Expert Systems with Applications 158 (2020) 113573

Contents lists available at ScienceDirect

Expert Systems with Applications

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

An intelligent financial portfolio trading strategy using deep Q-learning

Hyungjun Park [a], Min Kyu Sim [b], Dong Gu Choi [a,*]

[a] Department of Industrial and Management Engineering, Pohang University of Science and Technology, 77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk 37673, Republic of Korea
[b] Department of Industrial Engineering, Seoul National University of Science and Technology, 232, Gongneung-Ro, Nowon-Gu, Seoul 01811, Republic of Korea

ARTICLE INFO

Article history:
Received 3 January 2020
Revised 26 April 2020
Accepted 14 May 2020
Available online 18 May 2020

Keywords:
Portfolio trading
Reinforcement learning
Deep Q-learning
Deep neural network
Markov decision process

ABSTRACT

Portfolio traders strive to identify dynamic portfolio allocation schemes that can allocate their total budgets efficiently through the investment horizon. This study proposes a novel portfolio trading strategy in which an intelligent agent is trained to identify an optimal trading action using deep Q-learning. We formulate a Markov decision process model for the portfolio trading process that adopts a discrete combinatorial action space and determines the trading direction at a prespecified trading size for each asset, thus ensuring practical applicability. Our novel portfolio trading strategy takes advantage of three features to outperform other strategies in real-world trading. First, a mapping function is devised to handle and transform any action that is initially proposed but found to be infeasible into a similar and valuable feasible action. Second, by overcoming the dimensionality problem, this study establishes agent and Q-network models to derive a multi-asset trading strategy in the predefined action space. Last, this study introduces a technique that can derive a well-fitted multi-asset trading strategy by designing an agent to simulate all feasible actions in each state. To validate our approach, we conduct backtesting for two representative portfolios and demonstrate superior results over the benchmark strategies.

1. Introduction

A goal of financial portfolio trading is maximizing the trader's monetary wealth by allocating capital to a basket of assets in a portfolio over the periods during the investment horizon. Thus, portfolio trading is the most important investment practice in the buy-side financial industry. Portfolio traders strive to establish trading strategies that can properly allocate capital to financial assets in response to time-varying market conditions. Typical objective functions for trading strategy optimization include expected returns and the Sharpe ratio (i.e., risk-adjusted returns). In addition to optimizing an objective function, a trading strategy should achieve a reasonable turnover rate so that it is applicable to real-world financial trading. If the turnover rate is not reasonable, transaction costs hurt overall trading performance.

Portfolio trading is an optimization problem that involves a sequential decision-making process across multiple rebalancing periods. In this process, the stochastic components of time-varying market variables should be considered. Thus, the problem of deriving an optimal portfolio trading strategy has traditionally been formulated as a stochastic optimization problem (Consigli & Dempster, 1998; Golub, Holmer, Mckendall, Polhlman, & Zenios, 1995; Kouwenberg, 2001). To handle these stochastic components over multiple periods, most related studies have developed heuristic methods (Brock, Lakonishok, & Lebaron, 1992; Chen & Yu, 2017; Chen & Chen, 2016; Derigs & Nickel, 2003; Leigh, Modani, Purvis, Wu, & Robert, 2002; Papailias & Thomakos, 2015; Zhang et al., 2015; Zhu & Zhou, 2009). In very recent years, reinforcement learning (RL) has become another popular approach for solving financial portfolio trading problems. In RL methods, a learning agent can understand a complex financial environment by attempting various trading actions and revising its trading action policy and can optimize its trading strategy based on these experiences. In addition, these methods have the important advantage that learning agents can update their trading strategies based on their experiences on future trading days. Instead of simply maintaining trading strategies derived from historical data, learning agents can adapt their strategies using their observed experiences on each real trading day (Wang et al., 2016). With these advantages and the increasing popularity of RL algorithms, many previous studies have applied RL algorithms to various portfolio trading problem settings (Almahdi & Yang, 2017, 2019, 2012, 2006, 2006, Deng, Feng, Kong, Ren, & Dai, 2016, 2014, 2019, 2017, 2001, 1998, 1996, 1998, 2006, 2018, 2015).

According to the recent evolution of RL methods, some researchers (Deng et al., 2016; Jeong & Kim, 2019; Jiang, Xu, &

* Corresponding author.
E-mail addresses: hjhjpark94@postech.ac.kr (H. Park), mksim@seoultech.ac.kr (M.K. Sim), dgchoi@postech.ac.kr (D.G. Choi).

https://doi.org/10.1016/j.eswa.2020.113573
0957-4174/

Liang, 2017) have started to use deep RL (DRL) methods, which combine RL and deep neural network (DNN), to overcome the unstable performance of previous RL methods. Nevertheless, we believe that this line of study is not yet mature in terms of practical applicability for the following two reasons. First, most studies based on DRL methods focus on single-asset trading (Deng et al., 2016; Jeong & Kim, 2019). Because most traders generally hold multiple securities, additional decision-making steps are necessary even if single asset trading rules are derived. Second, even in studies dealing with multi-asset portfolio trading, action spaces determine portfolio weights (Jiang et al., 2017). However, the action spaces that determine the portfolio weight cannot be followed directly in practice because the trader must also determine how to satisfy the target weight. We explain this issue in more detail in Section 2.3.

To overcome these limitations, this study proposes a new approach for deriving a multi-asset portfolio trading strategy using deep Q-learning (DQL), one of the most popular DRL methods. In this study, we focus on a multi-asset trading strategy and define an intuitive set of trading actions that can be interpreted as a direct investment guide for traders. Each action in the action space used in this study includes trading directions corresponding to each asset in a portfolio, and each trading direction involves either holding the corresponding asset or buying or selling the asset at a prespecified trading size. Although a recent study (Pendharkar & Cusatis, 2018) argues that optimizing a trading strategy based on a discrete action space has a negative effect, we find that our discrete action space modeling allows for a lower turnover rate and is more practical than continuous action space modeling is.

Deriving a multi-asset trading strategy that can directly guide intuitive trading actions is challenging for a few reasons. Our proposed approach includes techniques to tackle these challenges. First, setting a discrete combinatorial action space may lead to infeasible actions, and, thus, the model may derive an unreasonable trading strategy (i.e., a strategy with frequent and pointless portfolio weight changes that only lead to more transaction costs). To address this issue, our approach introduces a mapping function that enables the agent to prevent the selection of unreasonable actions by mapping infeasible actions onto similar and valuable actions. We argue that our proposed approach is novel because it is the first to address infeasible actions in the field of financial trading with RL methods. Moreover, the mapping function introduced in this study can be viewed as an extension of recent methods related to RL with a constrained action space, and this mapping function is a more desirable method of accounting for the characteristics of problem domain knowledge. By applying this mapping function, our approach can derive a reasonable trading strategy in the practical action space.

Second, the practical action space that determines a trading direction for each asset in the portfolio faces a dimensionality problem (Moody & Saffell, 2001). As the number of assets in the portfolio increases, the sizes of both the state and action spaces increase exponentially because a trading agent must determine a combination of trading directions for several assets in the portfolio. As a result, previous studies related to this action space (Casqueiro & Rodrigues, 2006; Dempster & Leemans, 2006; Deng et al., 2016; Jeong & Kim, 2019; Moody & Saffell, 2001) only consider a single asset or a single risky asset with risk-free asset trading. By using DRL with a technique for reducing the dimension of the state space, our proposed approach can handle more than two risky-assets in the practical action space, which is a significant advance in this research field.

Third, although this study utilizes financial data spanning several years, these data may not provide enough training data for the DRL agent to learn a multi-asset trading strategy in the financial environment because the substantial experience is necessary to learn a trading strategy that projects from a joint state to a joint action space containing multiple assets. However, the amount of available data is fixed, so we need to devise a technique to enable the agent to gain more experience within the training data and learn as much as possible. Our approach can achieve sufficient learning by simulating all feasible actions in parallel in each state and then updating the agent's trading strategy using the learning experiences from the simulation results. This technique allows the agent to gain enough experience and learn enough to derive a well-fitted multi-asset trading strategy even when only limited data are available.

The rest of this paper is organized as follows. In Section 2, we first review the related literature and present the differences between our study and previous studies. Section 3 describes the definition of our problem, and Section 4 introduces our approach for deriving an intelligent trading strategy. In Section 5, we provide experimental results to validate the advantages of our approach. Finally, we conclude in Section 6 by providing relevant implications and identifying directions for future research.

2. Literature review

Portfolio trading is an optimization problem that involves a sequential decision-making process over multiple rebalancing periods. In addition, the stochastic components of market variables should be considered in this process. Thus, traditionally, the derivation of a portfolio trading strategy has been formulated as a stochastic programming problem to find an optimal trading strategy. Recently, much effort has been made to solve this stochastic optimization problem using a learning-based approach, the RL approach. To formulate this stochastic optimization problem, it is necessary to determine how to measure the features of the stochastic components corresponding to changes in the financial market. Utilizing technical indicators is more common than utilizing the fundamental indicators of securities in daily frequency portfolio trading, as in our study.

This section reviews how previous studies have attempted to model stochastic market components to formulate the portfolio trading problem and derive an optimal trading strategy. Section 2.1 provides a brief description of previous studies that formulate the stochastic components of the financial market. Section 2.2 reviews previous studies that discuss heuristic methods for deriving an optimal trading strategy. Section 2.3 reviews previous studies that address the stochastic optimization problem to derive an optimal trading strategy using RL.

2.1. Stochastic programming-based models

Early studies on portfolio trading and, sometimes, management used stochastic programming-based models. Stochastic programming models formulate a sequence of investment decisions over time that can maximize a portfolio manager's expected utility up to the end of the investment horizon. Golub et al. (1995) modeled an interest rate series as a binomial lattice scenario using Monte Carlo procedures to solve a money management problem with stochastic programming. Kouwenberg (2001) solved an asset-liability management problem using the event tree method to generate random stochastic programming coefficients. Consigli and Dempster (1998) used scenario-based stochastic dynamic programming to solve an asset-liability management problem. However, stochastic programming-based models have the limitation of needing to generate numerous scenarios to solve a complex problem, such as understanding a financial environment, resulting in a large computational burden.

2.2. Heuristic methods

Because of this limitation of stochastic programming-based models, many studies have devised heuristic methods (i.e., trading heuristics). One of the most famous such methods is technical analysis for asset trading. This method provides a simple and sophisticated way to identify hidden relationships between market features and asset returns through the study of historical data. Using these identified relationships, investments are made in assets by taking appropriate positions. Brock et al. (1992) conducted backtesting with real and artificial data using moving average and trading range strategies. Zhu and Zhou (2009) considered theoretical rationales for using technical analysis and suggested a practical moving average strategy to determine a portion of investments. Chourmouziadis and Chatzoglou (2016) suggested an intelligent stock-trading fuzzy system based on rarely used technical indicators for short-term portfolio trading. Another popular heuristic method is the pattern matching (i.e., charting heuristics) method, which detects critical market situations by comparing the current series of market features to meaningful patterns in the past. Leigh et al. (2002) developed a trading strategy using two types of bull flag pattern matching. Chen and Chen (2016) proposed an intelligent pattern-matching model based on two novel methods in the pattern identification process. The other well-known heuristic method is a metaheuristic algorithm that can find a near optimal solution in an acceptable computation time. Derigs and Nickel (2003) developed a decision support system generator for portfolio management using simulated annealing, and Potvin, Soriano, and Vallee (2004) applied genetic programming to generate trading rules automatically. Chen and Yu (2017) used a genetic algorithm to group stocks with similar price series to support investors in making more efficient investment decisions. However, these heuristic methods have limited ability to fully search a very large feasible solution space because they are inflexible. Thus, we need to be careful about the reliability of obtaining an optimal trading strategy using these methods.

2.3. Reinforcement learning-based methods

A recent research direction is optimizing a trading strategy using RL such that a learning agent develops a policy while interacting with the financial environment. Using RL, a learning-based method, the learning agent can search for an optimal trading strategy flexibly in a high-dimensional environment. Unlike supervised learning, RL allows learning from experience, meaning that the agent can be trained with unlabeled data obtained from interactions with the environment.

In the earliest such studies, Neuneier (1996) and Neuneier (1998) optimized multi-asset portfolio trading using Q-learning. In other early studies, Moody, Wu, Liao, and Saffell (1998) and Moody and Saffell (2001) used *direct* RL with *recurrent* RL as a base algorithm and derived a multi-asset long-short portfolio trading strategy and a single-asset trading rule, respectively. *direct* RL is policy-based RL, which optimizes an objective function by adjusting policy parameters, and *recurrent* RL is an RL algorithm in which the last action is received as an input. These studies introduced several measures, such as the Sharpe ratio and the Sterling ratio, as objective functions and compared the trading strategies derived using different objectives. Casqueiro and Rodrigues (2006) derived a single-asset trading strategy using Q-learning, which can maximize the Sharpe ratio. Dempster and Leemans (2006) developed an automated foreign exchange trading system using an adaptive learning system with a base algorithm of *recurrent* RL by dynamically adjusting a hyper-parameter depending on the market situation. Jangmin, Lee, Lee, and Zhang (2006) proposed a Q-learning-based local trading system that categorized an asset price series into four patterns and applied different trading rules. Bertoluzzo and Corazza (2012) suggested a single-asset trading system using Q-learning with linear and kernel function approximations. Eilers, Dunis, Mettenheim, and Breitner (2014) developed a trading rule for an asset with a seasonal price trend using Q-learning. Zhang et al. (2015) derived a trading rule generator using extended classifier systems combined with RL and a genetic algorithm. Almahdi and Yang (2017) suggested a *recurrent* RL-based trading decision system that enables multi-asset portfolio trading and compared the performance of the system across several different objective functions. Pendharkar and Cusatis (2018) suggested an indices trading rule derived using two different RL methods, on-policy (SARSA) and off-policy (Q-learning) methods, and compared the performance of these two methods. They also compared the performances of discrete and continuous agent action space modeling. Almahdi and Yang (2019) used a hybrid method that combined *recurrent* RL and particle swarm optimization to derive a portfolio trading strategy that considers real-world constraints.

More recently, DRL, which combines deep learning and RL algorithms, was developed, and, thus, studies have suggested using DRL-based methods to derive portfolio trading strategies. DRL methods enable an agent to understand a complex financial environment through deep learning and to learn a trading strategy by automatically applying an RL algorithm. Jiang et al. (2017) used a deep deterministic policy gradient, an advanced method of combining policy-based and value-based RL, and introduced various DNN structures and techniques to trade a portfolio consisting of several cryptocurrencies. Deng et al. (2016) derived an asset trading strategy using a *recurrent* RL-based algorithm, introduced a fuzzy deep recurrent neural network that used fuzzy representation to reduce uncertainty in noisy asset prices, and used a deep recurrent neural network to consider previous actions and utilize high-dimensional nonlinear features. Jeong and Kim (2019) derived an asset trading rule that determined trading action for an asset and the number of shares for the action taken. To learn this trading rule, Jeong and Kim (2019) used a deep Q-network (DQN) with a novel DNN structure consisting of two branches, one of which learned action values while the other learned the number of shares to take to maximize the objective function.

These studies all used various RL-based methods in different problem settings. All of the methods performed well in each setting, but some issues limit the applicability of these methods to the real world. First, some problem settings did not consider transaction costs (Bertoluzzo & Corazza, 2012; Eilers et al., 2014; Jeong & Kim, 2019; O et al., 2006; Pendharkar & Cusatis, 2018). A trading strategy developed without assuming transaction costs is likely to be impractical for real world applications. The second issue is that some strategies consider trading for only one asset (Almahdi & Yang, 2017; Bertoluzzo & Corazza, 2012; Casqueiro & Rodrigues, 2006; Dempster & Leemans, 2006; Eilers et al., 2014; Deng et al., 2016; Jeong & Kim, 2019; Moody & Saffell, 2001; Zhang et al., 2015). A trading strategy of investing in only one risky asset may have high risk exposure because it has no risk diversification. Finally, in previous studies deriving multi-asset portfolio trading strategies using RL, the agent's action space was defined as the portfolio weights in the next period (Almahdi & Yang, 2017; Almahdi & Yang, 2019; Jiang et al., 2017; Moody et al., 1998). The action spaces of these studies do not provide portfolio traders with a direct guide that is applicable to a real-world trading scenario that includes transaction costs because there are many different ways to transition from the current portfolio weight to the portfolio weight in the next period. Thus, previous studies using portfolio weights as the action space required finding a way to minimize transaction costs at each rebalancing moment. Rebalancing in a way that reduces both transaction costs and dispersion from the next target portfolio is not an easily solved problem (Grinold & Khan, 2000). In

addition, a portfolio trading strategy derived based on the action spaces of the previous studies may be difficult to apply to real-world trading because the turnover rate is likely to be high. An action space that determines portfolio weights can result in frequent asset switching because the amount of asset changes has no upper bound. Thus, we contribute to the literature by deriving a portfolio trading strategy that has no such issues.

3. Problem definition

In this study, we consider a portfolio consisting of cash and several risky assets. All assets in the portfolio are bought using cash, and the value gained from selling assets is held in cash. That is, the agent cannot buy an asset without holding cash and cannot sell an asset without holding the asset. This type of portfolio is called a long-only portfolio, which does not allow short selling. Our problem setting also has a multiplicative profit structure in that the portfolio value accumulates based on the profits and losses in previous periods. We consider proportional transaction costs that are charged according to a fixed proportion of the amount traded in transactions involving buying or selling. In addition, we allow the agent to partially buy or sell assets (e.g., the agent can buy or sell half of a share of an asset).

We set up some assumptions in our problem setting. First, transactions can only be carried out once a day, and all transactions in a day are made at the closing price in the market at the end of that day. Second, the liquidity of the market is high enough that each transaction can be carried out immediately for all assets. Third, the trading volume of the agent is very small compared to the size of the whole market, so the agent's trades do not affect the state transition of the market environment.

To apply RL to solve our problem, we need a model of the financial environment that reflects the financial market mechanism. Using the notations summarized in Table 1, we formulate a Markov decision process (MDP) model that maximizes the portfolio return rate in each period by selecting sequential trading directions for the individual assets in the portfolio according to time-varying market features (Table 2).

3.1. State space

The state space of the agent is defined as the weight vector of the current portfolio before the agent selects an action and the tensor that contains the market features (technical indicators) for the assets in the portfolio. This type of state space is similar to that used in a previous study (Jiang et al., 2017). That is, the state in period t can be represented as below (Eqs. (1)–(3)):

$$s_t = (X_t, w'_t), \tag{1}$$

$$w'_t = \left(w'_{t,0}, w'_{t,1}, w'_{t,2}, \ldots, w'_{t,I}\right)^T, \tag{2}$$

$$X_t = \left[K^c_t, K^o_t, K^h_t, K^l_t, K^v_t\right], \tag{3}$$

where w'_t denotes the weight vector of the current portfolio and X_t represents the technical indicator tensor for the assets in the portfolio. For this tensor, we use five technical indicators for the assets in the portfolio, as below (Eqs. (4)):

$$k^x_t = \left(k^x_{t,1}, k^x_{t,2}, \ldots, k^x_{t,I}\right)^T \quad \forall x \in \{c, o, h, l, v\}, \tag{4}$$

Every set of five technical indicators can be expressed as a matrix (Eq. (5)), where the rows represent each asset in the portfolio and the columns represent the series of recent technical indicators in the time window. Here, if we set a time window of size n (considering n-lag autocorrelation) and a portfolio of I assets, the technical indicator tensor is an $(I, n, 5)$-dimensional tensor, as in Fig. 1.

Table 1
Summary of notations.

Decision variables	
$a_t = (a_{t,1}, a_{t,2}, \ldots, a_{t,I})$	agent's action at the end of period t $\{a_t \in \mathbb{Z}^I : a_{t,i} \in \{-1, 0, 1\} \forall i\}$
Set and indices	
$i = 0, 1, 2, \ldots, I$	portfolio asset index (i=0 represents cash)
t	time period index
$F(s)$	feasible action set in state s
$S^-(a_t)$	set of indices of selling assets when the agent takes action a_t (i.e., $\{i \in \mathbb{Z} \mid 0 < i \leqslant I, a_{t,i} = -1\}$)
$S^+(a_t)$	set of indices of buying assets when the agent takes action a_t (i.e., $\{i \in \mathbb{Z} \mid 0 < i \leqslant I, a_{t,i} = 1\}$)
Parameters	
n	size of the time window containing recent previous market features
P_t	portfolio value changed by the action at the end of period t
P'_t	portfolio value before the agent takes an action at the end of period t
P^s_t	portfolio value at the end of period t when the agent takes no action at the end of the previous period $t-1$ (static portfolio value in period t)
$w_{t,i}$	proportion of asset i changed by the action at the end of period t
$w'_{t,i}$	proportion of asset i before the agent takes an action at the end of period t
$\hat{w}'_{t,i}$	auxiliary parameter used to derive $w_{t,i}$
c_t	decay rate of transaction costs at the end of period t
c^-	transaction cost rate for selling
c^+	transaction cost rate for buying
δ	trading size for selling or buying $\left(0 < \delta < \frac{P'_t}{I}\right)$
ρ_t	return rate of the portfolio in period t $\left(= \frac{P_t - P_{t-1}}{P_{t-1}}\right)$
$o_{t,i}$	opening price of asset i in period t
$p_{t,i}$	closing price of asset i in period t
$h_{t,i}$	highest price of asset i in period t
$l_{t,i}$	lowest price of asset i in period t
$v_{t,i}$	volume of asset i in period t

Table 2
Summary of market features.

Features	
$k^c_{t,i}$	rate of change of the closing price of asset i in period t $\left(= \frac{p_{t,i} - p_{t-1,i}}{p_{t-1,i}}\right)$
$k^o_{t,i}$	ratio of the opening price in period t to the closing price in period $t-1$ for asset i $\left(= \frac{o_{t,i} - p_{t-1,i}}{p_{t-1,i}}\right)$
$k^h_{t,i}$	ratio of the closing price to the highest price of asset i in period t $\left(= \frac{p_{t,i} - h_{t,i}}{h_{t,i}}\right)$
$k^l_{t,i}$	ratio of the closing price to the lowest price of asset i in period t $\left(= \frac{p_{t,i} - l_{t,i}}{l_{t,i}}\right)$
$k^v_{t,i}$	rate of change of the volume of asset i in period t $\left(= \frac{v_{t,i} - v_{t-1,i}}{v_{t-1,i}}\right)$

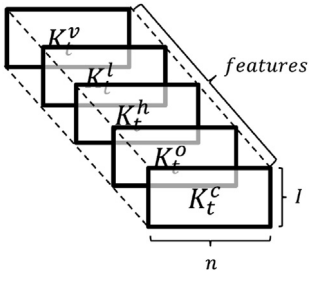


Fig. 1. Market feature tensor (X_t).

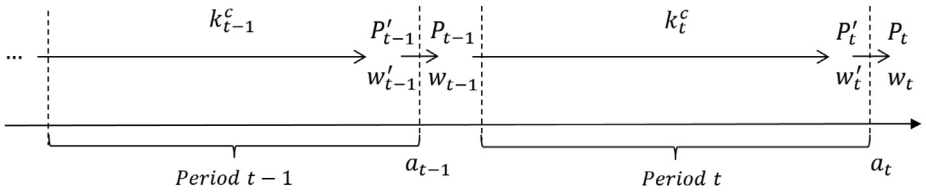


Fig. 2. Financial environment transition.

$$K^x_t = [k^x_{t-n+1}|k^x_{t-n+2}|\dots|k^x_t] \quad \forall x \in \{c, o, h, l, v\}, \tag{5}$$

3.2. Action space

We define the action space to overcome the limitations of the action spaces in previous studies. Agent actions determine which assets to hold and which assets to sell or buy at a prespecified constant trading size. For example, if a portfolio includes two assets and the trading size is 10,000 *USD*, then the agent can select the action of buying 10,000 *USD* of *asset1* and selling 10,000 *USD* of *asset2*. The action space includes the trading directions of buying, selling, or holding each asset in the portfolio, so the action space contains 3^I different actions. These actions are expressed in a vector form that includes trading directions for each asset in a portfolio. In addition, each trading direction $(sell, hold, buy)$ is encoded as $(-1, 0, 1)$, respectively. For example, an action that involves selling *asset1* and buying *asset2* can be encoded into the vector $(-1, 1)$.

Because the trading actions for individual assets are carried out at a fixed trading size, this action space is modeled as a discrete type. Although this discrete action space may not be able to derive a trading strategy that outperforms trading strategies derived using a continuous action space (Pendharkar & Cusatis, 2018), this action space can provide a direct trading guide that a portfolio trader can follow in the real world. Furthermore, this discrete action space can derive a portfolio trading strategy with lower turnover relative to the strategies developed in previous studies. In previous studies, if a portfolio with a very large amount of capital is changed by a small amount in terms of portfolio weight, then the trader may pay significant transaction costs. In addition, the losses from these transaction costs can be very high because portfolio weight changes have no upper bound. In contrast, our action space has an upper bound for portfolio weight changes, and, thus, the issue of massive changes in portfolio weights and the resulting large losses from transaction costs do not arise. Our agent's action space has these advantages, and the only disadvantage of the fixed trading amount is similar to the restrictions on hedge funds that allow portfolio traders to only trade below a certain amount each day. Thus, our discrete action space is not too unrealistic to apply to real-world trading.

3.3. MDP modeling

With the state space and action space defined in the previous subsections, we can define the MDP model as follows. The financial market environment operates according to this model during the investment horizon. To define the transitions in the financial market environment (i.e., the system dynamics in the MDP model), we need to define the following parameters and equations:

$$w_t = (w_{t,0}, w_{t,1}, w_{t,2}, \dots, w_{t,I})^T, \tag{6}$$

$$w_t \cdot \vec{1} = w'_t \cdot \vec{1} = 1 \quad \forall t, \tag{7}$$

$$P'_t = P_{t-1} w_{t-1} \cdot \phi(k^c_t) \quad \forall t, \tag{8}$$

$$w'_t = \frac{w_{t-1} \odot \phi(k^c_t)}{w_{t-1} \cdot \phi(k^c_t)} \quad \forall t, \tag{9}$$

where w_t denotes the portfolio weight after the agent takes an action at the end of period t (Eq. (6)). Eq. (7) provides the constraint that the portfolio weight elements sum to one in all periods. Eqs. (8) and (9) represent the change in the portfolio value and the change in the proportions of the assets in the portfolio given the changes in the value of each asset in the portfolio, respectively. Here, $\odot$ represents the elementwise product of two vectors, and $\vec{1}$ is a vector of size *I+1* with all elements equal to one. $\phi(\cdot)$ is an operator that not only increases a vector's dimension by positioning zero as the first element but also adds it to the $\vec{1}$ vector (ϕ: $(e_1, e_2, \dots, e_I)^T \to (1, e_1+1, e_2+1, \dots, e_I+1)^T$).

Now, we can define the state changes after the agent takes an action as follows:

$$c_t = \frac{\delta}{P'_t}\left(c^-|S^-(a_t)| + c^+|S^+(a_t)|\right) \quad \forall t, \tag{10}$$

$$P_t = P'_t(1 - c_t) \quad \forall t, \tag{11}$$

$$\hat{w}'_t = \left(\hat{w}'_{t,0}, \hat{w}'_{t,1}, \hat{w}'_{t,2}, \dots, \hat{w}'_{t,I}\right)^T, \tag{12}$$

$$\hat{w}'_{t,i} = \begin{cases} w'_{t,i} - \frac{\delta}{P'_t} \; if i \in S^-(a_t), \\ w'_{t,i} + \frac{\delta}{P'_t} \; if i \in S^+(a_t), \quad \forall_{i=1\dots I}, \\ w'_{t,i} \quad otherwise \end{cases} \tag{13}$$

$$\hat{w}'_{t,0} = w'_{t,0} + \frac{\delta}{P'_t}\left((1 - c^-)|S^-(a_t)| - (1 + c^+)|S^+(a_t)|\right), \tag{14}$$

$$w_t = \frac{\hat{w}'_t}{\hat{w}'_t \cdot \vec{1}}, \tag{15}$$

After the agent takes an action, transaction costs arise, and the portfolio value is then decayed (Eqs. (10) and (11)). Here, $|S|$ is the size of set S. $\hat{w}'_{t,i}$ denotes the auxiliary weight of the portfolio that is needed to connect the change in the portfolio weights before and after the agent takes an action at the end of period t (Eq. (12)). The procedure by which the action selected by the agent is handled for trading in the financial environment is as follows. The auxiliary weight of an asset in the portfolio increases (or decreases) as a proportion of the trading size when buying (or selling) the asset. On the contrary, the auxiliary weights of the assets do not change when the agent holds the assets (Eq. (13)). As a result of selling assets, the proportion of cash increases by the proportion of the trading size discounted by the selling transaction cost rate. As a result of buying assets, the proportion of cash decreases by the proportion of the trading size multiplied by the buying transaction cost rate (Eq. (14)). To ensure that the sum of the portfolio weight elements equals one after the agent takes an action, a process for adjusting the auxiliary weights is required (Eq. (15)). In summary, the financial market environment transition is illustrated by Fig. 2.

Last, the reward in the MDP model should reflect the contribution of the agent's action to the portfolio return. This reward can be simply defined as the portfolio return. However, if the portfolio return is defined only as a reward, then different reward criteria can be given depending on the market trend. For example, when the market trend is sufficiently improving, then, no matter how poor the agent's action is, a positive reward is provided to the agent. In contrast, if the market trend is sufficiently negative, then,

no matter how helpful the agent's action is, a negative reward is provided to the agent. Thus, the reward must be defined as the rate of change in the portfolio value by which the market trend is removed. We therefore define the reward as the change in the portfolio value at the end of the next period relative to the static portfolio value (Eq. (16)). The static portfolio value is the next portfolio value when the agent takes no action at the end of the current period (Eq. (17)).

$$r_t = \frac{P'_{t+1} - P^s_{t+1}}{P^s_{t+1}}, \tag{16}$$

$$P^s_{t+1} = P'_t w'_t \cdot \phi(k^c_{t+1}), \tag{17}$$

4. Methodology

In this section, we introduce our proposed approach for deriving the portfolio trading strategy using DQL. In our action space, some issues may prohibit a DQL agent from deriving an intelligent trading strategy. We first explain how to resolve these issues by introducing some techniques and applying existing methodologies. Then, we describe our DQL algorithm with these techniques.

4.1. Mapping function

Some previous studies have discussed ways to handle infeasible actions in real-world problems with several constraints (Dulac et al., 2015; Xiong et al., 2018; Pham, Magistris, & Tachibana, 2018; Bhatia, Varakantham, & Kumar, 2019; Shah, Sinha, Varakantham, Perrault, & Tambe, 2019). Similarly, in our setting, some actions are infeasible in some states (e.g., the agent cannot buy assets because of a cash shortage or cannot sell assets because of a shortage of held assets). To handle these infeasible actions, we need to set their action values (i.e., Q-values) to be very low to mask these actions (Lanctot et al., 2017) and to convert the originally-selected infeasible action to appropriate feasible action. This conversion can be defined as a simple rule in which the agent selects the action with the largest Q-value among the remaining actions under consideration (Xiong et al., 2018). However, because understanding the similarity between actions is difficult for an RL agent, the agent takes the remaining action with the largest Q-value without any doubt even if this selected action is the opposite of the action originally determined by the agent's strategy. For example, if an agent's strategy selects the action of selling both *asset1* and *asset2*, but this action is infeasible owing to a lack of *asset2*, the action of buying both *asset1* and *asset2*, which is the remaining action with the largest Q-value, may be selected. This scenario leads to an unreasonable trading strategy.

Thus, in this study, we propose a mapping function that contains two mapping rules for two infeasible action cases. In each case, each rule defines ways to map infeasible actions to similar and valuable actions in the feasible action set. In the first case, in which the amount of cash is insufficient to take an action that involves buying assets (i.e., the cash shortage case), the rule first derives a similar action set of an originally infeasible action as a subset of the action space. The similar action set includes only actions that are feasible and that do not have any opposite direction of each asset compared to the directions under the original action. Then, the rule finds the most valuable action (i.e., the action with the largest Q-value) in the similar action set, and it maps the infeasible action to the identified feasible action. As illustrated in Fig. 3(a), for example, if the action of buying both *asset1* and *asset2* is infeasible owing to a cash shortage, the similar action set of the action includes all actions that do not involve selling direction for any assets (i.e., holding *asset1* and buying *asset2*, buying *asset1* and holding *asset2*, and holding both *asset1* and *asset2*). Among the three actions in the similar action set, the rule finds that the first action, holding *asset1* and buying *asset2*, is the most valuable action, and it maps the originally infeasible action to the action of holding *asset1* and buying *asset2*. In the second case, in which an action involving selling assets is infeasible because of the shortage of the assets (i.e., the asset shortage case), the mapping rule converts the originally infeasible action to the action holding any assets that cannot be sold owing to the shortage. Fig. 3(b) describes a simple example.

We provide the details of the two mapping rules in the mapping function shown by the pseudocode in Algorithm (1). In Algorithm (1), the last part (i.e., Lines (26)-(27)) of the mapping rule for the second case (`Rule2`) is necessary because, in the second case, the original action of selling assets that cannot be sold is converted into the action of holding the selling assets that cannot be sold. Thus, cash is no longer gained from selling assets, causing the first infeasible action case to arise. Furthermore, this part of the code can handle the special case in which an asset shortage and a cash shortage occur simultaneously.

Algorithm 1 Mapping function

1: s_t: state of the agent
2: a_t: infeasible action in state s_t
3: $Q(s_t, a_t)$: Q-value for state action pair (s_t, a_t)
4: **procedure** Maps_t, a_t
5: **if** asset shortage for action a_t in state s_t **then**
6: $a_{map} \leftarrow$ `Rule2`(s_t, a_t)
7: **els if** cash shortage for action a_t in state s_t **then**
8: $a_{map} \leftarrow$ `Rule1`(s_t, a_t)
9: **return** a_{map}
10: **procedure** Rule1s_t, a_t
11: $MAXQ \leftarrow -inf$
12: power set of buying asset indices in a_t: $\mathscr{P} = \{C, \ldots\}$
13: **for** each subset C in $\mathscr{P}$ **do**
14: replicate action $\hat{a}_t \leftarrow a_t$
15: **for** each asset j in C **do**
16: $\hat{a}_{t,j} \leftarrow 0$
17: **if** converted action $\hat{a}_t$ is feasible in state s_t **then**
18: **if** $Q(s_t, \hat{a}_t) > MAXQ$ **then**
19: $MAXQ \leftarrow Q(s_t, \hat{a}_t)$
20: $a_{best} \leftarrow \hat{a}_t$
21: **return** a_{best}
22: **procedure** Rule2s_t, a_t
23: **for** asset i = 1,2...I **do**
24: **if** action a_t to asset i in state s_t infeasible **then**
25: $a_{t,i} \leftarrow 0$
26: **if** converted action a_t is infeasible in state s_t **then**
27: $a_t \leftarrow$ `Rule1`(s_t, a_t)
28: **return** a_t

The mapping function can be considered a heuristic approach for solving a quadratic programming to handle RL in a constrained action space. This approach is called *OptLayer* and was proposed by previous studies (Pham et al., 2018; Bhatia et al., 2019). Although *OptLayer* can be applied to handle infeasible actions in this setting, it faces some challenges. First, in our setting, the optimization layer becomes the integer quadratic programming (IQP) model, as described in Eq. (18), and the solution of the IQP model suffers from the multiple-choice issue when the model has several tied optimal actions. Moreover, the nearest L_2 projection of the originally infeasible action, that is, the feasible action obtained from the optimization layer, may be irrational because the optimization problem described in Eq. (18) simply interprets the difference

between buying and selling as twice the difference between buying (or selling) and holding without considering the context of the portfolio trading problem. Thus, the obtained feasible action may have opposite direction of some assets compared to the originally infeasible action, which should be prevented in practice.

$$\begin{aligned} \underset{a_t}{\operatorname{argmin}} \|a_t - \tilde{a}_t\|_2^2, \quad & \text{s.t. } w'_{t,0} \geqslant \tfrac{\delta}{P'_t}\left((1+c^+)\sum_{i=1}^{I}[a_{t,i}]^+ - (1-c^-)\sum_{i=1}^{I}[-a_{t,i}]^+\right), \\ & w'_{t,i} \geqslant -\tfrac{\delta}{P'_t}a_{t,i} \quad \forall_{i=1\ldots I}, \\ & a_{t,i} \in \{-1,0,1\} \quad \forall_{i=1\ldots I}, \end{aligned} \tag{18}$$

where $\tilde{a}_t$ denotes the originally infeasible action in period t and operator $[\cdot]^+$ denotes a non-negative value (i.e., $\max\{\cdot, 0\}$).

Our proposed mapping function can overcome these limitations of the *OptLayer* approach. In fact, the mapping rules defined in the mapping function can be interpreted as finding the optimal solution of variants of the IQP model for the *OptLayer* approach. The integer programming model, described in Eq. (19), is for the cash shortage case, and the IQP model, described in Eq. (20), is for the asset shortage case.

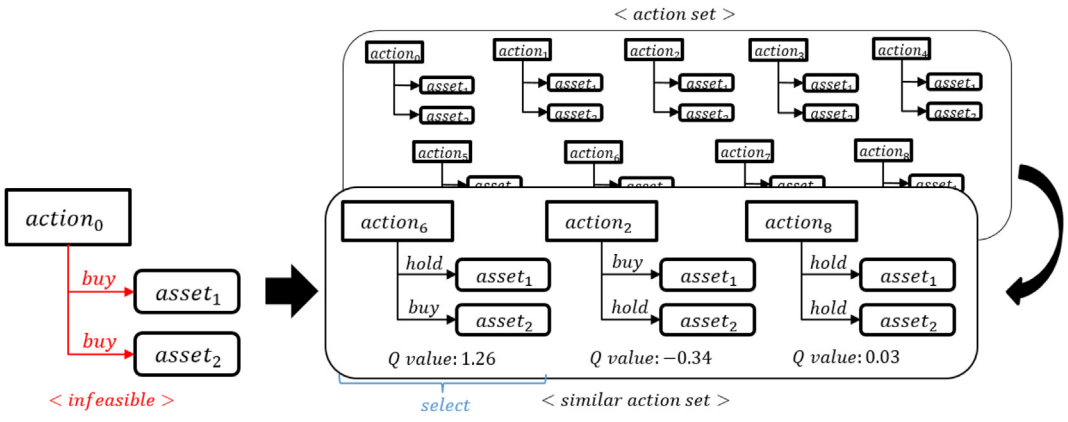
$$\begin{aligned} \underset{a_t}{\operatorname{argmax}} Q(s_t, a_t), \quad & \text{s.t. } w'_{t,0} \geqslant \tfrac{\delta}{P'_t}\left((1+c^+)\sum_{i=1}^{I}[a_{t,i}]^+ - (1-c^-)\sum_{i=1}^{I}[-a_{t,i}]^+\right), \\ & |a_{t,i} - \tilde{a}_{t,i}| < 2 \quad \forall_{i=1\ldots I}, \\ & a_{t,i} \in \{-1,0,1\} \quad \forall_{i=1\ldots I}, \end{aligned} \tag{19}$$

$$\begin{aligned} \underset{a_t}{\operatorname{argmin}} \|a_t - \tilde{a}_t\|_2^2, \quad & \text{s.t. } w'_{t,i} \geqslant -\tfrac{\delta}{P'_t}a_{t,i} \quad \forall_{i=1\ldots I}, \\ & a_{t,i} \in \{-1,0,1\} \quad \forall_{i=1\ldots I}, \end{aligned} \tag{20}$$

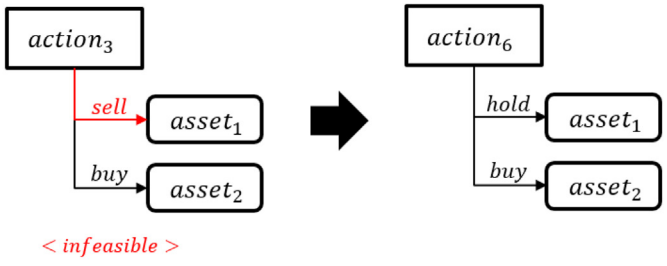
The first variant (Eq. (19)) can identify the single action with the largest Q-value in the similar action set which can be defined based on two constraints. In addition, the second variant (Eq. (20)) can find exactly one action corresponding to an originally infeasible action in the asset shortage case. The constraint filters all infeasible actions involving selling assets facing a shortage, and, thus, the optimization model can identify the nearest action to the originally infeasible action among the remaining filtered actions. In this case, the nearest feasible action is the action in which all assets that are sold under the originally infeasible action but that cannot be sold owing to a shortage are held instead. Thus, the two optimization models face no multiple-choice issue.

4.2. DQN algorithm

We optimize the multi-asset portfolio trading strategy by applying the DQN algorithm. DQN is the primary algorithm for DQL. Mnih et al. (2013) developed the DQN algorithm, and Mnih et al. (2015) later introduced additional techniques and completed the algorithm. The base algorithm for DQN, Q-learning, is value-based RL, which is a method that approximates an action value (i.e., a Q-value) in each state. Further, Q-learning is a model-free method such that even if the agent does not have knowledge of the environment, the agent can develop a policy using repeated experience by exploring. In addition, Q-learning is an off-policy algorithm, that is, the action policy for selecting the agent's action is not the same as the update policy for selecting an action on the target value. An algorithm based on Q-learning that approximates the Q-function using DNN is the basis of DQN (Mnih et al., 2013). To prevent DNN from learning only through the experience of a specific situation, experience replay was introduced to sample a



(a)



(b)

Fig. 3. Illustration of mapping rules for a two-asset portfolio example: (a) the cash shortage case and (b) the asset shortage case.

general experience batch from memory. Additionally, the DQN algorithm used two separate networks: a Q-network that approximates the Q-function and a target network that approximates the target value needed for the Q-network updated to follow a fixed target (Mnih et al., 2015). Based on this algorithm, we introduce several techniques to support the derivation of an intelligent trading strategy.

The existing DQN algorithm updates the Q-network with experience by allowing the agent to take only one action in each stage. Because the agent has no information about the environment, only one action is taken before proceeding to the next state. Thus, it is impossible to take multiple actions in the existing DQN. However, for this problem, we use historical technical indicator data of the assets in the portfolio as training data. Thus, our agent can take multiple actions in one state in each stage and can observe all of its experiences based on those actions. To utilize this advantage, we introduce a technique that simulates all feasible actions in one state at each stage and updates the trading strategy by using the resulting experiences from conducting these simulations.

Motivated by Tan, Liu, and Qiu (2009), we utilize a simulation technique that takes all feasible actions virtually to force the agent to learn about many experiences efficiently to derive a fully searched multi-asset trading strategy. Thus, this technique can relax the data shortage issue that arises when deriving a multi-asset trading strategy. Although simulating all feasible actions can result in a huge computational time, using multi-core parallel computing can prevent this computational time from greatly increasing. Moreover, even if the agent takes multiple actions in the current state, the next state only depends on the action selected by the action policy (epsilon-greedy) with the mapping rule. The application of this technique requires a change in the data structure of the elements in replay memory to store a list of experiences in a state. The concepts related to this technique are illustrated in Fig. 4. In this figure, a_t^j means that the $j-th$ action of the agent is taken at the end of period t. r_t^j is the reward obtained by taking action a_t^j, and s_{t+1}^j is the next state that results from taking action a_t^j.

In DQN, a multiple output neural network is commonly adopted as the Q-network structure. In this network structure, the input of the neural network is the state, and the output is the Q-value of each action. Using the above technique, we can approximate the Q-value of all feasible actions by updating this multiple output Q-network in parallel with the experience list. To maintain Q-values of infeasible actions, the current Q-value of an infeasible state-action pair is assigned to the target value of the Q-network output of the corresponding infeasible action to set a temporal difference error of zero. Furthermore, as in DQN, several experience lists are sampled from replay memory, and the Q-network is updated using the experience list batch. A detailed description of the process for updating the Q-network is shown in Fig. 5.

In addition, to apply RL, learning episodes must be defined for the agent to explore and experience the environment. Rather than defining all of the training data, which cover several years, as one episode, we divide the training data into several episodes. If we define a much longer training episode than the investment horizon of the test data that will be used to test the trading strategy, this difference in the lengths of the training and test data can produce negative results. For example, in our experiment, the training and testing processes begin with the same portfolio weights. In this case, the farther the agent is from the beginning of the long training episode, the farther the agent is from the initial portfolio weights. Thus, it is difficult for the agent to utilize the critical experience obtained from the latter half of the long episode in the early testing process. Therefore, we divide the training data into sets with the same length as the investment horizon of the test data

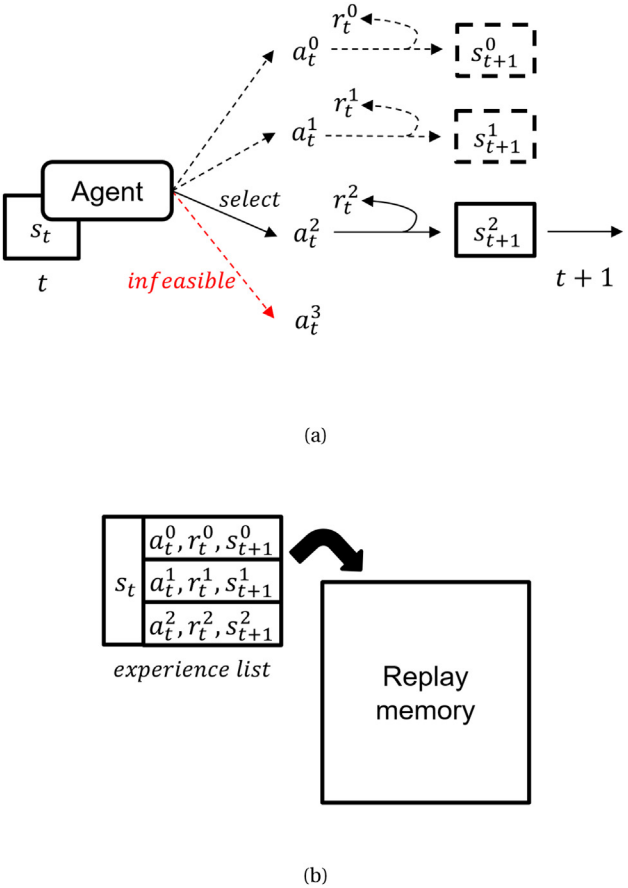


Fig. 4. (a) Simulating feasible actions, (b) Data structure for the experience list.

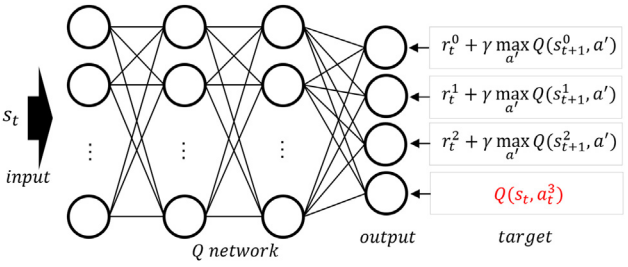


Fig. 5. Updating a multiple output Q-network using an experience list.

(i.e., one year, as the investment horizon of the test data is one year in our experiment). Thus, the criteria for dividing the training data are defined in yearly units so that the episodes do not overlap (e.g., *episode1* contains data from 2016, and *episode2* contains data from 2015). In each training epoch, the agent explores and learns in an episode sampled from the training data.

It is well known that more recent historical data have more ability to predict future data than less recent historical data have. Thus, it is reasonable to assign higher sampling probabilities to episodes that are closer to the test data period (Jiang et al., 2017). We use a truncated geometric distribution to assign higher sampling probabilities to episodes that are closer to the test period. This truncated geometric sampling distribution is expressed in Eq. (21). Here, y is the year of the episode, y_v is the year of the test data, and N is the number of total training episodes. β is a parameter for this sampling distribution that ranges from zero to one. If

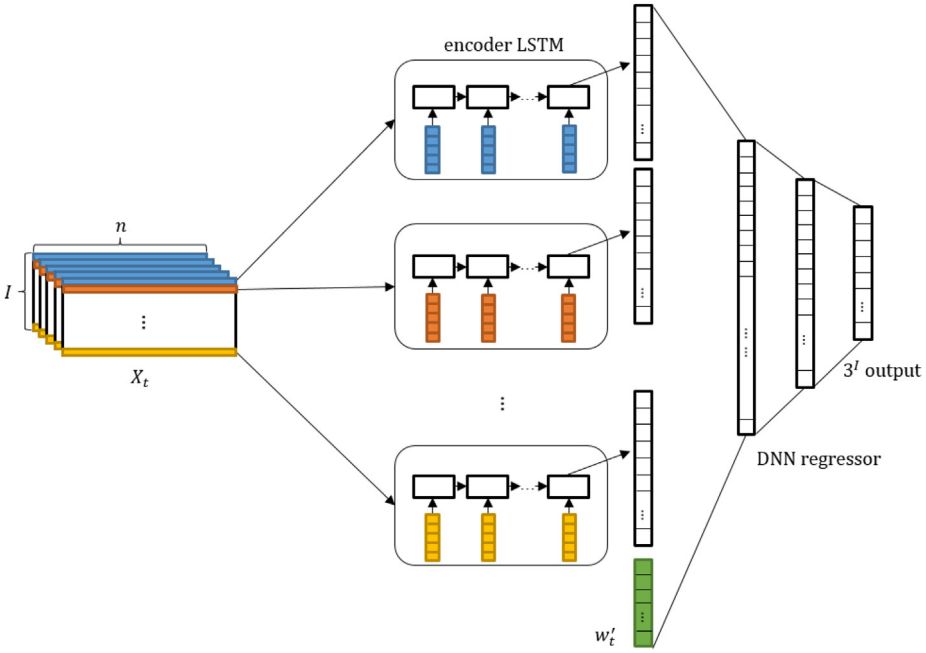


Fig. 6. Q-network structure.

this parameter is closer to one, episodes closer to the test period are sampled more frequently.

$$g_\beta(y) = \frac{\beta(1-\beta)^{y_v - y - 1}}{1-(1-\beta)^N}, \tag{21}$$

To implement DQN, we need to model the neural network structure for approximating the Q-function of an agent's state and action. We construct a hybrid encoder long short term memory (LSTM)-DNN neural network that enables us to approximate the Q-value of an agent's action in our predefined state and action space. First, we train the LSTM autoencoder, an unsupervised learning method for compressing sequence data, to identify latent variables in the historical sequence of predefined technical indicators of assets in the portfolio (Srivastava, Mansimov, & Salakhudinov, 2015). Then, the autoencoder is fitted to the historical pattern of technical indicators of assets, the decoder is removed, and the encoder remains as a standalone model that transforms the technical indicator sequences for assets into the low dimensional latent variables. Each asset in the portfolio shares the same encoder LSTM for the sequence encoding procedure because it is known that a single deep learning model is more effective for learning the feature patterns of different assets than multiple deep learning models for learning individual assets are Sirignano and Cout (2018). Then, the encoded outputs for each asset are concatenated to create the intermediate output, and this intermediate output is again combined with the current portfolio weights to serve as the input to the DNN. Through the layers of this DNN, we can obtain the Q-value of the agent's actions. Because these DNN layers extract meaningful features through nonlinear mapping using a multi-layer neural network and conduct a regression for the Q-value, we refer to these layers as the DNN regressor. The overall Q-network structure is as shown in Fig. 6. In summary, the overall DQN algorithm for our approach to deriving the portfolio trading strategy is given by Algorithm (2). In offline learning, we use this algorithm to build an initial trading strategy fitted to historical data and, through online learning, the trading strategy adapts further by updating based on the daily observed data in the trading process.

Algorithm 2 DQN algorithm for portfolio trading

Initialize replay memory D
Initialize weights of Q-network θ randomly
Initialize weights of target network $\theta' \leftarrow \theta$
for episode is sampled by sampling distribution $y \leftarrow g_\beta(\cdot)$ **do**
  Initialize state s_0
  for period t=0...T in episode y **do**
    With probability ϵ select random $a_t \in F(s_t)$
    otherwise, $a_t = \begin{cases} \underset{a}{\text{argmax}} Q(s_t, a; \theta) \text{ if } \underset{a}{\text{argmax}} Q(s_t, a; \theta) \in F(s_t), \\ \text{Map}\left(s_t, \underset{a}{\text{argmax}} Q(s_t, a; \theta)\right) \quad o/w \end{cases}$
    Take action a_t and then observe reward r_t and the next state s_{t+1}
    Simulate all actions $a \in F(s_t)$, then observe experience list L
    Store L in replay memory D
    Sample random batch of experience list K from D
    (s_t, a_t, r_t, s_{t+1}) is an element of the experience list in batch, update Q-network from current prediction $Q(s_t, a_t; \theta)$ to target
    $z_t = \begin{cases} r_t + \gamma \underset{a'}{\max} \hat{Q}(s_{t+1}, a'; \theta') \text{ if } \underset{a'}{\text{argmax}} \hat{Q}(s_{t+1}, a'; \theta') \in F(s_{t+1}), \\ r_t + \gamma \underset{a'}{\max} \hat{Q}\left(s_{t+1}, \text{Map}\left(s_{t+1}, \underset{a'}{\text{argmax}} \hat{Q}(s_{t+1}, a'; \theta')\right); \theta'\right) \quad o/w \end{cases}$
    Update θ by minimizing the loss: $L(\theta) = \frac{1}{|K|} \sum_{L \in K} \sum_{j \in L} \left(z_j - Q(s_j, a_j; \theta)\right)^2$
  $\theta' \leftarrow \theta$

5. Experimental results

In this section, we demonstrate that the DQN strategy (i.e., the trading strategy derived using our proposed DQN algorithm for portfolio trading) can outperform other strategies in real-world trading. We conduct a trading simulation for two different portfolio cases using both our DQN strategy and traditional trading strategies as benchmarks, and we verify that the DQN strategy is relatively superior to the other benchmark strategies based on several common performance measures.

5.1. Performance measures

Similar to previous studies described in Section 2, we use four different output performance measures to evaluate trading strategies. These measures are sufficient to evaluate the results of backtesting because they can assess a trading strategy's cumulative return, average return, turnover rate, and risks. The first measure is the cumulative return based on the change in portfolio value at the end of the investment horizon relative to the initial portfolio value, as defined as Eq. (22):

$$CR = \frac{P_{t_f} - P_0}{P_0} \times 100(\%), \tag{22}$$

where t_f is the final date of the investment horizon and P_0 is the initial portfolio value.

The second measure is the Sharpe ratio, as defined in Eq. (23):

$$SR = \frac{E\left[\rho_t - \rho_f\right]}{std(\rho_t)} \times \sqrt{252}, \tag{23}$$

where $std(\rho_t)$ is the standard deviation of the daily return rate, ρ_f is the daily risk-free rate (assumed to be 0.01%), and the annualization term (the square root of the number of annual trading days) is multiplied. This ratio is a common measure of the risk-adjusted return, and it is used to evaluate not only how high the risk premium is but also how small the variation in the return rate is.

The third measure is the Sterling ratio, as defined in Eq. (24):

$$SterR = \frac{E\left[\rho_t - \rho_f\right]}{\sqrt{\frac{1}{t_f}\sum_{t=1}^{t_f} \min\{\rho_t, 0\}^2}} \times \sqrt{252}. \tag{24}$$

As in the case of the Sharpe ratio, the annualization term is multiplied. This ratio is used to assess the downside risk-adjusted return, as discussed in previous studies (Moody & Saffell, 2001; Almahdi & Yang, 2017). This measure evaluates not only how high the risk premium is but also how small the downside risk is.

For the last measure, we use the customized average turnover rate defined as in Eq. (25):

$$AT = \frac{1}{2t_f}\sum_{t=1}^{t_f}\sum_{i=1}^{I}|\hat{w}'_{t,i} - w'_{t,i}| \times 100(\%). \tag{25}$$

The average turnover rate measures the average rate of change of the portfolio weight vector during the investment horizon. We do not have to consider changes in the proportion of cash, so we customize this measure by excluding the change in the weight on cash before and after the agent takes an action. This rate can therefore evaluate the change in the proportions of asset investments. Considering transaction costs, this measure should be low so that the trading strategy is more applicable in the real world.

5.2. Data summary

We experiment with two different three-asset portfolios. Similar to a recent study (Almahdi & Yang, 2017), the first portfolio (US-ETF) consists of three exchange traded funds (ETFs) in the US market that track the S&P500 index, the Russell 1000 index, and the Russell Microcap index. In addition, to validate our approach's robustness to various assets, we also test it using a Korean portfolio (KOR-IDX) consisting of the KOSPI 100 index, the KOSPI midcap index, and the KOSPI microcap index. Because the corresponding ETFs for these indices have a short history and are not yet mature, we assume that the indices themselves are tradable.

Ideally, the component assets in the US-ETF and KOR-IDX should be mutually independent to provide diversification benefits and ensure the validity of portfolio trading. However, in practice, financial assets are rarely mutually independent, and even stocks and bonds are often highly correlated (Aslanidis & Christiansen, 2014; Baur & Lucey, 2009). Nevertheless, we can observe that the correlations among the assets in our second test portfolio (KOR-IDX) are at an acceptable level. More information about these test portfolios is provided in Table 3.

In our experiments, we employ ETFs or indices as the component assets of the test portfolios because we cannot consider all possible combinations of individual stocks. A ETF or index-based asset can represent the characteristics of a cluster of several stocks well, allowing us to avoid issue around choosing stocks for test assets. Accordingly, our experimental results can be representative. This approach is commonly used in related studies (Almahdi & Yang, 2017; Pendharkar & Cusatis, 2018; Jeong & Kim, 2019).

Although we only consider three investable assets in a portfolio, increasing the number of assets is merely a matter of applying more computational effort because our proposed model can be applied to a multiple asset portfolio of any size. Additionally, using three component assets is a simple yet effective approach in the practice of portfolio trading with a well-structured hierarchy of investable assets. Portfolio traders typically build and perceive a hierarchy of investable assets. More specifically, all assets can be categorized into three asset classes. For example, stocks can be split into the classes of microcap, midcap, and large-cap stocks. Thus, training an intelligent agent that can identify an optimal portfolio trading strategy for a three-asset portfolio has significant practical value.

We obtain data on the three US ETFs from *Yahoo Finance* and data on the Korean indices from *Investing.com*. Using these websites, we can extract data for the US ETFs from 2010 to 2019 and data for the Korean indices from 2012 to 2017. Table 4 summarizes the information about the training and test periods for each portfolio.

5.3. Experiment setting

Through several rounds of tuning, we derive appropriate hyper-parameters. In particular, the time window size(n) is the most important hyper-parameter, and, from the candidate values (5,20,60,120), we adopted a time window size of 20. This time window size and the other tuned hyper-parameters are summarized in Table 5.

Table 3
Test portfolios.

Assets	Portfolio	
	US Portfolio (US-ETF)	Korean Portfolio (KOR-IDX)
Asset 1	SPDR S&P 500 ETF (SPY)[1]	KOSPI 100 Index (KSLAR)
Asset 2	iShares Russell 1000 Value ETF (IWD)[2]	Midcap KOSPI Index (KSMID)
Asset 3	iShares Microcap ETF (IWC)[3]	Microcap KOSPI Index (KSMIC)

[1] ETF tracks the S&P500 index.
[2] ETF tracks the Russell 1000 (mid- and large-cap US stocks) index.
[3] ETF tracks the Russell microcap index.

Table 4
Timing of the data period.

Portfolio	Period	
	Training period	Test period
US Portfolio	2010 ~ 2016	2017
Korean Portfolio	2012 ~ 2016	2017

Table 5
summary of hyper-parameters.

hyper-parameter	value	hyper-parameter	value
time window size (n)	20	replay memory size	2000
learning rate (α)	1e−7	number of epochs	500
distribution parameter (β)	0.3	discount factor (γ)	0.9
DNN input dimension	64	batch size	32
DNN layer	2	encoder LSTM layer	1
DNN 1st layer dimension	64	hidden dim of encoder LSTM	128
DNN 2nd layer dimension	32	output dim of encoder LSTM	20

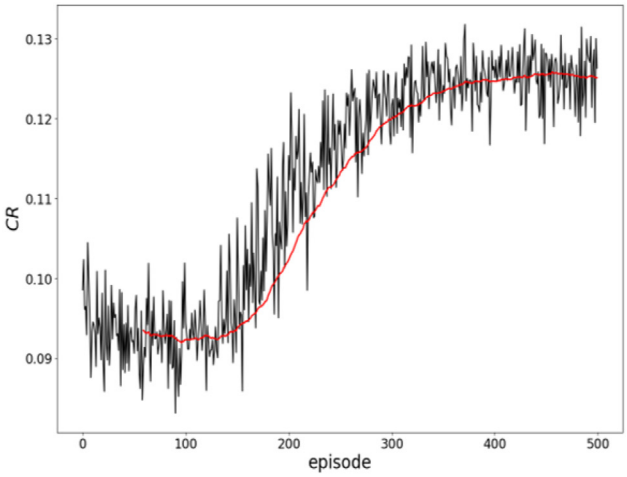
In the experiment, we also need to set trading parameters, such as the initial portfolio value and the trading size. We set the initial portfolio value as one million in both portfolio cases (e.g., 1M *USD* for the US portfolio and 1M *KRW* for the Korean portfolio). Similarly, we set the trading size as ten thousand in both portfolio cases (e.g., 10K *USD* trading size for the US portfolio case and 10K *KRW* for the Korean portfolio case). We set the transaction cost rate for buying and selling in both the US and Korean markets as 0.25%. In both cases, the initial portfolio is set up as an equally weighted portfolio in which every asset and cash has the same proportion. All algorithms are implemented in *Python* with *tensorflow* and run on a PC that contains an *Intel* six-core *2.9GHz CPU* with *16GB RAM* and *NVIDIA GeForce GTX1060 GPU*.

5.4. Benchmark strategy

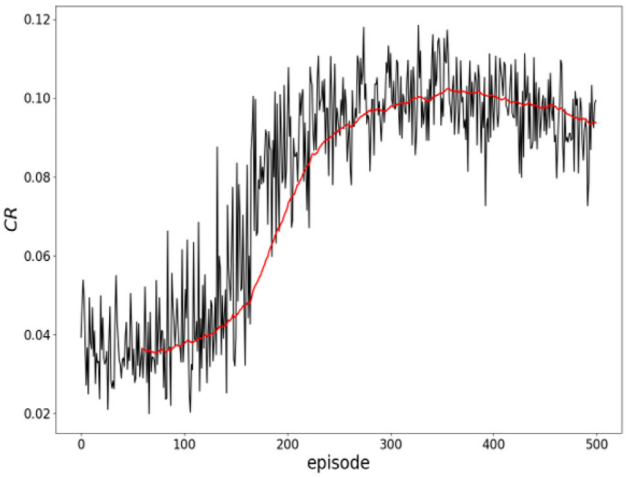
To evaluate our DQN strategy, we compare it to some traditional portfolio trading strategies. The first strategy is a buy-and-hold strategy (*B&H*) that does not take any action but rather holds the initial portfolio until the end of the investment horizon. The second strategy is a randomly selected strategy (*RN*) that takes an action within the feasible action space randomly in each state. The third strategy is a momentum strategy (*MO*). This strategy buys assets whose values increased in the previous period and sells assets whose values decreased in the previous period. However, if it cannot buy all the assets whose values increased, it gives buying priority to the assets whose values increased more. If it is unable to sell assets whose values decreased, it simply holds the assets. The last strategy is a reversion strategy (*RV*), which is the opposite of the momentum strategy. This strategy sells assets whose values increased in the previous period and buys assets whose values decreased in the previous period. However, if it cannot buy all of the assets whose values decreased, it gives buying priority to the assets whose values decreased more. If it is unable to sell the assets whose values increased, it simply holds the assets.

5.5. Result

We derive a trading strategy for both portfolio cases using DQN. For both cases, we identify the increase in the cumulative return over the investment horizon of the test period as episode learning continues. Fig. 7 shows the trend in the cumulative return performance over the learning episodes in both cases.



(a)



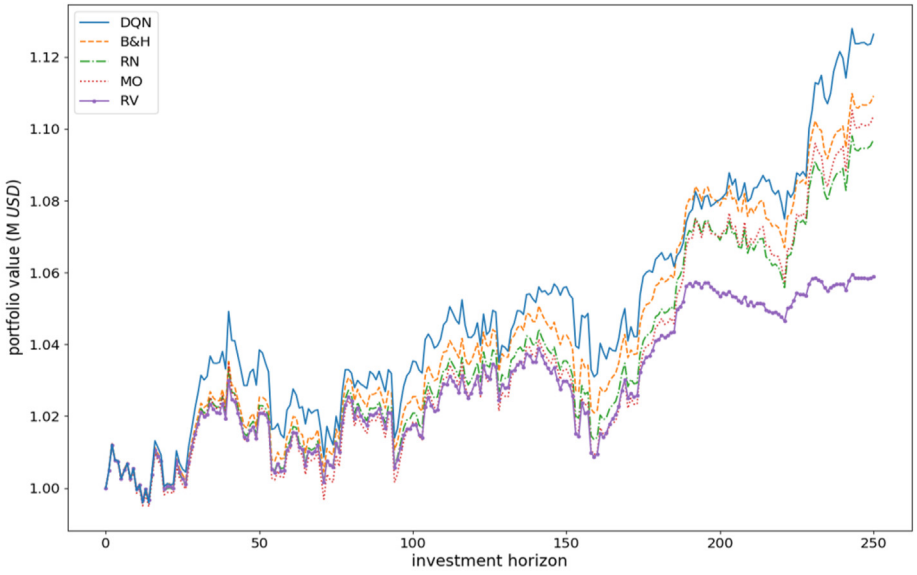
(b)

Fig. 7. Cumulative return rate as the learning episode continues (a) US portfolio, (b) Korean portfolio.

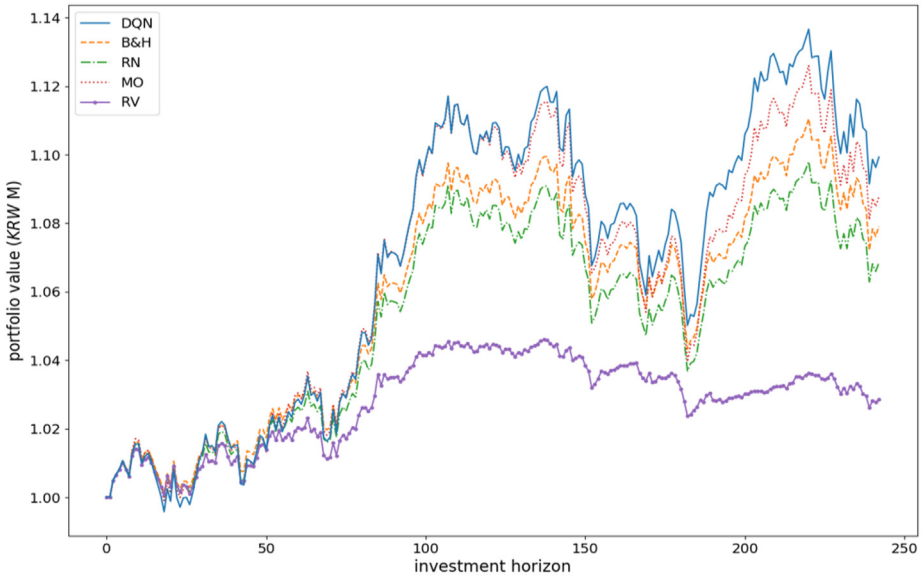
Table 6 shows the number of changes in the phase of the trading direction (i.e., the number of changes from selling to buying or vice versa, excluding holding) before and after applying the mapping function. Through the experimental results, we identify that when the mapping function is applied to the DQN strategy, the number of changes in the phase of the trading direction decreases relative to when it is not applied. By decreasing the number of changes in the phase of the trading direction, the trading strategy's cumulative return increases by 6.68% relative to the strategy that does not apply the mapping function in the US portfolio case. Likewise, in the Korean portfolio case, the cumulative return of the trading strategy increases by 10.83% relative to the strategy that does not apply the mapping function.

Table 6
Number of changes in the phase of the trading direction for the DQN strategy without and with the mapping function for the two test portfolios.

mapping function	US portfolio			Korean portfolio		
	SPY	IWD	IWC	KSLAR	KSMID	KSMIC
without	83	107	64	74	128	63
with	54	40	37	42	68	40
difference	−34.9%	−62.6%	−42.1%	−43.2%	−46.8%	−36.5%



(a)



(b)

Fig. 8. Comparative portfolio value results for the DQN and benchmark strategies for (a) the US portfolio, (b) the Korean portfolio.

Using the mapping function, the two rules usually convert infeasible actions for assets from buying or selling to holding. As a result, the DQN strategy with the mapping function rebalances the portfolio less frequently than the DQN strategy without the mapping function does. That is, the strategy without the mapping function can induce frequent changes in direction for each asset. A trading strategy with frequent changes in direction is unreasonable because these changes deteriorate trading performance by incur-

Table 7
Output performance measure values for our DQN strategy and the benchmark strategies for the two test portfolios.

	US portfolio				Korean portfolio			
strategy	Cumulative return	Sharpe ratio	Sterling ratio	Average turnover	Cumulative return	Sharpe ratio	Sterling ratio	Average turnover
Buy&Hold	10.921%	1.308	1.963	0.000%	7.913%	0.890	1.234	**0.000%**
Random*	9.446%	1.143	1.501	1.029%	7.358%	0.381	0.919	1.032%
Momentum	10.372%	1.114	1.653	1.368%	8.773%	0.840	1.165	1.233%
Reversion	5.891%	0.642	0.927	1.404%	2.855%	0.147	0.213	1.370%
DQN	**12.634%**	**1.382**	**2.071**	**0.954%**	**9.933%**	**0.946**	**1.326**	**0.989%**

* The performance of the Random trading strategy is the average over 30 observations.

ring meaningless transaction costs. This point supports the necessity of the mapping function to derive a trading strategy with better performance.

Fig. 8 shows the portfolio value trend when applying the DQN strategy and the benchmark strategies in the US and Korean portfolio cases. In the US portfolio case, we observe that the DQN strategy outperforms the benchmark strategies for most of the test period. The final portfolio value of the DQN strategy is 15.69% higher than that of the B&H strategy, 33.74% higher than that of the RN strategy, 21.81% higher than that of the MO strategy, and 114.47% higher than that of the RV strategy. Likewise, in the Korean portfolio case, we observe that the DQN strategy outperforms the benchmark strategies for most of the test period. The final portfolio value of the DQN strategy is 25.52% higher than that of the B&H strategy, 34.99% higher than that of the RN strategy, 13.22% higher than that of the MO strategy, and 247.91% higher than that of the RV strategy.

Table 7 summarizes the output performance measure results when using DQN and the benchmark strategies in both portfolio cases. This table shows that the DQN strategy has the best cumulative return, Sharpe ratio, and Sterling ratio performances for the US portfolio, and it also has the second-lowest turnover rate after the B&H strategy, which has no turnover rate. In the Korean portfolio case, the DQN strategy also has the best cumulative return, Sharpe ratio, and Sterling ratio performances. Moreover, the DQN strategy has the second-lowest turnover rate after B&H strategy. Given that the B&H strategy does not incur any transaction costs during the investment horizon, it is a remarkable achievement that the DQN strategy outperforms the B&H strategy in terms of the cumulative return, Sharpe ratio, and Sterling ratio.

Based on the results, the DQN strategy obtained using our proposed approach is superior to the benchmark strategies because the DQN strategy can consider much more various market features, relative to the simple, rule-based benchmark strategies. For example, the momentum and reversion trading strategies strictly follow predefined rules by considering only the prices of assets. Thus, these rule-based strategies respond to market variations inflexibly. Conversely, the DQN strategy can consider the history of several meaningful technical indicators.

To verify the robustness of the sample horizon of the proposed method, we conduct backtesting experiments for the US portfolio using by comparing the proposed strategy with the benchmark strategies for various other investment horizon years. These results of the sensitivity analysis for the investment horizon are summarized in Appendix A. These results demonstrate the robustness of our proposed method under different investment horizons. We conduct this verification using sufficiently many different sampling periods of investment horizon relative to previous studies, and, thus, we can argue that the verification in this study is sufficiently valid.

6. Conclusion

The main contribution of our study is that we apply the DQN algorithm to derive a portfolio trading strategy on the practical action space. However, applying DQN to portfolio trading has some challenges. To overcome these challenges, we devise a DQL model for trading and several techniques. First, we introduce a mapping function to handle infeasible actions and derive a reasonable trading strategy. Trading strategies derived from RL agents may be unreasonable for real-world applications. Thus, we apply a domain knowledge rule to develop a trading strategy with an infeasible action mapping constraint. As a result, this function works well, and we can derive a reasonable trading strategy. Second, we design a DQL agent and a Q-network to considering the features of multiple assets, and we derive a multi-asset trading strategy in the practical action space that determines the assets' trading directions by overcoming the dimensionality problem. Third, we relax the data shortage issue that makes it difficult to derive well-fitted multi-asset trading strategies by introducing a technique that simulates all feasible actions and then updates the trading strategy based on the experiences of these simulated actions.

The experimental results show that our proposed DQN strategy is superior to benchmark strategies. Based on the results for the cumulative return, Sharpe ratio, and Sterling ratio, the DQN strategy is more profitable with lower risk than other benchmark strategies. In addition, based on the results for the average turnover rate, the DQN strategy is more suitable for application in the real-world than the benchmark strategies are.

This study is based on the belief in technical analysis that the asset prices can be predicted using the history of assets' technical indicators. Even if this technical analysis lacks a theoretical basis, many previous studies have shown that an asset's past prices can be used to predict its future returns to some degree (Lo & Mackinlay, 1988; Lo & Mackinlay, 2011) and have identified the efficacy of technical analysis (Lo, Mamaysky, & Wang, 2000). In this context, our approach to producing a learning-based portfolio trading strategy can consider technical indicators efficiently and, thus, can outperform in this problem.

An RL agent observes the precise phenomena around its model-free and theory-free environment and builds its own model and theory that may not necessarily be understandable to humans. Comparably, an RL agent in the game of Go (Silver et al., 2017) has astonishingly outperformed models and theories of the game that have been built for thousand years. Likewise, our proposed approach also constructs a trading model that is not interpretable by humans, but the trading strategy derived by this model can outperform traditional rule-based trading strategies that have been constructed for some decades. In the same line of reasoning, our proposed model can reflect the macroeconomic factors well, which can vary depending on the time and region. Historically, there have been several events that cause the significant transition of the financial market (for example, the Dodd-Frank regulation (Cumming, Dai, & Johan, 2017)). The testing periods in our experiments contain several big events affecting the macroeconomic factors, such as interest rate changes and the US-China trade war, and the experimental results show that our model can adapt to the temporal changes of the factors well by employing the commonly used technical indicators and two techniques (i.e., episode sampling and online learning). In addition, our experimental results describe that the trading strategies derived by the proposed model can have superior performance in both US and Korean markets, which have different trading rules (Cumming, Johan, & Li, 2011). Consequentially, we can say that our model can be robust to the differences in trading rules across countries and over time.

However, our proposed methodology still faces scalability limits arising from the dimensionality problem if the number of assets in the portfolio is very large. Furthermore, in our study, the reward of the MDP model is optimized only for returns and not for risk. Nevertheless, the contributions of this study are still valuable because it uses the novel techniques to express the practical applicability of the portfolio trading strategy. In response to the limitations of this study, we will try to devise a method to resolve them in future research.

Conflict of interest

The authors declare that they have no conflict of interests regarding the publication of this paper.

CRediT authorship contribution statement

Hyungjun Park: Conceptualization, Methodology, Formal analysis, Software, Writing - original draft. **Min Kyu Sim:** Validation, Formal analysis, Writing - review & editing. **Dong Gu Choi:** Supervision, Funding acquisition, Project administration, Validation, Writing - review & editing.

Acknowledgement

This paper was supported by Korea Institute for Advancement of Technology (KIAT) grant funded by the Korea Government (MOTIE) (P0008691, The Competency Development Program for Industry Specialist).

Appendix A. Sensitivity analysis for different investment horizons

To verify the robustness of the sample horizon of the proposed method, we conduct backtesting experiments for the US portfolio using the proposed method by comparing the DQN strategy with the benchmark strategies under three additional investment horizons: 2016, 2018, and 2019. Table A.8 summarizes the information about the training and test periods for three different investment horizons in the case of the US portfolio.

We derive a trading strategy for the US portfolio under three test periods using our proposed approach. For these periods, we identify the increase in the cumulative return during the investment horizon of the test period as episode learning continues. Fig. A.9 shows the cumulative return performance trend over the learning episodes in these periods.

Fig. A.10 shows the portfolio value trend when the DQN strategy and the benchmark strategies are applied in the US portfolio case for the three periods. With a test period of 2016, we observe that the DQN strategy outperforms the benchmark strategies for most of the test period. The final portfolio value of the DQN strategy is 49.86% higher than that of the B&H strategy, 86.99% higher than that of the RN strategy, 81.25% higher than that of the MO strategy, and 169.21% higher than that of the RV strategy. On the other hand, with a test period of 2018, we observe that the DQN strategy does not outperform the benchmark strategies for most of the test period. However, the DQN strategy outperforms at the

Table A.8
Timing of the data period.

Portfolio	Period	
	Training period	Test period
US Portfolio in 2016	2010 ∼ 2015	2016
US Portfolio in 2018	2010 ∼ 2017	2018
US Portfolio in 2019	2010 ∼ 2018	2019

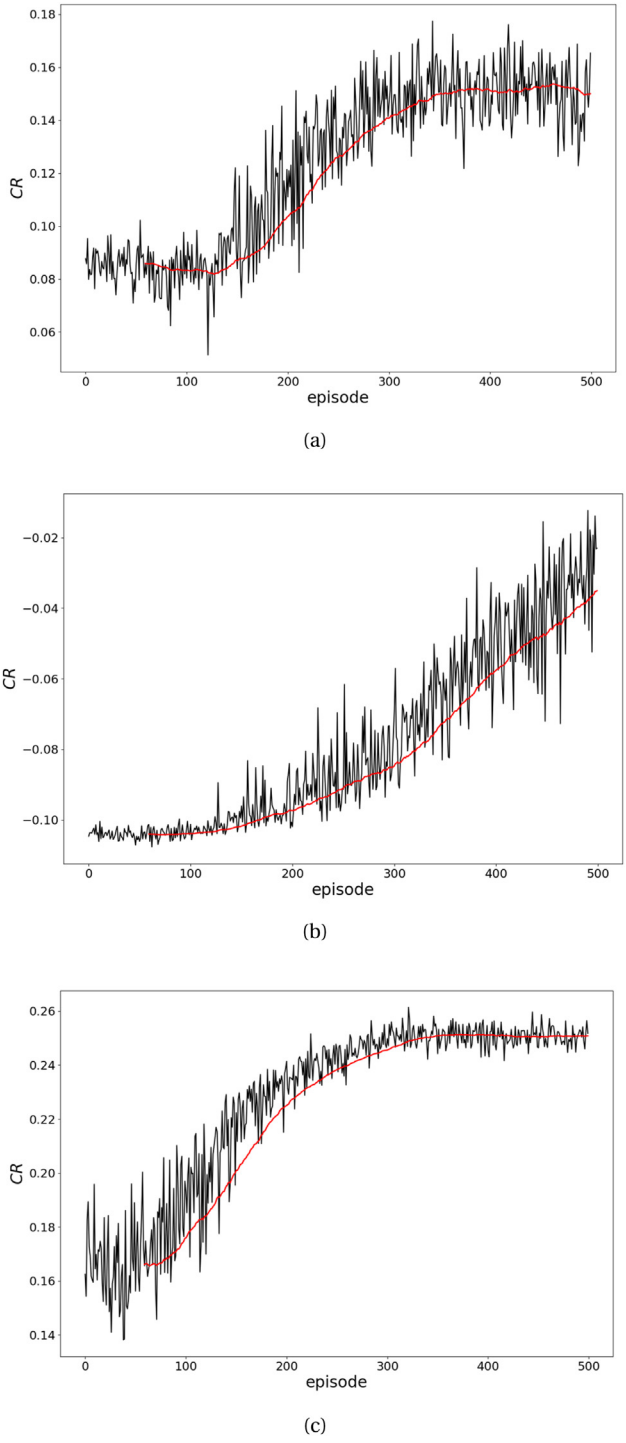
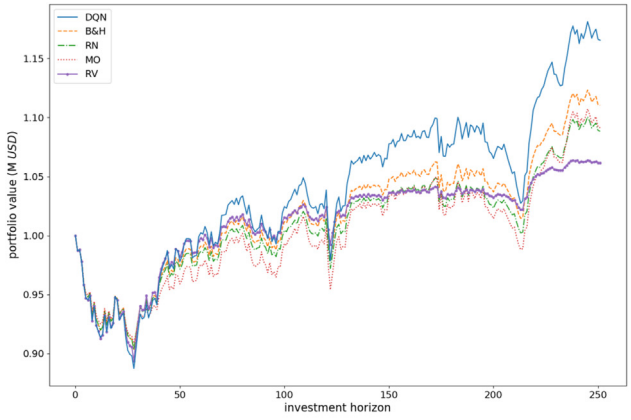


Fig. A.9. Cumulative return rate as the learning episode continues for the US portfolio in (a) 2016, (b) 2018, and (c) 2019.

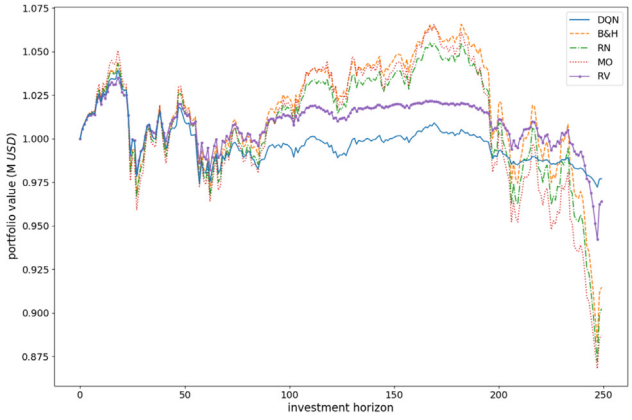
Table A.9
Output performance measures for the DQN strategy and the benchmark strategies for the US portfolio in the three test periods.

strategy	US portfolio in 2016				US portfolio in 2018				US portfolio in 2019			
	Cumulative return	Sharpe ratio	Sterling ratio	Average turnover	Cumulative return	Sharpe ratio[a]	Sterling ratio[a]	Average turnover	Cumulative return	Sharpe ratio	Sterling ratio	Average turnover
Buy&Hold	11.036%	0.791	1.117	**0.000%**	−8.524%	−5.07e−5	−3.97e−5	**0.000%**	18.484%	1.528	2.179	**0.000%**
Random*	8.841%	0.628	0.879	1.383%	−9.758%	−5.74e−5	−4.51e−5	1.072%	16.593%	1.372	1.942	1.084%
Momentum	9.129%	0.595	0.822	1.421%	−11.287%	−7.10e−5	−5.66e−5	1.429%	20.223%	1.463	2.067	1.219%
Reversion	6.144%	0.411	0.586	1.415%	−3.600%	−1.60e−5	−1.23e−5	1.363%	7.859%	1.104	1.818	1.297%
DQN	**16.536%**	**0.981**	**1.411**	**1.340%**	**−2.297%**	**−1.18e−5**	**−9.38e−6**	**1.045%**	**25.169%**	**1.790**	**2.544**	**1.066%**

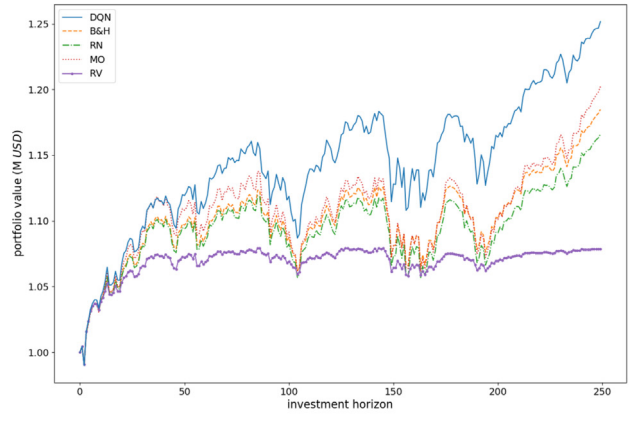
[a] The modified Sharpe/Sterling ratio (Israelsen et al., 2005) is used in the case of negative excess returns.
* The performance of the Random trading strategy is the average over 30 observations.



(a)



(b)



(c)

Fig. A.10. Comparative portfolio value results for the DQN and benchmark strategies of the US portfolio in year of (a) 2016, (b) 2018, and (c) 2019.

last of the investment horizon by performing stable trading and preventing significant loss under the bearish market. The final portfolio value of the DQN strategy is 73.05% higher than that of the B&H strategy, 76.46% higher than that of the RN strategy, 79.64% higher than that of the MO strategy, and 36.19% higher than that of the RV strategy. With a test period of 2019, we observe that the DQN strategy outperforms the benchmark strategies for most of the test period. The final portfolio value of the DQN strategy is 36.2% higher than that of the B&H strategy, 51.71% higher than that of the RN strategy, 24.48% higher than that of the MO strategy, and 220.22% higher than that of the RV strategy.

Table A.9 summarizes the output performance measure results when using the DQN strategy and the benchmark strategies for the US portfolio in the three test periods. This table shows that, for the US portfolio in these periods, the DQN strategy has the best cumulative return, Sharpe ratio, and Sterling ratio performances. The DQN strategy has the second-lowest turnover rate after the B&H strategy.

References

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

Almahdi, S., & Yang, S. Y. (2019). A constrained portfolio trading system using particle swarm algorithm and recurrent reinforcement learning. *Expert Systems With Applications, 130*, 145–156.

Aslanidis, N., & Christiansen, C. (2014). Quantiles of the realized stock–bond correlation and links to the macroeconomy. *Journal of Empirical Finance, 28*, 321–331.

Baur, D. G., & Lucey, B. M. (2009). Flights and contagion – An empirical analysis of stock–bond correlations. *Journal of Financial Stability, 5*, 339–352.

Bertoluzzo, F., & Corazza, M. (2012). Testing different Reinforcement Learning configurations for financial trading: Introduction and applications. *Procedia Economics and Finance, 3*, 68–77.

Bhatia, A., Varakantham, P., & Kumar, A. (2019). Resource constrained deep reinforcement learning. *Proceedings of the International Conference on Automated Planning and Scheduling, 29*, 610–620.

Brock, W., Lakonishok, J., & Lebaron, B. (1992). Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance, 47*, 1731–1764.

Casqueiro, P. X., & Rodrigues, A. J. L. (2006). Neuro-dynamic trading methods. *European Journal of Operational Research, 175*, 1400–1412.

Chen, C. H., & Yu, H. Y. (2017). A series based group stock portfolio optimization approach using the grouping genetic algorithm with symbolic aggregate approximations. *Knowledge-Based Systems, 125*, 146–163.

Chen, T., & Chen, F. (2016). An intelligent pattern recognition model for supporting investment decisions in stock market. *Information Sciences*, 261–274.

Chourmouziadis, K., & Chatzoglou, P. D. (2016). An intelligent short term stock trading fuzzy system for assisting investors in portfolio management. *Expert Systems With Applications, 43*, 298–311.

Consigli, G., & Dempster, M. A. H. (1998). Dynamic stochastic programming for asset–liability management. *Annals of Operations Research, 81*, 131–161.

Cumming, D., Dai, N., & Johan, S. (2017). Dodd-Franking the hedge funds. *Journal of Banking & Finance* 105216.

Cumming, D., Johan, S., & Li, D. (2011). Exchange trading rules and stock market liquidity. *Journal of Financial Economics, 99*, 651–671.

Dempster, M. A. H., & Leemans, V. (2006). An automated FX trading system using adaptive reinforcement learning. *Expert Systems With Applications, 30*, 543–552.

Deng, Y., Feng, B., Kong, Y., Ren, Z., & Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems, 28*, 653–664.

Derigs, U., & Nickel, N. H. (2003). Meta-heuristic based decision support for portfolio optimization with a case study on tracking error minimization in passive portfolio management. *OR Spectrum, 25*, 345–378.

Dulac, A. G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., et al. (2015). Deep reinforcement learning in large discrete action spaces. arXiv preprint arXiv:1512.07679..

Eilers, D., Dunis, C. L., Mettenheim, H. J., & Breitner, M. H. (2014). Intelligent trading of seasonal effects: A decision support algorithm based on reinforcement learning. *Decision Support Systems, 64*, 100–108.

Golub, B., Holmer, M., Mckendall, R., Polhlman, L., & Zenios, S. A. (1995). A stochastic programming model for money management. *European Journal of Operations Research, 85*, 282–296.

Grinold, R. C. & Khan, R. N. (2000). Active portfolio management: A quantitative approach for producing superior returns and controlling risk. McGraw Hill, New York; NY, 2nd ed..

Israelsen, C. L. et al. (2005). A refinement to the sharpe ratio and information ratio. *Journal of Asset Management, 5*, 423–427.

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

Jiang, Z., Xu, D., & Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. arXiv preprint arXiv:1706.10059..

Kouwenberg, R. (2001). Scenario generation and stochastic programming models for asset liability management. *European Journal of Operational Research, 134*, 279–292.

Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., et al. (2017). A unified game-theoretic approach to multiagent reinforcement learning. arXiv preprint arXiv:1711.00832..

Leigh, W., Modani, N., Purvis, R., Wu, Q., & Robert, T. (2002). Stock market trading rule discovery using technical charting heuristics. *Expert Systems with Applications, 23*, 155–159.

Lo, A. W., & Mackinlay, A. C. (1988). Stock market prices do not follow random walks: Evidence from a simple specification test. *The Review of Financial Studies, 1*, 41–65.

Lo, A. W., & Mackinlay, A. C. (2011). *A non-random walk down Wall Street*. Princeton University Press.

Lo, A. W., Mamaysky, H., & Wang, J. (2000). Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. *The Journal of Finance, 55*, 1705–1765.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al.. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602..

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

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

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

Neuneier, R. (1996). Optimal asset allocation using adaptive dynamic programming. *Advances in Neural Information Processing Systems*, 952–958.

Neuneier, R. (1998). Enhancing Q-learning for optimal asset allocation. *Advances in Neural Information Processing Systems*, 936–942.

Jangmin, O., Lee, J., Lee, J. W., & Zhang, B. T. (2006). Adaptive stock trading with dynamic asset allocation using reinforcement learning. Information Sciences 176:2121–2147..

Papailias, F., & Thomakos, D. D. (2015). An improved moving average technical trading rule. *Physica A, 428*, 458–469.

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

Pham, T. H., Magistris, G. D., & Tachibana, R. (2018). Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 6236–6243).

Potvin, J. Y., Soriano, P., & Vallee, M. (2004). Generating trading rules on the stock markets with genetic programming. *Computers & Operations Research, 31*, 1033–1047.

Shah, S., Sinha, A., Varakantham, P., Perrault, A., & Tambe, M. (2019). Solving online threat screening games using constrained action space reinforcement learning. arXiv preprint arXiv:1911.08799..

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature, 550* (7676), 354–359.

Sirignano, J. & Cout, R. (2018). Universal features of price formation in financial markets: Perspectives from Deep Learning. arXiv preprint arXiv:1803.06917..

Srivastava, N., Mansimov, E., & Salakhudinov, R. (2015). Unsupervised learning of video representations using LSTMs. *International Conference on Machine Learning*, 843–852.

Tan, Y., Liu, W., & Qiu, Q. (2009). Adaptive power management using reinforcement learning. *ICCAD*, 461–467.

Wang, Y., Wang, D., Zhang, S., Feng, Y., Li, S., & Zhou, Q. (2016). Deep Q-trading. http://cslt.riit.tsinghua.edu.cn..

Xiong, J., Wang, Q., Yang, Z., Sun, P., Han, L., Zheng, Y., et al. (2018). Parametrized deep Q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. arXiv preprint arXiv:1810.06394..

Zhang, X., Hu, Y., Xie, K., Zhang, W., Su, L., & Liu, M. (2015). An evolutionary trend reversion model for stock trading rule discovery. *Knowledge-Based Systems, 79*, 27–35.

Zhu, Y., & Zhou, G. (2009). Technical analysis: An asset allocation perspective on the use of moving averages. *Journal of Financial Economics, 92*, 519–544.