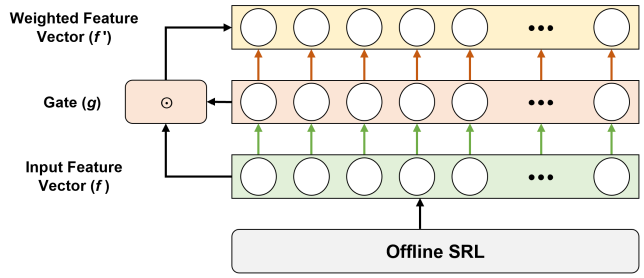

FIGURE 4. Gate structure for weighting features.

weighted feature vector f' in Equation (18) where $\odot$ denotes the element-wise multiplication of vectors g and f.

$$g = \sigma(W \cdot f + b) \tag{17}$$
$$f' = g \odot f \tag{18}$$

After weighting the features, we apply an LSTM layer to learn temporal characteristics. The input of the LSTM layer is a sliding window of weighted feature vectors, and its output is the hidden state of the last time step as shown in Fig. 3. We term the network up to the LSTM layer as the *online SRL network*. Each actor and critic network has a corresponding online SRL network. We co-train the online SRL networks with actor and critic networks, respectively.

In addition to the online SRL networks, we train a regression network whose structure is shown in Fig. 5(a), which predicts the next closing price to provide accurate state information for the actor network. The MSE between the real and predicted prices is used as the loss function for the regression network. The predicted price does not participate in training the actor network, but the underlying online SRL network of the regression network does because the online SRL network is shared with the actor network as explained in the next section.

C. IMITATIVE REINFORCEMENT LEARNING

1) ACTION

On each trading day t, a trading action $a_t \in \{buy, hold, sell\} = \{1, 0, -1\}$ is taken. Because we use only the long position, the buy action precedes the sell action. The agent of the SIRL-Trader starts with a certain amount of capital. For

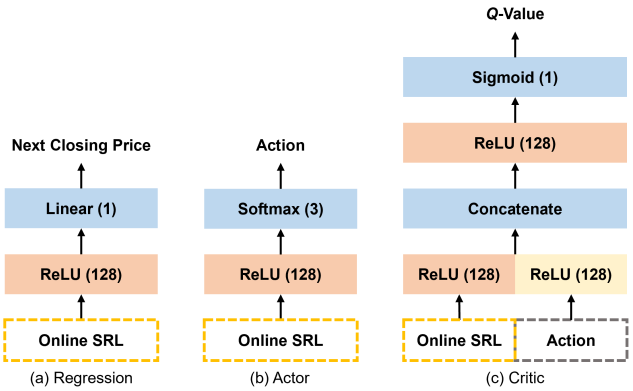

FIGURE 5. Neural network structures.

the sake of simplicity, the agent buys the maximum number of shares of a stock and sells all shares of the stock at the closing price.

2) REWARD

We define the reward r_t for action a_t as the change rate of the portfolio value as in Equation (19). The portfolio value V_t^p is the sum of the stock value V_t^s and remaining cash balance V_t^c, as in Equation (20).

$$r_t = a_t \times (V_{t+1}^p - V_t^p)/V_t^p \tag{19}$$
$$V_t^p = V_t^s + V_t^c \tag{20}$$

The stock value V_t^s is the current value of the stock, which is computed by multiplying the closing price p_t^c of the stock by the number of shares n_s owned, as in Equation (21). To simulate a real trading environment, we include a term ζ for transaction cost rate.

$$V_t^s = n_s \times p_t^c \times (1 - \zeta) \tag{21}$$

3) ALGORITHM

We incorporate offline/online SRL, behavior cloning, multistep learning, and dynamic delay into TD3. Algorithm 1 presents the proposed reinforcement learning algorithm. As shown in Fig. 6, we use two critic networks $Q_{\theta_1}, Q_{\theta_2}$, two online SRL networks o_{η_1}, o_{η_2} for $Q_{\theta_1}, Q_{\theta_2}$, an actor network μ_ϕ, an online SRL network λ_υ for μ_ϕ, and a regression network. The input for the online SRL networks is a sliding window w_t of weighted feature vectors $\{x_{t-N_w+1}, \ldots, x_{t-1}, x_t\}$ obtained from the offline SRL.

To accelerate the training process, we use multistep learning, which collects transitions of $\langle w_t, a_t, r_t, w_{t+1}\rangle$ using the N-step buffer $\mathcal{D}$ (lines 9 to 11). An N-step transition of $\langle w_{t-N+1:t+1}, a_{t-N+1:t}, r_{t-N+1:t}\rangle$ is constructed from the transitions stored in $\mathcal{D}$ (line 13) and used to compute the target value Y (line 16).

The actor network μ_ϕ, whose structure is shown in Fig. 5(b), is combined with the online SRL network λ_υ as shown in Fig. 6. To support discrete actions, we use the softmax layer as the output layer of the actor network.

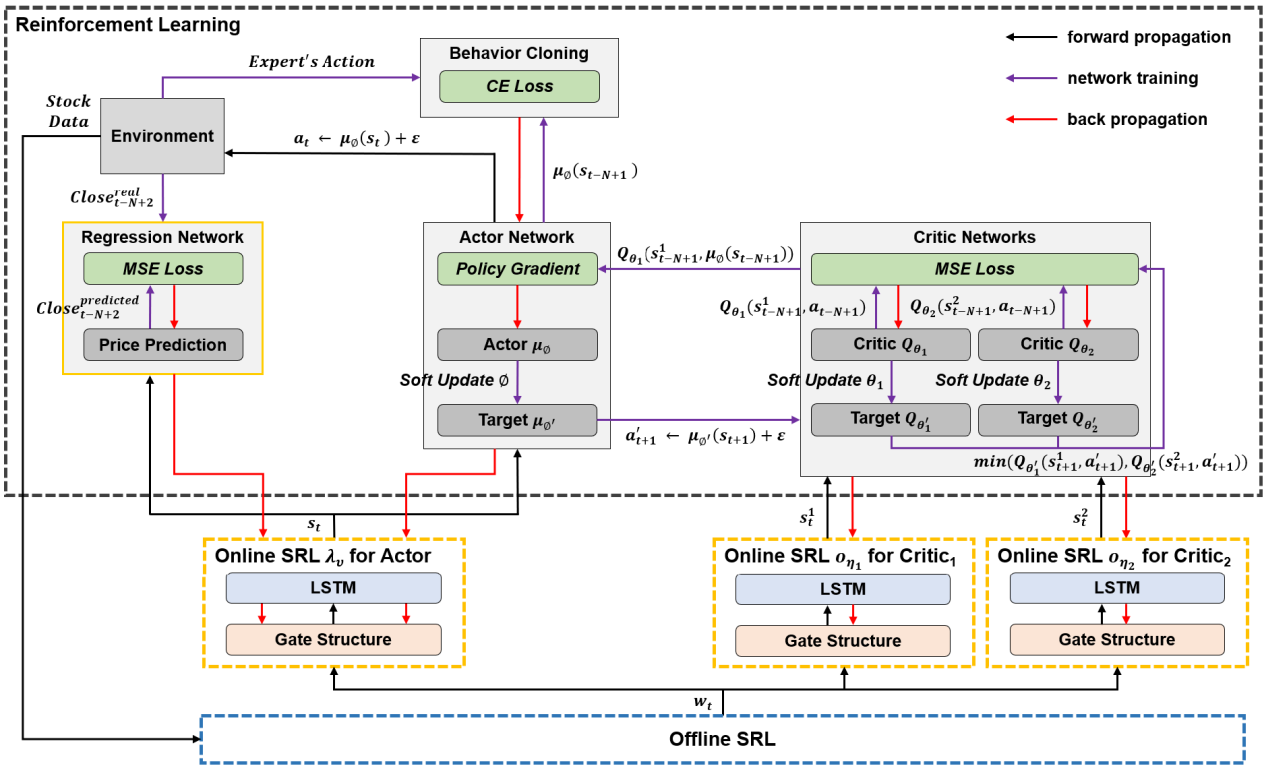

FIGURE 6. Architecture for reinforcement learning.

To select an action with exploration noise, we add a different amount of noise to each output of the softmax layer (line 9) and then apply **argmax**. As in the original TD3 algorithm, we use target policy smoothing as a regularization strategy (line 15). When the actor network μ_ϕ is updated, the online SRL network λ_υ is also updated by backpropagation.

The critic networks Q_{θ_1}, Q_{θ_2}, whose structure is shown in Fig. 5(c), are combined with the online SRL networks o_{η_1}, o_{η_2}, respectively. To solve the overestimation problem, we use the clipped double Q-learning technique of TD3 with multistep learning (lines 16 to 17). When the critic networks are updated, the corresponding online SRL networks are also updated by backpropagation.

To provide accurate state information for the actor network, we update the regression network using the MSE between the real and predicted prices (line 20). When the regression network is updated, the online SRL network λ_υ is also updated by backpropagation. The actor network is indirectly affected by this update through the underlying online SRL network λ_υ, which is shared with the regression network as shown in Fig. 6.

For imitation learning, we introduce a behavior-cloning technique to guide the actor network training. We create a prophetic trading expert who selects an action at day_{t-N+1} using information about today's closing price $close_{t-N+1}$ and tomorrow's closing price $close_{t-N+2}$. The expert buys when $close_{t-N+2} > h \times close_{t-N+1}$ and sells when $close_{t-N+2} < h \times close_{t-N+1}$ where $h \geq 1$ is a hyperparameter. Otherwise, the expert holds the stock. We train the actor network to minimize the cross-entropy (CE) loss between the softmax output vector of $\langle buy, hold, sell \rangle$ and the action a_{expert} of the expert, which is represented as a one-hot vector (line 21).

For more stable and efficient training, we propose a *dynamic delay* technique for updating the actor and target networks. In the original TD3 algorithm, the delay is fixed to a constant value, and it is hard to find an optimal value. The dynamic delay technique allows us to try various delay values while the reinforcement learning model is being trained. For each epoch, we compute the delay value d using Equation (22) (used in line 7). In Equation (22), α is a constant for adjusting the variance in delay values, and β is a constant for setting the minimum delay value.

$$d = (e \bmod \alpha) + \beta \tag{22}$$

D. DISCUSSION

Table 2 summarizes algorithm trading methods along three dimensions: offline SRL, online SRL, and reinforcement learning. SIRL-Trader is the only method that integrates all the techniques (dimensionality reduction, clustering, gate structure, regression model, imitation learning, multistep learning, and dynamic delay) for the three dimensions.

V. EXPERIMENTS

In this section, we present experiments designed to answer the following questions.

Algorithm 1 The Reinforcement Learning Algorithm of SIRL-Trader

Input: Sliding-window data $w_t \leftarrow \{x_{t-N_w+1}, \ldots, x_{t-1}, x_t\}$ obtained from the offline SRL

Initialize critic networks Q_{θ_1}, Q_{θ_2}, online SRL networks o_{η_1}, o_{η_2} for the critics
Initialize an actor network μ_ϕ, an online SRL network λ_υ for the actor, and a regression network for λ_υ
Initialize target networks: $\theta'_{1,2} \leftarrow \theta_{1,2}$, $\eta'_{1,2} \leftarrow \eta_{1,2}$, $\phi' \leftarrow \phi$, $\upsilon' \leftarrow \upsilon$
Initialize a replay buffer $\mathcal{B}$

for $e = 1$ to N_{epochs} **do**
  Initialize an N-step buffer $\mathcal{D}$
  Compute the delay value for each epoch: $d \leftarrow (e \bmod \alpha) + \beta$

  for $t = N_w$ to $T - 1$ **do**
    Select an action with exploration noise $\epsilon \sim \mathcal{N}(0, \sigma)$: $a_t \leftarrow \mu_\phi(s_t) + \epsilon$ where $s_t = \lambda_\upsilon(w_t)$
    Observe a reward r_t and the next input w_{t+1}
    Store a transition $\langle w_t, a_t, r_t, w_{t+1} \rangle$ to $\mathcal{D}$

    if $t \geq N_w + N - 1$ **then**
      Obtain an N-step transition $\langle w_{t-N+1:t+1}, a_{t-N+1:t}, r_{t-N+1:t} \rangle$ from $\mathcal{D}$ and store it to $\mathcal{B}$
      Sample a mini-batch of B transitions of length N from $\mathcal{B}$
      Smooth the target policy with $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma'), -c, c)$: $a'_{t+1} \leftarrow \mu_{\phi'}(s_{t+1}) + \epsilon$ where $s_{t+1} = \lambda_{\upsilon'}(w_{t+1})$
      $Y \leftarrow \sum_{i=0}^{N-1} \gamma^i r_{t-N+1+i} + \gamma^N \min_{j=1,2} Q_{\theta'_j}(s^j_{t+1}, a'_{t+1})$ where $s^j_{t+1} = o_{\eta'_j}(w_{t+1})$
      Update the critics θ_j by the MSE loss: $\frac{1}{B} \sum (Y - Q_{\theta_j}(s^j_{t-N+1}, a_{t-N+1}))^2$ where $s^j_{t-N+1} = o_{\eta_j}(w_{t-N+1})$

      if $t \bmod d$ **then**
        Update the actor ϕ by the deterministic policy gradient:
        $\frac{1}{B} \sum \nabla Q_{\theta_1}(s_{t-N+1}, a_{t-N+1})$ where $s_{t-N+1} = o_{\eta_1}(w_{t-N+1})$, $a_{t-N+1} = \mu_\phi(\lambda_\upsilon(w_{t-N+1}))$
        Update the regression network by the MSE loss: $\frac{1}{B} \sum (close^{real}_{t-N+2} - close^{predicted}_{t-N+2})^2$
        Update the actor ϕ by the CE loss for the behavior cloning: $\frac{1}{B} \sum \nabla CE(a_{t-N+1}, a_{expert})$ where $a_{t-N+1} = \mu_\phi(\lambda_\upsilon(w_{t-N+1}))$
        Soft-update the target networks: $\theta'_{1,2} \leftarrow \tau\theta_{1,2} + (1-\tau)\theta'_{1,2}$, $\eta'_{1,2} \leftarrow \tau\eta_{1,2} + (1-\tau)\eta'_{1,2}$, $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$, $\upsilon' \leftarrow \tau\upsilon + (1-\tau)\upsilon'$
      end if
    end if
  end for
end for

TABLE 2. Comparison of algorithmic trading methods.

	Offline SRL		Online SRL		Reinforcement Learning		
	dim. reduction	clustering	gate	regression	imitation	multi-step	dynamic delay
SIRL-Trader	○	○	○	○	○	○	○
K-line [3]	×	○	×	×	×	×	×
TFJ-DRL [4]	×	×	○	○	×	×	×
iRDPG [5]	×	×	×	×	○	×	×
GDPG [6]	×	×	×	×	×	×	×

- Can SIRL-Trader outperform state-of-the-art methods?
- What leads to the gain obtained by SIRL-Trader?
- How do hyperparameters affect the performance of SIRL-Trader?
- Is SIRL-Trader robust to high transaction cost rates?

A. EXPERIMENTAL SETUP

1) DATASETS

We test algorithmic trading methods using two datasets with different numbers of stocks included in the S&P 500 index. We obtain stock data that consist of opening, high, low, closing, and volume values from Yahoo Finance [32]. We use a smaller dataset to compare SIRL-Trader with other methods in detail. To ensure the diversity of price trends, we select six stocks with different price trends (upward, sideways, and downward), as shown in Fig. 7. We use a larger dataset to verify the generalization ability of the methods. Similar to [4], we select 56 stocks from twelve different sectors as shown in Table 4. For the two datasets, the training period is from Jan. 2014 to Dec. 2018 (1258 days), and the test period is from Jan. 2019 to Dec. 2020 (504 days). To simulate the real trading environment, we do not use any information from the current trading day for testing.

2) EVALUATION METRICS

For each method, we evaluate the rate of return $(V_{end} - V_{start})/V_{start}$ obtained using starting capital V_{start} of \$10,000 after a trading test period. We also evaluate the Sharpe ratio [33], which measures the return of an investment compared to its risk. In Equation (23), $\mathbb{E}[R]$ is the expected return, and $\sigma[R]$ is the standard deviation of the return, which is a measure of fluctuations, that is, the risk. A greater Sharpe ratio indicates a higher risk-adjusted return rate. We use the change rate of the portfolio value as the return in Equation (23).

$$Sharpe\_ratio = \frac{\mathbb{E}[R]}{\sigma[R]} \tag{23}$$

3) BASELINE METHODS

The state-of-the-art methods compared with SIRL-Trader are listed below. For each method, network structures and

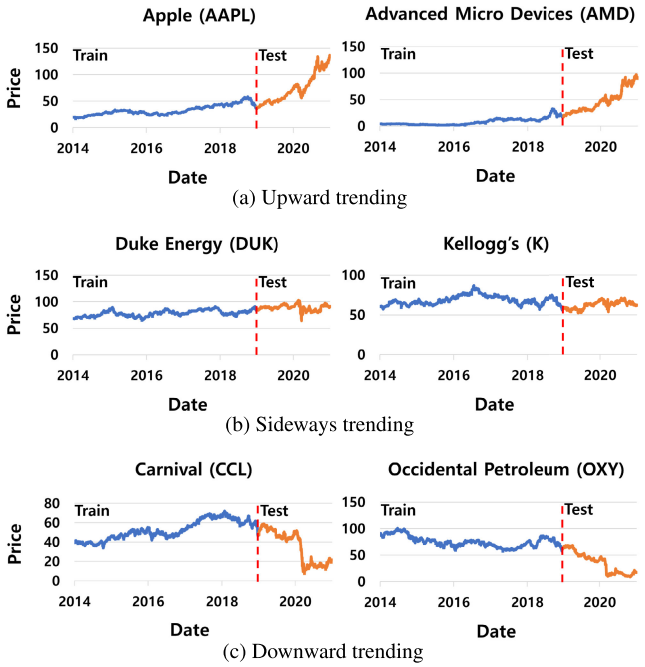

FIGURE 7. Stocks with different price trends.

hyperparameters are manually optimized as stated below.[1] We use the same reward function in Equation (19) and the Adam optimizer for all methods. For the hyperparameters of Adam optimizer, we set the momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, the epsilon to 10^{-7}, and the decay to 0.99.

- **Buy and Hold (B&H)** method buys the stock on the first day of the test and holds it throughout the test period. This directly reflects price trends.
- **K-line [3]** clusters candlestick components using the FCM method. The policy network comprises three ReLU dense layers with 128 units and a softmax output layer. We set the number of clusters to 5, the sliding window size to 10, the learning rate to 0.0002, and ϵ to 0.9 with a decay of 0.95.
- **TFJ-DRL [4]** uses a gate structure, GRU, temporal attention mechanism, and a regression network for SRL. It combines SRL with a policy gradient method. The policy and regression networks comprise two ReLU dense layers with 128 and 64 units, a dropout layer with an elimination fraction of 0.3, and a softmax output layer. We set the sliding window size to 3, the mini-batch size to 32, the learning rate to 0.0001, and ϵ to 0.7 with a decay of 0.9.
- **iRDPG [5]** uses GRU and imitative actor-critic reinforcement learning. The actor network comprises a ReLU dense layer with 16 units and a softmax output layer. The critic network comprises a ReLU dense layer with 16 units and a linear output layer. We set the sliding window size to 10, the standard deviation of the noise (or the *noise size*) for the exploration to 0.9, the mini-batch size to 32, and the learning rate to 0.0001.
- **GDPG [6]** uses GRU and DDPG. The actor network comprises two GRU layers with 20 and 24 units, a dropout layer with an elimination fraction of 0.3, and a softmax output layer. The critic network comprises a concatenate layer, two ReLU dense layers with 64 and 16 units, and a linear output layer. We set the sliding window size to 5, the noise size for the exploration to 0.6, the mini-batch size to 64, and the learning rate to 0.001.

[1]Unstated network structures and hyperparameters are set to the same as in the original paper.

4) IMPLEMENTATION DETAILS OF SIRL-TRADER

In the offline SRL, we set the tumbling window size for the z-score normalization to 20, the dimensionality threshold F in Fig. 2 to eight, and the number of clusters to 20. In the online SRL, we set the size of the input sliding window to five and the number of units of the LSTM layer to 128. In the reinforcement learning, we set the transaction cost rate ζ in Equation (21) to 0.25%, the N of the multistep learning to two, the h for determining the action of the expert to 1.001, the α and β of the dynamic delay to four and two, the noise size σ for the exploration to 0.7, the noise size σ' for the regularization to 0.7, the clipping size c to 1, the mini-batch size to 64, and the learning rate to 0.0001.

B. EXPERIMENTAL RESULTS

1) COMPARISON WITH OTHER METHODS

We compare SIRL-Trader with other methods in detail using the smaller dataset with various price trends. As shown in Table 3, for the stocks trending upward such as AAPL and AMD, all methods make good profits, but SIRL-Trader has the best return rate and Sharpe ratio. For the stocks trending sideways such as DUK and K, SIRL-Trader is the best in K; GDPG is the best in DUK, but the difference between the return rates of GDPG and SIRL-Trader is small. For the stocks trending downward such as CCL and OXY, most of the methods suffer losses, but SIRL-Trader makes a good profit. Particularly in CCL, all other methods suffer significant losses, but SIRL-Trader makes a huge profit. Fig. 8, which is extracted from the test period in CCL, shows that SIRL-Trader can fully exploit the price fluctuations compared with other methods in the dotted area. We can also observe that TFJ-DRL trades too frequently and iRDPG too rarely. In summary, SIRL-Trader can yield significant profits in the stocks with different trends by integrating all the techniques in Table 2.

We verify the generalization ability of all methods using the larger dataset. As shown in Table 4 and Fig. 9, SIRL-Trader outperforms all other methods in terms of the minimum, maximum, and average return rate and Sharpe ratio. SIRL-Trader achieves an average return rate of 57.8%, which is 14.1%P higher than the second-highest method, iRDPG. The average Sharpe ratio of SIRL-Trader is 1.06, which is 0.25 higher than the second-highest

TABLE 3. Experimental results on the smaller dataset.

Rate of Return							
Stocks		K-line	GDPG	B&H	iRDPG	TFJ-DRL	SIRL-Trader
Upward Trending	AAPL	60.5%	271.1%	235.0%	237.6%	265.7%	**344.7%**
	AMD	94.8%	195.0%	385.8%	388.9%	355.7%	**427.7%**
Sideways Trending	DUK	-23.5%	**24.6%**	7.8%	6.7%	8.7%	20.0%
	K	-1.2%	8.6%	9.6%	8.3%	11.1%	**43.3%**
Downward Trending	CCL	-39.6%	-35.8%	-56.5%	-43.7%	-14.3%	**120.0%**
	OXY	-39.5%	-10.6%	-72.0%	-29.1%	27.9%	**39.3%**
Minimum		-39.6%	-35.8%	-72.0%	-43.7%	-14.3%	**20.0%**
Maximum		94.8%	271.1%	385.8%	388.9%	355.7%	**427.7%**
Average		8.6%	75.5%	84.9%	94.8%	109.1%	**165.8%**
Sharpe Ratio							
Stocks		K-line	GDPG	B&H	iRDPG	TFJ-DRL	SIRL-Trader
Upward Trending	AAPL	1.37	3.05	2.53	2.55	2.75	**3.39**
	AMD	1.28	1.77	2.34	2.36	2.31	**2.51**
Sideways Trending	DUK	-0.48	**1.12**	0.39	0.37	0.41	0.66
	K	0.13	0.41	0.43	0.40	0.86	**1.38**
Downward Trending	CCL	-0.75	0.02	-0.05	0.15	0.16	**1.56**
	OXY	-0.38	0.10	-0.49	-1.57	**1.09**	0.79
Minimum		-0.75	0.02	-0.49	-1.57	0.16	**0.66**
Maximum		1.37	3.05	2.53	2.55	2.75	**3.39**
Average		0.20	1.08	0.86	0.71	1.26	**1.71**
Number of Trading Actions							
Stocks		K-line	GDPG	B&H	iRDPG	TFJ-DRL	SIRL-Trader
Upward Trending	AAPL	**234**	30	1	1	5	27
	AMD	**124**	75	1	1	3	25
Sideways Trending	DUK	**123**	104	1	1	6	20
	K	**109**	67	1	1	52	73
Downward Trending	CCL	132	**151**	1	15	118	78
	OXY	**78**	62	1	8	22	37

method, iRDPG. These results indicate that the reinforcement learning algorithm of SIRL-Trader, which is integrated with offline/online SRL, is effective for generalization.

2) ABLATION STUDIES

We conduct ablation studies to demonstrate the contribution of each component of SIRL-Trader. We exclude the components one by one and report the results in Fig. 10. The excluded components are dimensionality reduction (Dim), clustering (Clu), gate structure (Gat), multistep learning (Mul), regression model (Reg), and imitation learning (Imi); 'All' denotes SIRL-Trader with all the components. We evaluate the minimum, maximum, and average performance on the smaller dataset. The results show that all the components contribute to improving the performance. In particular, the offline SRL (Dim and Clu) and the gate structure are very crucial.

To demonstrate the effectiveness of the dynamic delay, we compare it with static delays ranging from two[2] to five. As shown in Fig. 11, the dynamic delay significantly improves the performance compared with the static delays.

3) COMPARISON WITH OTHER HYPERPARAMETER VALUES

We conduct experiments to evaluate the effect of important hyperparameters of SIRL-Trader, including the threshold for the dimensionality reduction, number of clusters, sliding window size, and noise size for the exploration. We evaluate the average return rate on the smaller dataset.

From Figs. 12(a) and (b), we can see that if the dimensionality threshold (or the number of clusters) is too small or too large, it degrades the performance. This is because of the trade-off between information loss (if too small) and noise inclusion (if too large). Fig. 12(c) shows that as the

[2]In the TD3 paper [7], the static delay was set to two.

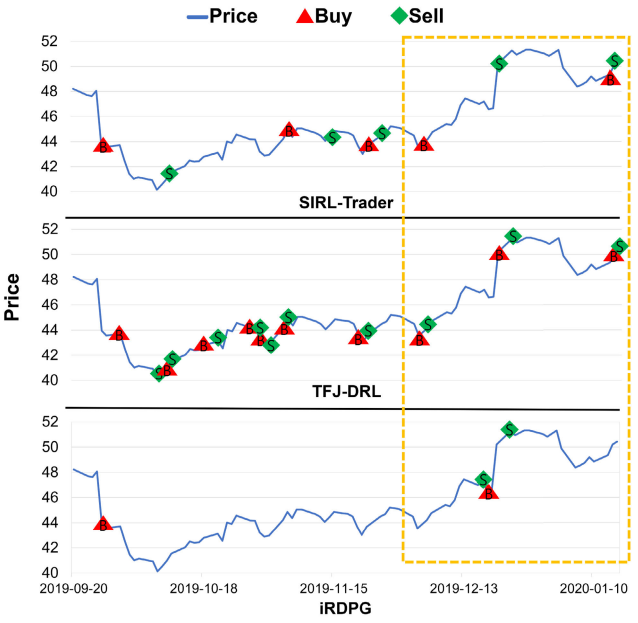

FIGURE 8. The trading actions performed in CCL.

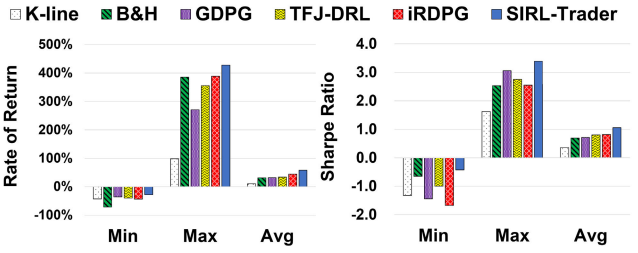

FIGURE 9. Experimental results on the larger dataset.

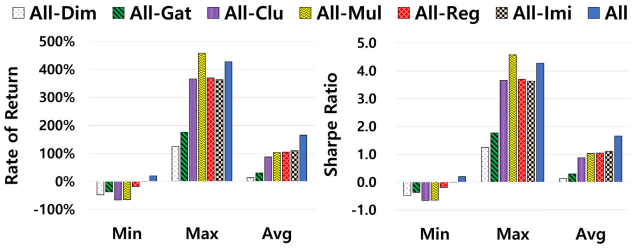

FIGURE 10. Experimental results of the ablation studies.

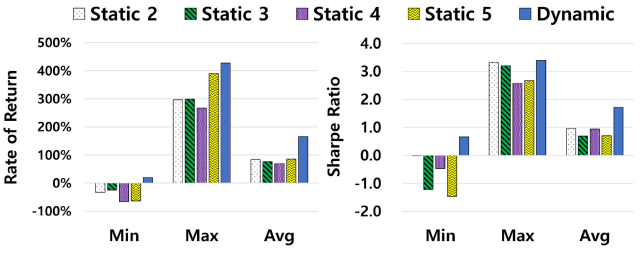

FIGURE 11. Comparison of static and dynamic delays.

TABLE 4. Experimental results on the larger dataset.

		Rate of Return						Sharpe Ratio					
Sectors	Stocks	K-line	B&H	GDPG	TFJ-DRL	iRDPG	SIRL-Trader	K-line	B&H	GDPG	TFJ-DRL	iRDPG	SIRL-Trader
Communication Services	CMCSA	26.1%	51.9%	21.8%	36.8%	48.7%	**85.4%**	1.04	1.19	0.70	1.28	1.14	**2.06**
	EA	70.9%	78.0%	17.5%	50.0%	76.0%	**82.3%**	**1.44**	1.37	1.26	1.09	1.35	1.42
	FB	51.4%	99.9%	**117.4%**	85.4%	98.9%	106.6%	1.11	1.57	1.78	**2.43**	1.56	1.64
	NFLX	98.5%	100.5%	60.8%	86.2%	104.2%	**108.1%**	1.62	1.50	1.20	1.39	1.54	**1.63**
	OMC	-4.6%	-14.4%	3.9%	**5.0%**	-11.3%	-9.5%	0.10	-0.04	0.28	**0.31**	0.03	0.07
	T	-17.3%	-2.9%	6.4%	7.0%	-3.8%	**10.8%**	-0.32	0.13	0.49	0.37	0.10	**0.71**
Consumer Discretionary	BBY	26.1%	85.5%	79.8%	94.5%	96.2%	**101.7%**	0.79	1.31	1.26	1.46	1.59	**1.76**
	CCL	-39.6%	-56.5%	-35.8%	-14.3%	-43.7%	**120.0%**	-0.75	-0.05	0.02	0.16	0.15	**1.56**
	F	-19.7%	11.0%	27.9%	3.5%	21.3%	**51.9%**	-0.15	0.47	0.84	0.25	0.62	**1.07**
	GPC	12.7%	5.7%	-7.8%	12.7%	32.4%	**57.8%**	0.49	0.37	0.10	0.49	0.80	**1.35**
	HRB	-32.7%	-38.1%	-18.1%	-29.3%	**14.9%**	-17.6%	-1.33	-0.41	-1.13	-0.24	**0.54**	-0.43
	WHR	3.1%	66.4%	62.8%	81.8%	**185.6%**	73.7%	0.29	1.08	1.61	1.22	**2.35**	1.14
Consumer Staple	HSY	-8.8%	43.7%	24.1%	**51.5%**	42.0%	45.9%	-0.11	1.10	0.78	**1.28**	1.08	1.17
	K	-1.2%	9.6%	8.6%	11.1%	8.3%	**43.3%**	0.13	0.43	0.41	0.86	0.40	**1.38**
	MO	-15.4%	-17.0%	-14.7%	**-1.9%**	-9.6%	-7.5%	-0.42	-0.21	-0.22	**0.13**	-0.10	0.02
	PG	9.6%	51.8%	55.4%	26.8%	**57.6%**	54.0%	0.47	1.33	1.48	0.85	**1.79**	1.38
	SYY	2.8%	19.4%	**48.5%**	21.4%	17.8%	20.3%	0.38	0.61	**1.73**	0.63	1.52	0.64
	WMT	25.8%	54.0%	**60.4%**	25.7%	54.0%	**60.4%**	1.34	1.42	**1.67**	1.31	1.42	1.57
Health Care	ABT	-3.7%	56.8%	**68.3%**	51.0%	55.3%	62.3%	0.09	1.25	**1.49**	1.33	1.23	1.41
	COO	20.4%	43.6%	28.5%	26.6%	40.9%	**69.8%**	0.73	1.05	0.80	0.77	1.01	**1.69**
	VRTX	11.3%	43.0%	31.1%	37.4%	40.8%	**70.3%**	0.48	0.95	1.03	0.91	0.92	**1.43**
	XRAY	9.1%	37.7%	30.8%	23.3%	**76.9%**	43.9%	0.42	0.85	0.76	1.24	**1.68**	0.96
Energy	APA	**55.8%**	-47.7%	-20.0%	-39.9%	-36.5%	-28.1%	**0.94**	0.22	-1.01	0.27	-1.68	0.42
	COP	-29.7%	-36.9%	-24.2%	-27.1%	-5.0%	**24.5%**	-0.49	-0.21	-0.11	-0.05	-0.28	**0.68**
	OXY	-39.5%	-72.0%	-10.6%	27.9%	-29.1%	**39.3%**	-0.38	-0.49	0.10	**1.09**	-1.57	0.79
	SLB	**-14.3%**	-41.3%	-27.8%	-37.9%	-28.0%	-18.9%	0.01	-0.21	-0.06	-0.17	-0.71	**0.14**
	XOM	-43.2%	-40.9%	-36.1%	-32.4%	-23.4%	**-10.1%**	-0.83	-0.66	-1.44	-1.00	-0.26	**0.06**
Financials	AMG	26.8%	3.3%	17.9%	24.1%	37.7%	**52.5%**	0.75	0.38	0.74	0.82	0.86	**0.99**
	COF	**30.8%**	27.5%	26.3%	13.3%	26.3%	29.1%	**0.77**	0.70	0.69	0.64	0.69	0.72
	NTRS	7.9%	10.5%	7.5%	9.7%	9.1%	**13.3%**	0.40	0.46	0.39	0.43	0.44	**0.50**
	STT	-24.5%	13.5%	**37.2%**	3.4%	13.0%	22.8%	-0.50	0.52	**0.84**	0.30	0.51	0.64
Industrials	EXPD	17.0%	40.8%	43.8%	21.6%	**68.9%**	57.9%	0.68	1.09	1.16	0.71	**2.30**	1.48
	GE	3.4%	39.2%	33.4%	78.8%	38.0%	**86.7%**	0.29	0.81	0.76	1.17	0.81	**1.22**
	JCI	16.7%	52.2%	16.1%	43.6%	46.4%	**119.9%**	0.58	1.15	0.58	1.07	1.16	**2.36**
Information Technology	AAPL	60.5%	235.0%	271.1%	265.7%	237.6%	**344.7%**	1.37	2.53	3.05	2.75	2.55	**3.39**
	AMD	94.8%	385.8%	195.0%	355.7%	388.9%	**427.7%**	1.28	2.34	1.77	2.31	2.36	**2.51**
	CSCO	21.6%	3.9%	5.5%	2.7%	**31.4%**	15.7%	0.81	0.32	0.34	0.30	**0.88**	0.55
	IBM	-15.3%	8.9%	-2.0%	7.1%	7.6%	**12.5%**	-0.44	0.42	0.13	0.38	0.39	**0.49**
	INTC	-16.2%	5.5%	27.9%	**33.6%**	33.3%	19.2%	-0.27	0.39	0.78	**1.29**	0.83	0.95
	ORCL	4.6%	42.7%	42.7%	15.8%	42.0%	**50.9%**	0.30	1.01	1.02	0.56	1.00	**1.20**
Materials	APD	30.0%	70.0%	65.8%	**72.1%**	68.7%	**72.1%**	0.87	1.36	1.32	1.54	1.34	**1.58**
	FCX	81.8%	150.5%	150.8%	111.5%	155.4%	**201.7%**	1.41	1.52	1.79	1.43	1.55	**2.36**
	NEM	-0.9%	73.3%	72.4%	27.8%	65.9%	**73.9%**	0.21	1.31	1.32	1.14	1.23	**1.34**
	NUE	-2.7%	1.6%	-4.5%	7.3%	10.5%	**15.3%**	0.05	0.31	0.19	0.40	**0.80**	0.54
	VMC	42.8%	51.4%	19.8%	49.5%	41.0%	**64.6%**	0.98	1.01	0.61	1.00	0.89	**1.31**
Real Estate	DRE	10.5%	59.2%	47.0%	31.2%	57.1%	**73.2%**	0.53	1.19	1.08	1.13	1.17	**1.47**
	EQR	17.5%	-7.2%	**19.5%**	1.7%	19.4%	18.9%	**0.72**	0.12	0.68	0.21	0.65	0.60
	HST	**49.3%**	-11.0%	-10.4%	3.3%	44.6%	38.8%	**1.16**	0.19	-0.53	0.32	0.89	1.01
	IRM	-6.2%	-8.4%	0.3%	**13.0%**	12.8%	9.2%	-0.26	0.07	0.29	0.61	**1.04**	0.42
	VNO	**1.3%**	-38.1%	-18.2%	-13.9%	-35.7%	-16.2%	**0.26**	-0.31	-0.28	0.09	-1.33	0.08
Utilities	AES	37.5%	65.3%	22.4%	67.9%	**91.4%**	76.6%	0.85	1.13	0.71	1.39	**1.76**	1.74
	DUK	-23.5%	7.8%	**24.6%**	8.7%	6.7%	20.0%	-0.48	0.39	**1.12**	0.41	0.37	0.66
	ETR	-6.9%	18.6%	**42.8%**	4.3%	17.1%	35.6%	-0.07	0.59	**1.19**	0.38	0.57	0.89
	FE	-16.3%	-16.8%	**8.2%**	-5.2%	-13.6%	-0.8%	-0.11	-0.05	**0.42**	0.13	-0.11	-0.39
	PPL	-19.9%	0.1%	-6.1%	0.1%	-2.0%	**7.8%**	-0.25	0.25	0.13	0.25	0.21	**0.40**
ETF	SPY	7.7%	47.9%	50.5%	50.5%	48.5%	**54.0%**	0.46	1.29	1.35	**1.87**	1.30	1.42
Minimum		-43.2%	-72.0%	-36.1%	-39.9%	-43.7%	**-28.1%**	-1.33	-0.66	-1.44	-1.00	-1.68	**-0.43**
Maximum		98.5%	385.8%	271.1%	355.7%	388.9%	**427.7%**	1.62	2.53	3.05	2.75	2.55	**3.39**
Average		10.4%	30.7%	31.5%	33.7%	43.7%	**57.8%**	0.35	0.69	0.70	0.80	0.81	**1.06**

sliding window size increases, the performance decreases. This is because price information older than one week does not aid in decision-making and just increases the amount of noise. Fig. 12(d) shows that if the noise size is too small or too large, it degrades the performance. This is because of the exploration-exploitation trade-off. Greater noise means more exploration and less exploitation. Smaller noise indicates the opposite.

4) ROBUSTNESS STUDY

In a real trading environment, transaction costs such as transaction fees, taxes, and trading slippages[3] exist. We study the robustness of SIRL-Trader by varying the transaction cost rate ζ in Equation (21). We evaluate the average return rate

[3]The difference between the expected price at which a trade takes place and the actual price at which the trade is executed.

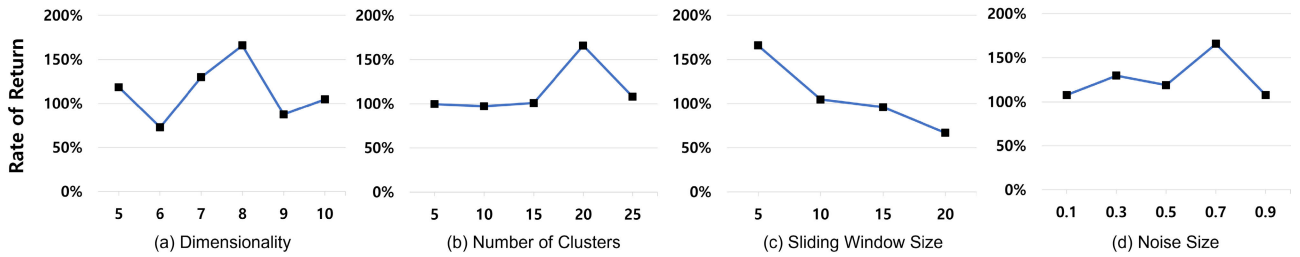

FIGURE 12. Comparison with other hyperparameter values.

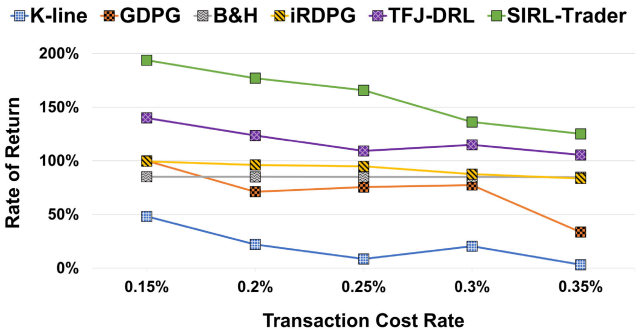

FIGURE 13. Experimental results of the robustness study.

on the smaller dataset. Fig. 13 shows that the average return decreases as the transaction cost rate increases for all methods. However, SIRL-Trader shows the best results regardless of the transaction cost rate. We observe that, though the transaction cost rate of 0.35% is much higher than that of real trading environments, SIRL-Trader still makes a good profit.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed a practical algorithmic trading method, SIRL-Trader, which achieves good profit using only long positions. We used offline/online SRL and imitative reinforcement learning to learn a profitable trading policy from nonstationary and noisy stock data. In the offline SRL, we used dimensionality reduction and clustering to extract robust features. In the online SRL, we co-trained a regression model with a reinforcement learning model to provide accurate state information for decision-making. In the imitative reinforcement learning, we incorporated a behavior cloning technique with the TD3 algorithm and applied multistep learning and dynamic delay to TD3. The experimental results showed that SIRL-Trader yields significantly higher profits and has superior generalization ability compared with state-of-the-art methods. We expect our approach to be generalizable to other complex sequential decision-making problems. SIRL-Trader can be extended to support short positions by redefining the action space and reward of reinforcement learning. We plan to apply our work to future markets where both long and short positions are always possible.

APPENDIX

TABLE 5. List of abbreviations.

Abbreviation	Description
A3C	Asynchronous Advantage Actor-Critic
B&H	Buy and Hold
CE	Cross Entropy
DPG	Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Network
DQN	Deep Q-Network
FCM	Fuzzy C-Means
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
MSE	Mean Squared Error
SIRL	State representation learning and Imitative Reinforcement Learning
SRL	State Representation Learning
TD3	Twin-Delayed Deep Deterministic policy gradient

REFERENCES

[1] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2017.

[2] Y. Li, W. Zheng, and Z. Zheng, "Deep robust reinforcement learning for practical algorithmic trading," *IEEE Access*, vol. 7, pp. 108014–108022, 2019.

[3] D. Fengqian and L. Chao, "An adaptive financial trading system using deep reinforcement learning with candlestick decomposing features," *IEEE Access*, vol. 8, pp. 63666–63678, 2020.

[4] K. Lei, B. Zhang, Y. Li, M. Yang, and Y. Shen, "Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading," *Expert Syst. Appl.*, vol. 140, Feb. 2020, Art. no. 112872.

[5] Y. Liu, Q. Liu, H. Zhao, Z. Pan, and C. Liu, "Adaptive quantitative trading: An imitative deep reinforcement learning approach," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2020, vol. 34, no. 2, pp. 2128–2135.

[6] X. Wu, H. Chen, J. Wang, L. Troiano, V. Loia, and H. Fujita, "Adaptive stock trading strategies with deep reinforcement learning methods," *Inf. Sci.*, vol. 538, pp. 142–158, Oct. 2020.

[7] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. ICML*, 2018, pp. 1587–1596.

[8] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, nos. 1–2, pp. 99–134, 1998.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, and A. Graves, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] R. Bellman, "On the theory of dynamic programming," *Proc. Nat. Acad. Sci. USA*, vol. 38, no. 8, p. 716, 1952.
[11] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI*, vol. 30, no. 1, 2016, pp. 2094–2100.
[12] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI*, vol. 32, no. 1, 2018, pp. 3215–3222.
[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. ICLR*, 2016.
[14] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.
[15] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, vol. 135. Cambridge, U.K.: MIT Press, 1998.
[16] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proc. ICML*, 2017, pp. 449–458.
[17] M. Fortunato, M. G. Azar, B. Piot, J. Menick, M. Hessel, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, and C. Blundell, "Noisy networks for exploration," in *Proc. ICLR*, 2018.
[18] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. NIPS*, 2000, pp. 1057–1063.
[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. ICML*, 2014, pp. 387–395.
[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. ICLR*, 2016.
[21] L. J. Cao and F. E. H. Tay, "Support vector machine with adaptive parameters in financial time series forecasting," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1506–1518, Nov. 2003.
[22] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Deep learning for event-driven stock prediction," in *Proc. IJCAI*, 2015, pp. 2327–2333.
[23] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Forecasting stock prices from the limit order book using convolutional neural networks," in *Proc. IEEE 19th Conf. Bus. Inform. (CBI)*, vol. 1, Jul. 2017, pp. 7–12.
[24] E. Chong, C. Han, and F. C. Park, "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies," *Expert Syst. Appl.*, vol. 83, pp. 187–205, Oct. 2017.
[25] L. Zhang, C. Aggarwal, and G.-J. Qi, "Stock price prediction via discovering multi-frequency trading patterns," in *Proc. 23rd Int. Conf. Knowl. Discovery Data Mining (ACM SIGKDD)*, Aug. 2017, pp. 2141–2149.
[26] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj, "Temporal attention-augmented bilinear network for financial time-series data analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 5, pp. 1407–1418, May 2018.
[27] F. Feng, X. He, X. Wang, C. Luo, Y. Liu, and T.-S. Chua, "Temporal relational ranking for stock prediction," *ACM Trans. Inf. Syst.*, vol. 37, no. 2, pp. 1–30, Mar. 2019.
[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[29] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734.
[30] *TA-Lib: Technical Analysis Library*. Accessed: Sep. 11, 2021. [Online]. Available: http://ta-lib.org/
[31] J. C. Bezdek, W. Full, and R. Ehrlich, "FCM: The fuzzy *c*-means clustering algorithm," *Comput. Geosci.*, vol. 10, nos. 2–3, pp. 191–203, 1984.
[32] *Yahoo Finance*. Accessed: Sep. 11, 2021. [Online]. Available: https://finance.yahoo.com/
[33] W. F. Sharpe, "The Sharpe ratio," *J. Portfolio Manage.*, vol. 21, no. 1, pp. 49–58, 1994.

DEOG-YEONG PARK received the B.S. degree in computer engineering from Kwangwoon University, Seoul, Republic of Korea, in 2019, where he is currently pursuing the Master of Engineering degree.

KI-HOON LEE received the B.S., M.S., and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea, in 2000, 2002, and 2009, respectively.

From 2010 to 2012, he was a Manager with the Advanced Institute of Technology, Korea Telecom (KT). From 2012 to 2013, he was a Senior Developer with SAP Labs Korea. He joined Kwangwoon University, in 2013, where he is currently a Professor with the School of Computer and Information Engineering. He has published papers in leading international journals and conferences, including IEEE Access, *VLDB Journal*, *SIGMOD Record*, and the IEEE ICDE.