Expert Systems With Applications 117 (2019) 125–138

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Improving financial trading decisions using deep Q-learning: Predicting the number of shares, action strategies, and transfer learning

Gyeeun Jeong[a], Ha Young Kim[a,b,*]

[a] Department of Financial Engineering, Ajou University, Worldcupro 206, Yeongtong-gu, Suwon, 16499, Republic of Korea
[b] Department of Data Science, Ajou University, Worldcupro 206, Yeongtong-gu, Suwon, 16499, Republic of Korea

ARTICLE INFO

Article history:
Received 10 June 2018
Revised 20 August 2018
Accepted 16 September 2018
Available online 20 September 2018

Keywords:
Reinforcement learning
Deep Q-learning
Stock trading
Trading strategy
Transfer learning

ABSTRACT

We study trading systems using reinforcement learning with three newly proposed methods to maximize total profits and reflect real financial market situations while overcoming the limitations of financial data. First, we propose a trading system that can predict the number of shares to trade. Specifically, we design an automated system that predicts the number of shares by adding a deep neural network (DNN) regressor to a deep Q-network, thereby combining reinforcement learning and a DNN. Second, we study various action strategies that use Q-values to analyze which action strategies are beneficial for profits in a confused market. Finally, we propose transfer learning approaches to prevent overfitting from insufficient financial data. We use four different stock indices—the S&P500, KOSPI, HSI, and EuroStoxx50—to experimentally verify our proposed methods and then conduct extensive research. The proposed automated trading system, which enables us to predict the number of shares with the DNN regressor, increases total profits by four times in S&P500, five times in KOSPI, 12 times in HSI, and six times in EuroStoxx50 compared with the fixed-number trading system. When the market situation is confused, delaying the decision to buy or sell increases total profits by 18% in S&P500, 24% in KOSPI, and 49% in EuroStoxx50. Further, transfer learning increases total profits by twofold in S&P500, 3 times in KOSPI, twofold in HSI, and 2.5 times in EuroStoxx50. The trading system with all three proposed methods increases total profits by 13 times in S&P500, 24 times in KOSPI, 30 times in HSI, and 18 times in EuroStoxx50, outperforming the market and the reinforcement learning model.

1. Introduction

Financial trading is an important, but challenging issue in the financial market. Its processes are usually complicated, because numerous problems must be considered when making financial decisions, particularly financial trading decisions (Brock, Lakonishok, & LeBaron, 1992; Dymova, Sevastianov, & Bartosiewicz, 2010; Kuo, Chen, & Hwang, 2001; Lin, Yang, & Song, 2011; Fama & Blume, 1966). Moreover, real traders are frequently confused by market situations because high volatility and noise often make financial data hard to analyze. In addition, financial task modeling is challenging when traditional algorithms are used owing to the estimation of relationships that are highly nonlinear (Ang & Quek, 2006; Chavarnakul & Enke, 2008; Chen, Firth, & Rui, 2001; Oh & Kim, 2002).

The key elements of financial trading are the identification of trading strategies (Brock et al., 1992; Fama, 1995; Levich & Thomas, 1993) and the determination of the number of shares to trade (Black & Scholes, 1973; Mastinšek, 2006). Early studies on financial trading focused on finding trading rules. For example, Brock et al. (1992) test the moving-average and trading-range breakout rules by examining Dow Jones indexes from 1897 to 1986. Their study shows that buy signals consistently result in higher returns than sell signals, and they are less volatile. Levich and Thomas (1993) research the profitability of the moving-average and simple technical trading rules using a statistical test in foreign currency futures. The results show that simple technical trading rules are profitable; according to a new statistical test, this rule is more significant than the empirical distribution with bootstrap method. Fama (1995) discusses random walks in stock prices and analyzes whether the fundamental buy-and-hold strategy is efficient.

Many studies focus on the relationship between price and trading volume or finding the optimal number of shares to trade (Black & Scholes, 1973; Campbell, Grossman, & Wang, 1993;

* Corresponding author at: Department of Financial Engineering, Ajou University, Worldcupro 206, Yeongtong-gu, Suwon 16499, Republic of Korea.
E-mail addresses: jgenny13@ajou.ac.kr (G. Jeong), hayoungkim@ajou.ac.kr (H.Y. Kim).

https://doi.org/10.1016/j.eswa.2018.09.036
0957-4174/

Karpoff, 1987; Mastinšek, 2006). Since traders in the real market do not trade a fixed number of shares each time, the profits are directly affected by the number of shares. Thus, they generally want to assess the optimal number of shares to trade in order to clearly identify hedge strategies that can optimize portfolios using derivatives.

Many studies investigate automated trading systems (or algorithmic trading [AT]) to identify the best trading strategies (Brownlees, Cipollini, & Gallo, 2011; Chaboud, Chiquoine, Hjalmarsson, & Vega, 2014; Hendershott & Riordan, 2011; Scholtus, Van Dijk, & Frijns, 2014). AT is based on a preprogrammed model that traders can use to improve liquidity and enhance their profits (Hendershott, Jones, & Menkveld, 2011). Hendershott and Riordan (2011) examine AT and its roles, noting that they contribute to the market in terms of liquidity and price deviations from fundamental values. Chaboud, Chiquoine, Hjalmarsson, and Vega (2014) study the impact of AT in the foreign exchange market and find that it increases the speed of price discovery and improves the autocorrelation of high-frequency returns.

However, AT is insufficient to make a profit on every decision at every trading moment because the financial market is highly complicated (Domowitz & Yegerman, 2006; Hu et al., 2015; Yadav, 2015). In their review of its limitations, Hu et al. (2015) find that AT poses a challenge in predicting future market trends. In fact, it hardly outperforms the market when a persistent uptrend exists over a long period. Taking these limitations into account, many attempts have been made to trade stocks based on reinforcement learning.

Reinforcement learning is a technique that enables sequential decisions and can learn unlabeled data. Moody and Saffell (2001) use direct reinforcement learning to optimize portfolios and trade stocks. They use recurrent reinforcement learning to solve stochastic control problems, such as investment decisions. O et al., (2006) apply meta-policy to reinforcement learning—that is, an asset-allocation strategy that selects stocks and optimizes trading. Gorse and Street (2011) attempts index trading using recurrent reinforcement learning. The author uses stock index data with various time intervals, such as daily, weekly, and monthly, and compares the results with genetic programming. The aim is to find an appropriate solution using a genetic algorithm based on Darwin's theory of evolution.

However, there still are some challenges when trading financial stocks with reinforcement learning. First, trading in determining the number of shares is more difficult than trading a fixed number of shares. Few studies deal with this challenge based on the reinforcement study. However, it is necessary to determine how many shares should be bought or sold to adapt the reinforcement system to the real market. Predicting the number of shares to trade is an important issue in the financial market, because trading a fixed number of shares in every trading position cannot reflect the real market situation and it would directly affect profits. If we change the number according to the market situation, we can increase profit. Most research based on reinforcement learning, though, normally focuses on finding trading strategies only. The number of shares is generally set as just one or a fixed value or is regarded as a portfolio's weight (Nevmyvaka, Feng, & Kearns, 2006; O et al., 2006). As far as we know, few studies investigate mechanisms that could predict the appropriate number of shares to trade when trading only one stock.

Second, financial market directions and situations are not always clearly distinct. The financial market often cannot be clearly represented, which makes it difficult for the reinforcement learning model to generate proper decisions, especially when the financial market is chaotic. In reality, traders also face difficulties when deciding what action to take. In this case, it is necessary to better represent financial features, select more appropriate actions or select actions clearly even in confused situations. Some scholars apply various algorithms, such as fuzzy algorithm, to reinforcement learning to extract better features from data (Kuo et al., 2001; Tan, Quek, & Cheng, 2011). Tan et al. (2011) apply the adaptive network fuzzy inference system to reinforcement learning. This supplement model can dynamically identify a trend in the financial market and stock movement. Almahdi and Yang (2017) include economic factors in their financial market model. They propose extending recurrent reinforcement learning with the Calmar ratio to identify buy and sell signals. They also use reinforcement learning in asset allocation and portfolio rebalancing, where it outperforms hedge fund benchmarks.

Finally, financial data is a small amount of data for deep learning. In deep learning, overfitting occurs when there are less data than the parameters to be learned. Financial data are also highly volatile and include unpredictable noise, which makes their patterns unclear. These characteristics lead training to overfitting, and the network is prone to fall into local minimums. To address this problem, some studies extract various features by analyzing small amount of data with various algorithms. Dempster, Payne, Romahi, and Thompson (2001) consider reinforcement learning in foreign exchange trading and compare the method to genetic programming. In subsequent research, Dempster and Leemans (2006) apply reinforcement learning to foreign exchange trades by separating the network into risk management layers and dynamic optimization layers. Deng, Bao, Kong, Ren, and Dai (2017) separate the deep learning part to address market conditions and the reinforcement learning part to make active decisions.

In this paper, our goal is to maximize a trader's total profit by overcoming previously mentioned challenges. We propose three new methods to improve a trading system with reinforcement learning and reflect a financial situation's reality—predicting the number of shares to be traded, action strategies for a confused market, and transfer learning.

First, we propose a *reinforcement learning trading system* combined with the DNN regressor that makes it possible to predict the number of shares to trade stock and the trading action. The network training consists of stepwise training to reduce search space and efficiently predict the number of shares.

Second, we measure market uncertainty with the robustness of the trading action and apply it to our new proposed method, *action strategy*. The action strategy determines one action for a complicated financial situation before training. We measure market condition using the difference of the value of action. When the condition exceeds the pre-determined threshold, we use a predetermined action instead of trained results. This allows us to determine proper action in uncertain financial markets and ensure the algorithm avoids uncertainties. We empirically compare various action strategies. We then verify that the best strategy in a confused market is to postpone the decision to buy or sell the stock. Action strategies can also reflect an investor's propensity by adjusting the strategy's action and threshold.

Finally, we propose *transfer learning* to handle the overfitting problem arising from small and highly volatile financial data. We verify this method with four different indexes: S&P500, KOSPI, HSI, and EUROSTOXX50. To test our new methods, six groups are selected for transfer learning based on correlation and the neural network that predicts errors; each group comprises a subset of component stocks with various relationships with their index. We overcome the issue of data insufficiency and prevent overfitting by using the data characteristics itself. Our basic automatic trading model is based on the deep Q-network (DQN) algorithm, which combines DNN with reinforcement learning. DQN optimizes reinforcement learning through a DNN (Mnih et al., 2013; Mnih et al., 2015), which can discover nonlinear relationships and approximate complicated models. A data-driven approach, such as deep learn-

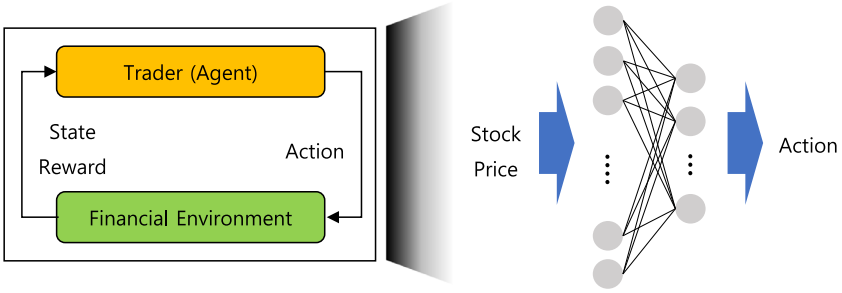

Fig. 1. Deep Q-learning updated by the neural network.

ing, can also find unknown trading rules. Profit is calculated using the modified return over a single period (daily), and the model is evaluated by total profit—that is to say, the summation of profit over the whole test period.

The remaining paper is organized as follows. In Section 2, we describe how to trade a stock using Q-learning. Section 3 introduces our three proposed models. These models determine the number of shares to trade, propose an action for a case of confusion, and transfer learning using correlation and the neural network. The results of the three models are presented in Section 4. Section 5 presents our conclusions.

2. Background: trading based on Q-learning

Reinforcement learning is a learning to maximize a cumulative reward signal when the agent interacts with the environment. As the left hand-side illustration in Fig. 1 shows, in the environment defined by a Markov decision process, the agent selects an action using a given state (Kaelbling, Littman, & Moore, 1996; Sutton & Barto, 1998). According to the Markov decision process, an agent corresponding to the environment chooses an action to maximize the reward (Bellman, 1957). We explain this process further through mathematical formulas. This process consists of (S, A, P, R, γ), where S is a set of states $\{s_1, s_2, \cdots\}$, A is a set of actions $\{a_1, a_2, \cdots\}$, P is a set of probabilities of state s' at time $t+1$ where s is the state at time t and a is the action at time t. R is a set of rewards $\{r_1, r_2, \cdots\}$, and γ is the discount factor. The reward at time $t+1$ depends on the action at time t; further, the next state is determined by the action. The value of the action can be considered as the discounted sum of the future reward. Thus, if we choose action a in state s in accordance with policy π, the action value is

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} | s_t = s, a_t = a\right]. \quad (1)$$

This action value is called the Q-value. When the state at time t, s_t is s and the action at time t, a_t is a, the optimal Q-value, $Q^*(s, a)$, is determined in accordance with the optimal policy that maximizes the Q-value. It can be represented as follows:

$$Q^*(s,a) = max_\pi Q_\pi(s,a). \quad (2)$$

This value can be expressed using the Bellman equation:

$$Q^*(s,a) = \max_a \left[r_t + \gamma Q\left(s_{t+1}, a'\right)\right]. \quad (3)$$

Q-learning (Watkins & Dayan, 1992) is one of the methods of reinforcement learning while updating the Q-value to maximize the cumulative rewards. Without a deterministic policy, the algorithm selects the action with the largest Q-value. The Q-value is continuously updated so that it converges to the optimal Q-value to find the best action. The process of updating the Q-value is as follows:

$$Q(s_t,a_t) := Q(s_t,a_t) + \theta * \left\{r_t + \gamma Q\left(s_{t+1}, a'\right) - Q(s_t,a_t)\right\}, \quad (4)$$

where θ is the learning rate.

Q-learning has been used in many studies for trading stock because financial trading is difficult to generalize to a deterministic policy (Yang et al., 2012). Many variables, as well as randomness, exist in a financial environment; therefore, we use a DQN introduced by Mnih et al. in 2013 that updates the Q-value with a neural network and represents the data's nonlinearity effectively (see Fig. 1). A neural network comprises fully connected layers. Unlike the original DQN, we do not use the experience replay technique to update the Q-value because financial data are highly random by themselves.

Here, S is a finite set expressed as $S = \{s_1, s_2, \cdots, s_T\}$, where each state s_t represents $p_t - p_{t-1}$ from $t-199$ to t, and p_t is the trade closing price at time t. A is a finite action set that is expressed as $A = \{1, 0, -1\}$. In the action set, 1 represents BUY, 0 represents HOLD, and -1 represents SELL. At each step, the action with the biggest Q-value is selected. The reward function is introduced in Wang et al. (2017) where it calculates the rate of return at time t, reflecting the price at time $t-n$. The function is provided in formula (5), where p_t is the trade closing price at time t and a_t is an action at time t.

$$r_t = \left(1 + a_t \times \frac{p_t - p_{t-1}}{p_{t-1}}\right)\frac{p_{t-1}}{p_{t-n}}. \quad (5)$$

As you can see, this reward function is a transformation of stock return. Wang et al. (2017) set n to 200. For a fair comparison we also set n to 200. As we set n to a large number, we can reflect the long-term trend of stock prices. The performance of every model is estimated as the sum of the daily rate of returns, that is called 'total profit', during the test period. This allows us to evaluate the models intuitively, as shown in formulas (6) and (7):

$$profit_t = a_t \times \frac{p_t - p_{t-1}}{p_{t-1}} \quad (6)$$

$$Total\ profit = \sum_t profit_t \quad (7)$$

The profit is used as an indicator for whether we choose a good action and whether we predict the next day's trend of stock well. According to formula (7), for example, if we do not trade (that is, only HOLD action is selected) in the whole period, our total profit is zero.

Our trading system is a system that determines the trading action at time t by analyzing the closed price from time t-200 to time t-1. This trading system is based on day trading, including a constraint that the system can trade once in a day. When we trade a stock, we assume that it is possible to trade regardless of the time, day, and the number of shares. The transaction cost is set to zero and short selling is allowed. We assume that trading is not affected by the trader's wealth and situational financial exchanges.

3. Methodology: proposed new trading system methods

3.1. Determining the number of shares to trade

We propose three methods to predict the number of shares to trade for maximizing the total profits depending on the Q-value and the DNN regressor at each moment. The number of shares is determined as a continuous number. The continuous number means that the number is determined not to be an integer, such as 1 or 2, but to a rational number according to the output of the network. Jiang, Xu, & Liang (2017) optimize a portfolio by weighting all stocks that make up the portfolio with deep learning. To manage problems arising from a discrete action in reinforcement learning, they use continuous weighting on the portfolio. Pendharkar and Cusatis (2018) further show that continuous action is more effective in stock trading compared to discrete action. The DNN regressor predicts continuous numbers of shares. By trading continuous numbers of shares, the discrete action risk can be regulated.

At each moment, the number of shares is limited to a fixed number (L) to prevent it from being severely influenced by the data pattern and to prevent a serious impact on the market. This limitation also circumscribes a large learning space within a narrow bound, thereby leading to an improvement in the learning speed. In this study, L is assigned the value of 10 based on observation; this value can be changed as per the trader's needs and wealth. As shown in formulas (8) and (9), reward function and profit function are calculated by this predicted number of shares at time t, num_t.

$$r_t = num_t \times \left(1 + a_t \times \frac{p_t - p_{t-1}}{p_{t-1}}\right)\frac{p_{t-1}}{p_{t-n}}, \tag{8}$$

$$profit_t = num_t \times a_t \times \frac{p_t - p_{t-1}}{p_{t-1}}. \tag{9}$$

We use the following three methods to determine the appropriate number of shares to trade.

3.1.1. Determining the number of shares with the Q-values

We determine the number of shares according to the reliability of the actions in this method. In deep Q-learning, the action is determined by the Q-values. If the selected Q-value is adequately larger than the other values, we assume that the selected action is reliable.

We determine the number of shares depending on the Q-value's proportion using the *Softmax* function, called the NumQ method. The *Softmax* function consists of an exponential function that makes the values positive and scales a large value to a larger value when the value is bigger than zero, so that the proportion of the selected Q-value can be captured more effectively. The *Softmax* function can be described as follows:

$$Softmax_{j,l}(q) = \frac{e^{q_{j,l}}}{\sum_k e^{q_{k,l}}} \tag{10}$$

where $q_{j,l}$ is the output of the jth neuron in the lth layer. We place the output of the Q-value into a *Softmax* function to calculate the number of shares and select the number of shares according to the selected Q-value. For example, if the buy action is selected, the number of shares for the buy action is also consistently selected.

As shown in Fig. 2, every layer consists of fully connected layers. There are two branches—one that determines the action and another that determines the number of shares per action. If the branch is updated to find the proper action, we call the branch the *action branch*. If the branch is used to predict the number of shares, we call the branch the *number branch*. The details of the process are expressed in formulas (11) to (16), where all Ws are weight vectors, I_t is an input vector at time t, all bs are biases,

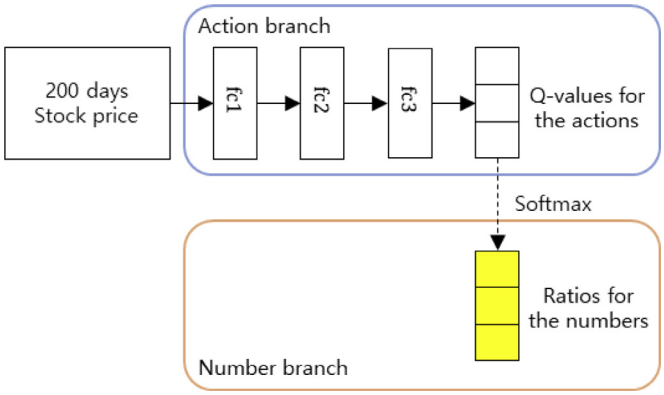

Fig. 2. Architecture of the **NumQ** method.

Q_{action} denotes the Q-values for the actions, R_{num} denotes the ratio of the number of shares, and num_t is the number of shares at time t.

The first three layers for the action branch use the *Relu* function Nair & Hinton, 2010) for the activation function, while the last layer for the action branch uses a linear function as described in formulas ((11) to (14). The R_{num} is calculated using the output of fc3 layers with the *Sigmoid* function and the *Softmax* function, as shown in formula (15). The *Sigmoid* function is used to normalize the value. The number of shares, num_t, is determined by L and $R_{num}(s_t,\ a_t^*)$. It is determined by the action a^* that maximizes Q_{action} in formula (16). Throughout the study, L denotes the maximum number of shares as described above even when there is no explanation offered.

$$\text{fc1} = Relu(W_1 I_t + b_1) \tag{11}$$

$$\text{fc2} = Relu(W_2 \times \ \text{fc1} + b_2) \tag{12}$$

$$\text{fc3} = W_3 \times \ \text{fc2} + b_3 \tag{13}$$

$$Q_{action}(s_t,\ a_t) = W_4 \times Relu(\text{fc3}) + b_4 \tag{14}$$

$$R_{num}(s_t,\ a_t) = Softmax(W_4 \times Sigmoid(\text{fc3}) + b_4) \tag{15}$$

$$num_t = R_{num}(s_t, a_t^*) \times L \text{ where } a_t^* = argmax\, Q_{action}(s_t,\ a_t) \tag{16}$$

Fully connected (fc) layers in this DQN have 200 neurons in the first (fc1), 100 in the second (fc2), and 50 in the third (fc3) hidden layer. The number of neurons in each layer is determined by trial and error. The last layer is the output layer and has three neurons because we have three actions to make the decision (BUY, HOLD, and SELL). Each neuron in the last layer of the action branch reflects three action possibilities.

3.1.2. Determining the number of shares with an action-dependent DNN regressor

In this method, we determine the number of shares using the number's own neural network. We add a separated branch instead of using the existing weight of the action branch and optimize this branch only for the number. To maintain the features' consistency, we add the number branch from the second layer. Thus, the number branch can share the features of the action branch and this method is called 'NumDReg – AD' . The networks are updated separately, as shown in formulas (17) and (18), and we use the same notations as in Section 3.1.1.

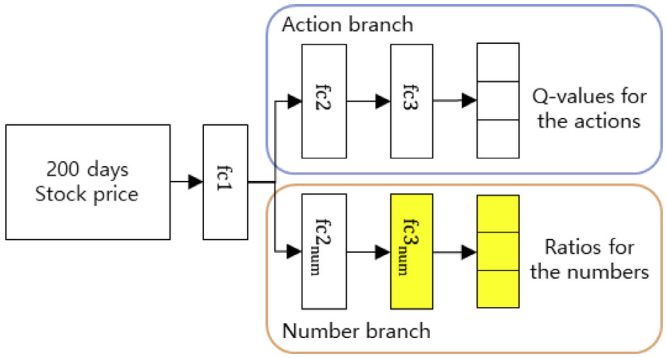

Fig. 3. Architecture of the **NumDReg – AD** method.

$$Q_{action}(s_t, a_t) := Q_{action}(s_t, a_t) + \theta * \left\{ r_t + \gamma Q_{action}\left(s_{t+1}, a'\right) - Q_{action}(s_t, a_t) \right\}, \quad (17)$$

$$R_{num}(s_t, a_t) := R_{num}(s_t, a_t) + \theta * \left\{ r_t + \gamma R_{num}\left(s_{t+1}, a'\right) - R_{num}(s_t, a_t) \right\}. \quad (18)$$

The process of calculating the action branch is same as in Section 3.1.1, and the number branch is expressed in formulas (19) to (21) with the same notations as in Section 3.1.1.

$$\mathrm{fc2_{num}} = Relu(W_2 \times \mathrm{fc1} + b_2) \quad (19)$$

$$\mathrm{fc3_{num}} = Sigmoid(W_3 \times \mathrm{fc2_{num}} + b_3) \quad (20)$$

$$R_{num}(s_t, a_t) = Softmax(W_4 \times \mathrm{fc3_{num}} + b_4) \quad (21)$$

The action branch and the number branch have similar layer structure. Both branches comprise three layers. Their activation functions are used in the same order. The first layer uses *Relu* as the activation function. The layers of the action branch also use the *Relu* function for the second and third layers. The last layer of the action branch uses the linear function. Moreover, the number branch takes the *Relu*, *Sigmoid*, and *Softmax* functions in the order of the layers, as described in formulas (19) to (21). The number of shares is calculated in the same way as in Section 3.1.1. As we add the DNN regressor, the trading network includes both the neurons in the action branch and the neurons in the number branch which is from the DNN regressor. The increased number of neurons can cause overfitting (Aquino, Chamhum Salomão, & Azevedo, 2016; Taghavifar, Taghavifar, Mardani, & Mohebbi, 2014). Thus, the number of neurons in each layer is adjusted to 100 in the first (fc1), 50 in the second (fc2, $\mathrm{fc2_{num}}$), and 20 in the third (fc3, $\mathrm{fc3_{num}}$) layers. The last layers are the output layers and have three neurons. Fig. 3 describes the complete architecture.

The learning space in the NumDReg – AD is too large to provide a direct solution even with reduced number of neurons in each layer compared with NumQ. Thus, we train the network step by step. This method is called NumDReg – AD with three – step training. The whole structure is the same as that in the NumDReg – AD method. Further, we only divide the training process into three steps. In the first step, we train the weights using the NumQ method with reduced neurons to get information on the Q-value for the action. Next, we train the weights for the number branch by freezing the weights for the action branch. After training the number branch, we optimize the whole weight using an end-to-end process. The process is described in Fig. 4.

3.1.3. Determining the number of shares with an action-independent DNN regressor

As we add more branches to the NumDReg – AD method, the number branch has its own network. However, the number of shares is still affected by the action, as the number is determined according to the selected action, which is the biggest value among the Q-value for action. Thus, we determine the number of shares in the regression to construct the layers for the number regardless of the action. This method is called 'NumDReg – ID' . We can then construct the branches that are not affected by the selected action, but only affected by the characteristics of the financial market. The network structure is the same as that in NumDReg – AD method, except for the output size of the number branch. The output size is set to one instead of three. The structure is described in Fig. 5. The training process is also divided into three steps, called NumDReg – ID with three – step training.

3.2. Action strategies in a confused market

When the market situation proves to be too difficult to make a robust decision, a pre-defined signal is given to minimize the loss caused by uncertain information. We call this the *action strategy*. In this method, the Q-value is still regarded as the robustness of the action, and we compare the probability of the Q-values to decide whether the market is confused. If a selected Q-value does not have a markedly large percentage compared with the other values, we presume that the stock has no clear trend, and then provide a specific action. Otherwise, we choose the action with the biggest Q-value as a general case. More specifically, if the difference between the probabilities of BUY,$Q(s_t, a_{\mathrm{BUY}})/\Sigma|Q(s_t, a)|$ and SELL, $\frac{Q(s_t,\ a_{\mathrm{SELL}})}{\sum|Q(s_t,\ a)|}$ is lower than our pre-determined threshold, we provide an action in accordance with our strategy. Threshold is determined by experience. The function of the action strategies is given in formula (22):

$$\frac{|Q(s_t,\ a_{\mathrm{BUY}}) - Q(s_t,\ a_{\mathrm{SELL}})|}{\sum|Q(s_t,\ a)|} < threshold. \quad (22)$$

We use three action strategies, for which the given action is BUY, HOLD, or SELL and the threshold is given. The training algorithm, including the action strategy, is expressed in Algorithm 1. At each episode, we first set state s_t and train the Q-network and the DNN regressor to obtain the Q-values for the action and the ratio for the number. Using these values, we select the action and apply the action strategy according to formula (22). When the difference between the probabilities of the BUY signal and SELL signal is lower than our threshold, the action is set to the predetermined action by the strategy. However, if the difference is higher than the threshold, the action is chosen greedily.

3.3. Transfer learning from the index component stocks

We introduce transfer learning using a subset of the indexes' component stocks. Financial data are hard to train efficiently because the financial market is highly volatile and such data alone are insufficient. Thus, we pretrain the model with transfer learning to solve this problem.

Transfer learning is a method that transfers the pretrained model and learns the target data. One method of transfer learning is a learning that involves using different data with target data, but related to target data (Pan & Yang, 2010). This approach is useful when the data for learning or the information provided are insufficient (Ghosh, Laksana, Morency, & Scherer, 2016; Lu et al., 2014; Wu, Lance, & Parsons, 2013).

Owing to time constraints, it is difficult to use all component stocks. Besides, we select some subsets of component stocks that have different relationships between the index stocks and their

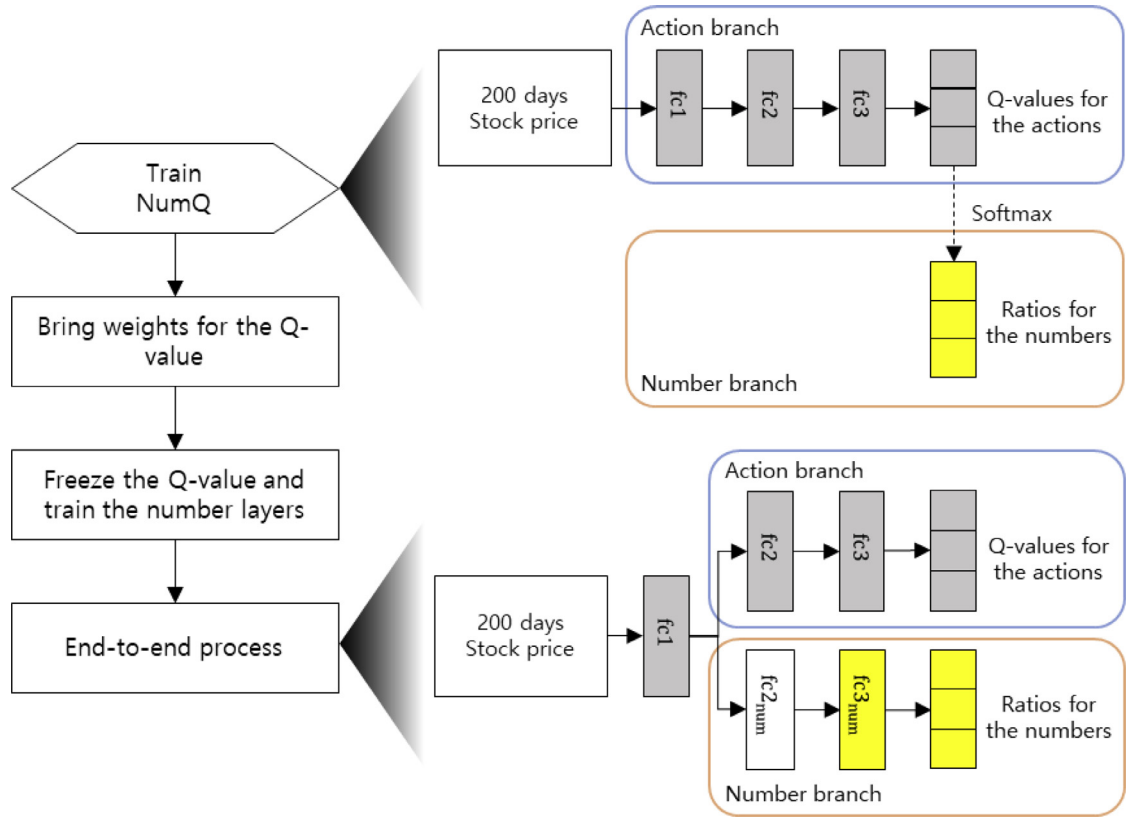

Fig. 4. Flow chart of the **NumDReg** – **AD with three** – **step training** training process.

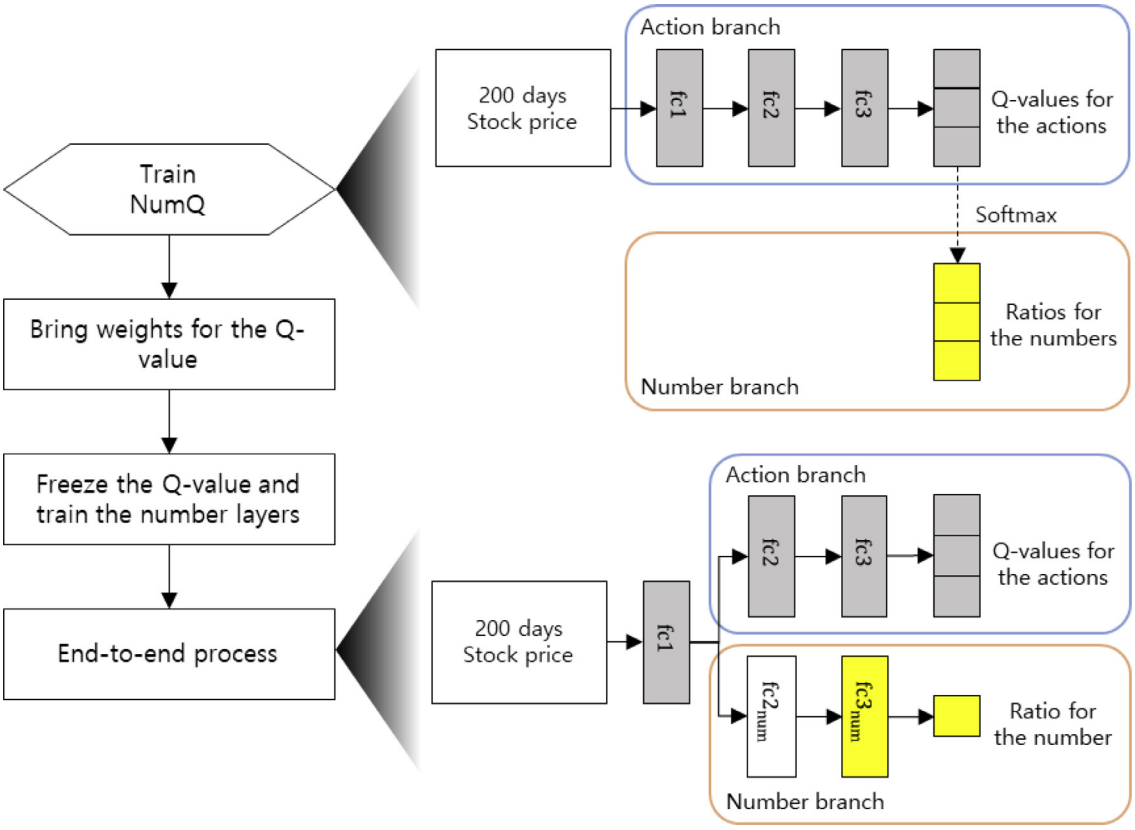

Fig. 5. Flow chart of the **NumDReg** – **ID with three** – **step training** training process.

Algorithm 1 Q learning including the action strategy.

Initialize the Q network and load the pretrained network.
Total profit = 0
for each episode **do:**
  Set state s_t;
  Obtain the Q-values for the action and the number in accordance with the Q-network.
  if $\frac{|Q(s_t,\ a_{BUY})-Q(s_t,\ a_{SELL})|}{\sum |Q(s_t,\ a)|} < threshold$, **then:**
    The action is given by the strategy;
  else:
    Action $a_t = \text{argmax}\{Q(s_t, a)\}$;
    Set next state s_{t+1};
    Calculate r_t and $profit_t$;
    Store memory (s_t, a_t, r_t, s_{t+1}) in buffer.
  for each mini-batch sample **do:**
    $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + learning\ rate^*(r_t + \gamma Q(s_{t+1},\ a') - Q(s_t,a_t))$,
    $Total\ profit \leftarrow Total\ profit + profit_t$.
  end for
end for

component stocks according to two different measures. We then compare their effect. We use two relationship measures, correlation and the neural network, to select the stocks. We compare these two methods to analyze the relationships and their effects. Correlation is a statistical indicator often used to analyze the relevance between stocks in financial markets. The correlation, ρ, between two stocks, X and Y, is shown in formula (23):

$$\rho_{X,Y} = \frac{cov(X,\ Y)}{\sigma_X \sigma_Y}, \tag{23}$$

where σ_X is the standard deviation of X, σ_Y is the standard deviation of Y, and $cov(X,\ Y)$ is the covariance of X and Y.

The neural network is used to determine the relevance between data patterns. It comprises two layers. When the training data are X and output is Y, and there is only one hidden layer, both layers can be described, such as in formulas (24) and (25), respectively:

$$Z = f^1\ \left(W^1 X + b^1\right) \tag{24}$$

$$Y = f^2\left(W^2 Z + b^2\right) \tag{25}$$

where W^1 and W^2 are weight matrices, b^1 and b^2 are the biases, and f^1 and f^2 are activation functions. The goal is to train weight matrices that ensure $X = Y$. We measure their relationship according to the predicting error. To find the proper weight matrices, our network has one hidden layer with five neurons. The activation functions, f^1 and f^2, are *Relu* and the loss function is a mean-squared-error (MSE).

With these two methods, we create three groups by selecting from component stocks inspired by Heaton, Polson, and Witte (2017): *high-relationship component stocks, half of high-relationship and half of low-relationship component stocks*, and *low-relationship component stocks*. High-relationship component stocks and low-relationship component stocks comprise stocks that have high or low relationship with the index stock. Half of high-relationship and half of low-relationship component stocks (here on, high and low relationship) comprise two relationships. Half of the stocks have high relationship with the index, while the other half has low relationship with the index. Thus, we create six groups in total.

In case of correlation, three groups are created by the stocks based on their correlation. The stocks in order of high correlation with index stocks are selected for the high-relationship group. Similarly, the stocks in order of low correlation with index stock are selected for the low-relationship group. We choose half of the stocks in order of high correlation and other half in order of low correlation for the high and low relationship group.

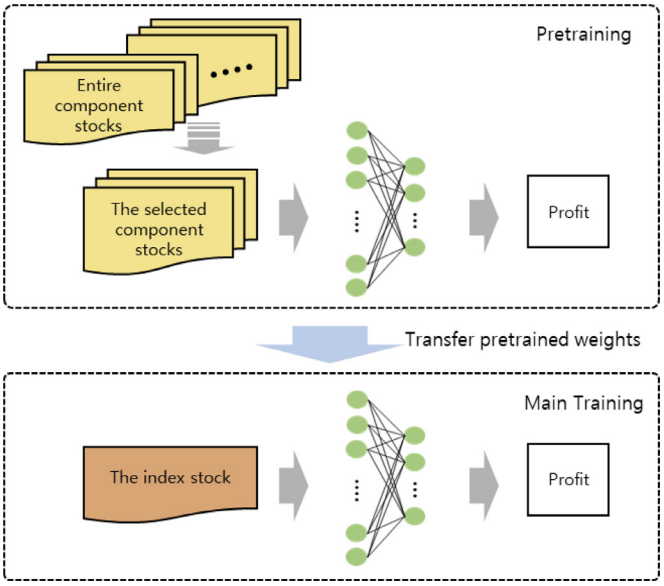

Fig. 6. Training process of transfer learning.

In case of the neural network, three groups are created by the stocks based on the predicting error, MSE. The stocks with low MSE are selected for the high-relationship group, the stocks with high MSE are selected for the low-relationship group, and half of the high-relationship stocks and half of the low-relationship stocks are selected for the high and low relationship.

After we choose the stocks, we pretrain the model for each group with the selected stocks. In each group, every stock is pretrained separately; however, all the pretrained stocks share weights. With these pretrained weights, we train the main model that makes a profit with the index. To verify its effectiveness, we use four different indexes: S&P500, HIS, EuroStoxx50, and KOSPI. A different number of component stocks are chosen according to the number of component stocks included in the index. Fig. 6 shows the training process of transfer learning.

Our final model is presented in Algorithm 2. The process starts by determining the data. We calculate the correlation and MSE of the neural network between the index stock and its component stocks. According to these two measures and the different relationships, we create six groups. Then, we select the group with the highest total profit. Next, we apply transfer learning to the DQN with the DNN regressor. The DNN regressor predicts the number of shares to trade. After pretraining, we train the model with the

Algorithm 2 The entire process, including transfer learning, determining the share trading number method, and using the action strategy.

```
Load the index data and the index component stocks;
Calculate the correlation between the index and the index components stocks or calculate the MSE
  between them with the neural network;
Create the six groups with the highest or lowest relationship or both using correlation and the MSE;
Select the best group among the six groups;
Pretrain the model including NumQ or NumDReg – AD with the chosen group;
if NumQ, then:
    Load NumQ weights pretrained by transfer learning;
    Train the model, including NumQ with the action strategy;
else if NumDReg – AD, then:
    Load NumDReg – AD weights pretrained by transfer learning;
    Train the model, including NumDReg – AD with the action strategy;
else if NumDReg – AD with three – step training, then:
    Load NumQ weights pretrained by transfer learning;
    Freeze NumQ weights and only train the number branch based on the NumDReg – AD algorithm;
    Complete an end-to-end process with the action strategy;
else if NumDReg – ID, then:
    Load NumQ weights pretrained by transfer learning;
    Freeze NumQ weights and only train the number branch based on the NumDReg – ID algorithm;
    Complete an end-to-end process with the action strategy;
```

Table 1
Data Descriptions.

Main training	Training period		Test period
Pretraining	Training	Validation	x
S&P500 index (SP500)	Jan 1, 1987–Nov 4, 2002	Nov 5, 2002–Aug 10, 2006	Aug 11, 2006–Dec 31, 2017
Hang Seng Index (HSI)	Jan 2, 2001–May 2, 2008	May 3, 2008–Jul 21, 2009	Jul 22, 2009–Dec 29, 2017
EuroStoxx50 index	Apr 05, 1991–Feb 19, 2003	Feb 20, 2003–Jul 13, 2005	Feb 21, 2003–Dec 29, 2017
Korea Stock Price Index (KOSPI)	Jul 1, 1997–Jul 11, 2006	Jul 12, 2006–May 05, 2008	Mar 06, 2008–Dec 28, 2017

index data. The algorithm differs depending on which method to predict the number of shares is used.

4. Experimental results and comparisons

We conduct various experiments to verify our proposed approaches using four different index stocks: S&P500, HSI, Eurostoxx50, and KOSPI. All stocks are downloaded from *Thomson Reuters* and *Yahoo! Finance*. Missing values are assigned using the value of the previous day.

As shown in Table 1, training with the index data is called *main training*. The training period accounts for nearly 60% of the total period. The remaining 40% is used as the test dataset. Transfer learning with the index's component stocks is called *pretraining*. The training period accounts for nearly 80% of the training period of main training. The remaining 20% is used as the validation data. The whole period includes various financial issues, such as the real estate bubble; the stock market bubbles in Japan (1985–89), Thailand, Malaysia, and other Asian countries (1992–1997); the dot-com bubble (1992–2002); the over-the-counter stocks bubble in the United States (1995–2000); and the savings and loan crisis (1980s) (DeLong & Magin, 2006; Flood, 2012; Kindleberger & Aliber, 2005). It also involves the global financial crisis (2007–08) (Flood, 2012).

The four index stocks can have different patterns during the same period. The movement of the index stocks and their detailed training periods are described in Fig. 7 and Table 1. As Fig. 7 illustrates, S&P500 shows the most gradual increase in these stocks. HSI and KOSPI also show an upward trend, but they are more volatile than S&P500. Euro stoxx50 does not show a particular trend in the test period, but shows a variety of movements. As we mentioned in Section 2, we set the window size to 200 days to capture these financial price trends.

Mini-batch learning (Cotter, Shamir, Srebro, & Sridharan, 2011; Feyzmahdavian, Aytekin, & Johansson, 2016; Li, Zhang, Chen, & Smola, 2014), where weights are updated for each minibatch, is used for pretraining. For the main training, we use online learning, where weights are updated for each input (Duchi & Singer, 2009; Shalev-Shwartz, Singer, & Ng, 2004; Sudo, Sato, & Hasegawa, 2009; Vijayakumar, D 'souza, & Schaal, 2005). The Adam optimizer is used as the optimization function (Kingma & Ba, 2014). The network comprises four layers, with the hyperparameters set to 0.0001 for the learning rate, 0.85 for gamma, and 64 for the batch size. The learning rate for the end-to-end process is set to 0.0000001 for NumDReg – AD with three – step training and 0.00001 for NumDReg – ID with three – step training. Every hyperparameter is identified by manual search (Bergstra & Yoshua Bengio, 2012), and then determined by trial and error.

Table 2 provides the abbreviations of models for convenience. To identify whether the model outperforms the market, we use the MKT model, which buys the index stock for the whole period, and then calculate the profit and total profit using formulas (6) and (7). The RL model involves trading with reinforcement learning using the index data; it does not include pretraining. This is also consistent with the model proposed by Wang et al. (2017). IDX involves pretraining with the index stock. The CR and NE models involve pretraining. They are separated by the criteria used to select the component stocks. If we use correlation to select the stocks, we use the CR model, and if we use the neural network, we use the NE model. The NQ and NDA models refer to the method that predicts the number of shares. Here, the NumQ and NumDReg – AD methods are used for main training without pretraining. If the pretraining is conducted using the NE component stocks and also includes predicting the number of shares, we use the NENQ and NENDA models. The NENDA3 model represents the NumDReg – AD with three – step training method, which applies stepwise training to NENDA. In NENDI3, we use the NumDReg – ID with three – step training method. The NENDI3–BUY, NENDI3–HOLD, and NENDI3–SELL are used to finally apply action strategies to our NENDI3 model.

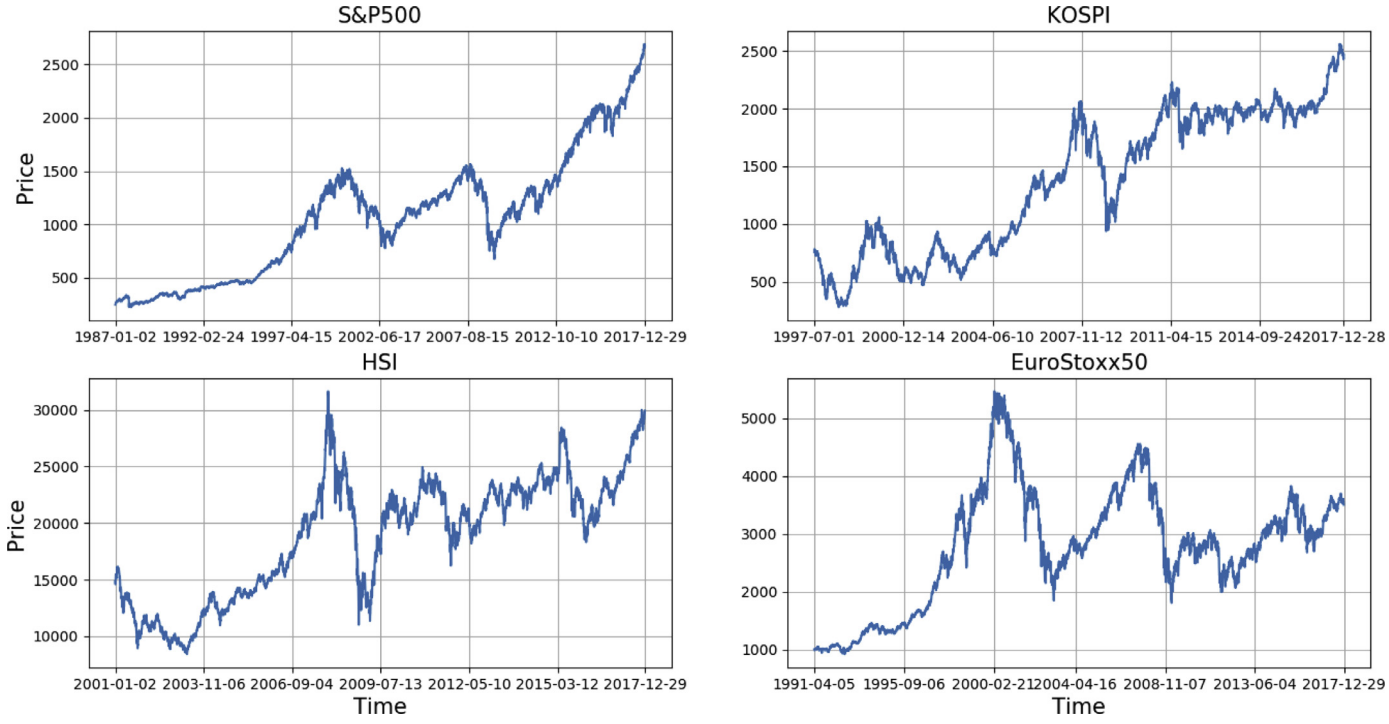

Fig. 7. The movement of four index stocks.

Table 2
The abbreviations of models.

Abbreviation	Pretrained data measure	Main training data	Pretrained Number method	Number method	Step-wise training	Action strategy
MKT	N/A	N/A	N/A	N/A	N/A	N/A
RL	N/A	Index	N/A	N/A	N/A	N/A
IDX	Index	Index	N/A	N/A	N/A	N/A
CR	Correlation	Index	N/A	N/A	N/A	N/A
NE	Neural Network	Index	N/A	N/A	N/A	N/A
NQ	NE	Index	N/A	NumQ	N/A	N/A
NENQ	NE	Index	NumQ	NumQ	N/A	N/A
NDA	NE	Index	N/A	NumDReg – AD	N/A	N/A
NENDA	NE	Index	NumDReg – AD	NumDReg – AD	N/A	N/A
NENDA3	NE	Index	NumQ	NumDReg – AD	Yes	N/A
NENDI3	NE	Index	NumQ	NumDReg – ID	Yes	N/A
NENDI3–BUY	NE	Index	NumQ	NumDReg – ID	Yes	Buy
NENDI3–HOLD	NE	Index	NumQ	NumDReg – ID	Yes	Hold
NENDI3–SELL	NE	Index	NumQ	NumDReg – ID	Yes	Sell

4.1. Transfer learning from related component stocks

To pretrain the model, ten stocks are selected in accordance with the correlation and the neural network that predicts errors for S&P500 and KOSPI. There are 304 component stocks in S&P500 and 342 in KOSPI. Further, as HSI is 25 and Eurostoxx50 is 30, the number of available component stocks is relatively small. Thus, we need to generate three groups with these components stocks. We selected six of these for pretraining. We list the selected companies using their stock symbols in Table 3. *High, High and Low*, and *Low* represent the relationships between the index stocks and their six or ten selected component stocks. *High* describes the component stocks that are highly related to the index (i.e., stocks with high Pearson's correlation and small predicting error, as mentioned in Section 3.3). *High and Low* describes half of component stocks with high correlation and another half with low correlation to the index. *Low* describes the component stocks that are relatively less related to the index (i.e., stocks with low Pearson's correlation and large predicting error). We first pretrain the model using these selected component stocks.

After pretraining the model with the selected component stocks, we train the main model with the indexes. We compare the models to find which group is the most effective for pretraining.

As in Table 4 and Fig. 8, all pretraining was found to be effective in improving the total profits; indeed, every pretrained model outperforms the MKT model. When we compare the CR models with the NE models, NE total profits are higher than CR total profits in most cases—that is to say, they are higher than CR total profits by 4% to 40%. In S&P500, NE total profits for *High, High and Low*, and *Low* are 40%, 36%, and 11% higher than CR total profits for *High, High and Low*, and *Low*, respectively. In KOSPI, NE total profits for *High, High and Low*, and *Low* are 4%, 49%, and 38% higher than CR total profits for *High, High and Low*, and *Low*, respectively. In HSI, NE total profits for *High and High* and *Low* are 7% and 10% higher than CR total profits for *High* and *High and Low*, respectively. However, NE total profits are 13% lower than CR total profits for *Low*. In EuroStoxx50, NE total profits for *High, High and Low*, and *Low* are 31%, 33%, and 25% higher than CR total profits for *High, High and Low*, and *Low*, respectively.

Table 3
List of stock symbols selected by correlation and the neural network that predicts errors.

	S&P500		KOSPI					
	Correlation		Neural Network		Correlation		Neural Network	
	High	Low	High	Low	High	Low	High	Low
1	AXP	MAC	JPM	AJG	005930.KS	002420.KS	007590.KS	006040.KS
2	MS	HCN	PG	RJF	016360.KS	003060.KS	002690.KS	008930.KS
3	BK	SPG	HON	ECL	005940.KS	015540.KS	009810.KS	002990.KS
4	EMR	SCG	ABT	DHR	006800.KS	005610.KS	003920.KS	011200.KS
5	RJF	FL	JNJ	XRX	003540.KS	006110.KS	012750.KS	05930.KS
6	DOV	SIG	KMB	MNST	009150.KS	012170.KS	008730.KS	010060.KS
7	PPG	HCP	TRV	PVH	005490.KS	000950.KS	000810.KS	009970.KS
8	BBT	WEC	ADP	ROST	003470.KS	002070.KS	003460.KS	007310.KS
9	ETN	KIM	DOV	MKC	005380.KS	004960.KS	006890.KS	002350.KS
10	C	SO	CAH	ETR	016610.KS	011150.KS	103140.KS	004490.KS

	HSI		EuroStoxx50					
	Correlation		Neural Network		Correlation		Neural Network	
	High	Low	High	Low	High	Low	High	Low
1	0001.HK	0002.HK	0002.HK	0005.HK	ALVG.DE	AD.AS	AXAF.PA	BASFn.DE
2	0005.HK	0006.HK	0012.HK	0175.HK	AXAF.PA	DANO.PA	DBKGn.DE	ESSI.PA
3	0012.HK	0175.HK	0016.HK	0267.HK	DAIGn.DE	ESSI.PA	ISP.MI	SAF.PA
4	0016.HK	1038.HK	0019.HK	0941.HK	DBKGn.DE	SAF.PA	NOKIA.HE	SGEF.PA
5	0857.HK	1044.HK	0023.HK	1038.HK	SIEGn.DE	SGEF.PA	PHG.AS	VOWG_p.DE
6	0941.HK	1093.HK	0386.HK	1109.HK	SOGN.PA	URW.AS	SGOB.PA	URW.AS

Table 4
Results of the pretrained model.

Model	S&P500			KOSPI		
MKT	0.9654			0.6176		
RL	0.8647			0.5511		
IDX	1.3621			0.6501		
CR	**High**	**High and Low**	**Low**	**High**	**High and Low**	**Low**
	0.9254	1.0392	1.4310	1.0097	1.0761	1.4988
NE	**High**	**High and Low**	**Low**	**High**	**High and Low**	**Low**
	1.2920	1.0761	1.595	1.0515	1.6083	2.0641

Model	HSI			EuroStoxx50		
MKT	0.2852			−0.3359		
RL	0.3841			0.5096		
IDX	0.5932			0.8675		
CR	**High**	**High and Low**	**Low**	**High**	**High and Low**	**Low**
	0.6949	0.6559	0.6235	0.9877	0.6129	0.3791
NE	**High**	**High and Low**	**Low**	**High**	**High and Low**	**Low**
	0.7422	0.7242	0.5422	1.2901	0.8179	0.4752

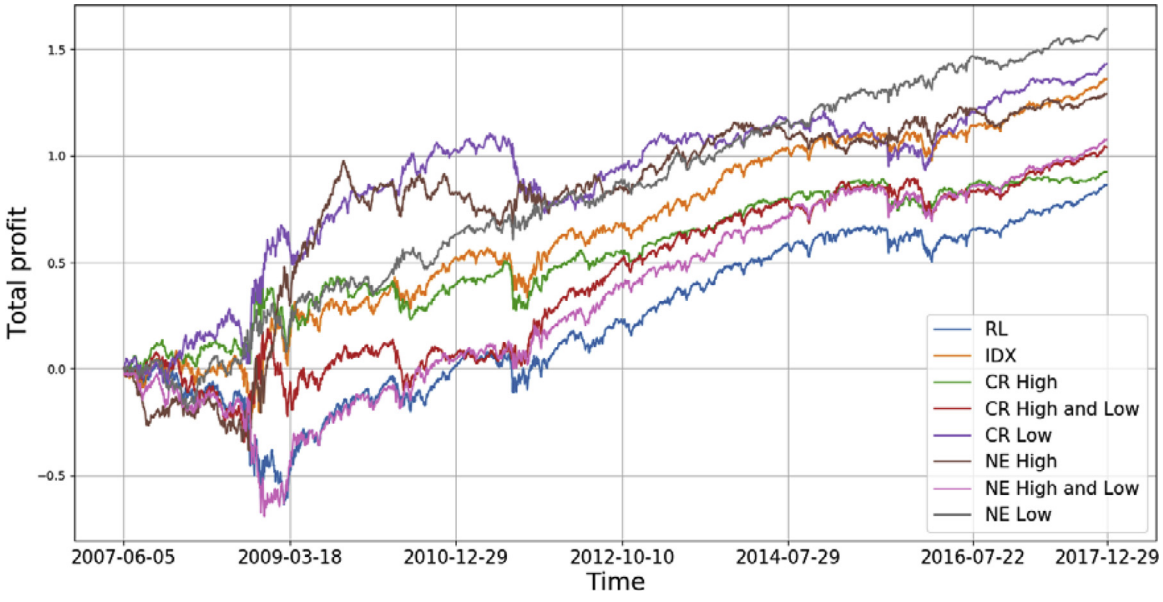

Fig. 8. Comparison of the transfer learning models according to the relationship between S&P500 and its component stocks.

Table 5
The model's results from trading different numbers of shares using the NQ and ND models, and their transfer learning applications with step-wise training.

Model	S&P500	KOSPI	HSI	EuroStoxx50
NQ	3.8912	3.2163	4.7626	3.3882
NDA	2.9759	2.5334	4.3639	2.6322
NENQ	7.7169	3.9665	7.2330	4.0376
NENDA	4.4279	4.5590	6.1747	3.1082
NENDA3	8.6873	5.8359	9.1167	4.3487
NENDI3	9.6789	10.5704	10.4296	6.1502

Thus, the neural network is more effective in modeling the relationship between component stock and index. We also compare the models in accordance with their relationships with the index. According to Table 4, for S&P500 and KOSPI, the *Low* model improves total profit more than the *High* model does, while for HSI and EuroStoxx50, the *High* model improves total profit more than the *Low* model, irrespective of the technical measures to select stocks. Accordingly, when there are enough component stocks, such as S&P500 and KOSPI, it is more effective to use component stocks that have a low relationship with the index stock. However, if only a small amount of component stock, such as HSI and EuroStoxx50, exist, a high relationship with the index stock is more effective. In S&P500 and KOSPI, component stocks account for about 3% of total component stocks, while HSI and Eurostoxx50 account for more than 20%. Hence, we can claim that the S&P500 and KOSPI results are different from those of HSI and Eurostoxx50.

Consequently, we find that the NE Low model is the most effective for S&P500 and KOSPI, as it increases total profit 84% and 275% compared with their RL models. These results are 17% and 218% higher than the IDX models and 65% and 234% higher than the MKT models. Further, the NE High model is the most effective for HSI and EuroStoxx50, as it increases total profits by 93% and 153% compared with their RL models. These results are 25% and 49% higher than the IDX models, and 160% and five times higher than the MKT models. Thus, pretraining methods have been found to be help in improving the total profits. It is better to pretrain with selected component stocks than with the index, as we have delineated how the choice of component stocks affects the total profits.

4.2. Determining the number of shares

We introduce three different methods for making decisions about the number of shares to trade at each point in time: NumQ, NumDReg – AD, and NumDReg – ID. Each method is developed step by step. First, we compare two methods, NumQ and NumDReg – AD, with the same number of outputs in the number branch. In addition, we apply transfer learning to increase total profit and stabilize learning. The NE Low model for S&P500 and KOSPI and the NE High model for HSI and Eurostoxx50, which are shown as the most effective pretrained models in Section 4.1, are used to determine the models for the number of shares. As indicated below, they are described as a unified abbreviation, NE. The results are shown in Table 5 and Fig. 9.

As shown in Tables 4 and 5, the models that include trading with different numbers of shares show an increase in total profits. Total profits increase even without the pretrained model compared with the RL model, which is based on trading with fixed number of shares. The NQ and NDA models increase total profits from 3.4 to 12.4 times compared with the RL model by the indexes. In S&P500, total profits of the NQ and NDA models are 4.5 times and 3.4 times higher than the RL model. In KOSPI, total profits of the NQ and NDA models are 5.8 times and 4.6 times higher than the RL model. In HSI, total profits of the NQ and NDA models are 12.4 times and 11.4 times higher than the RL model. Finally, EuroStoxx50, total profits of the NQ and NDA models are 6.6 times and 5.2 times higher than the RL model.

Table 6
Results of the NENDI3 model and the models applying action strategies.

Model	S&P500	KOSPI	HSI	EuroStoxx50
NENDI3	9.6789	10.5704	10.4296	6.1502
NENDI3–BUY	9.1976	7.8825	11.5862	6.9175
NENDI3–HOLD	11.4188	13.1416	9.3136	9.3237
NENDI3–SELL	4.5838	10.7684	11.2926	−9.5000

After applying transfer learning, in the NENQ and NENDA models, the level of increase varies by the type of model. Nevertheless, all models show a distinct increase in total profits compared with the non-pretrained models, NQ and NDA. In S&P500, transfer learning increases total profits twice in the NQ model and 1.5 times in the NDA model. In KOSPI, it increases total profits 1.5 times in the NQ model and 1.2 times in the NDA model. In HSI, it increases total profits 1.8 times in the NQ model and 1.5 times in the NDA model. Finally, transfer learning increases total profits 1.4 times in the NQ model and 1.2 times in the NDA model. These findings show that transfer learning can still effectively increase total profits. Although adding the optimizing number network increases the instability of training, transfer learning can stabilize the network.

In Table 5, the NQ model outperforms the NDA model in most cases, whether pretrained or not, because the searching space of the latter is too large for simultaneous training. To address this problem, we divide the training process into three steps to reduce the searching space. Table 5 and Fig. 10 show the total profits of the NENDA3 model results in a higher profit without the stepwise training model, NENDA. The total profits increase twice in S&P500, 1.3 times in KOSPI, 1.5 times in HSI, and 1.4 times in EuroStoxx50. These results mean that stepwise training limits the searching space effectively and increases total profits. Further, the total profits exceed that of the NENQ model by about 13%, 47%, 26% and 8% in the S&P500, KOSPI, HSI and EuroStoxx50, respectively.

Finally, we construct an NENDI3 model to predict the layer numbers that are unaffected by the actions. As the output size decreases to one, the number is predicted regardless of the action. It only considers the market situation. Table 5 shows that the NENDI3 model increases total profits by 11%, 81%, 14%, and 44% for S&P500, KOSPI, HSI, and EuroStoxx50, respectively, compared with the NENDA3 models. This result shows 11 times, 19 times, 27 times, and 12 times better profits than for the RL model for S&P500, KOSPI, HSI, and EuroStoxx50, respectively.

4.3. Action strategies in a confused market

To postpone the buy or sell action in a confused market, we apply action strategies to the architecture. Our action strategies are defined as follows. When the difference between the probabilities of buy and sell is less than 0.2 (which means that the threshold is 0.2), we determine the action in accordance with the strategies. We apply action strategies to the NENDI3 model, and compare the strategies when the action strategy is BUY, HOLD, or SELL with the same threshold. Table 6 and Fig. 11 present the results.

In S&P500 and EuroStoxx50, the NENDI3–SELL model shows the lowest total profit, but it shows better profits than the NENDI3 model in KOSPI and HSI. The NENDI3–BUY model shows a similar total profit to the NENDI3 model. In this experiment, though the results can be affected by the trends of the index stocks, the action strategy of HOLD shows the highest total profit in most cases.

Fig. 9. Comparison of the NQ and ND models, and their transfer learning applications using S&P500.

Fig. 10. Comparison between NENDA, NENDA3, and NENDI3 models using S&P500.

Consequently, the action strategy of HOLD is most effective in avoiding incorrect decisions in uncertain situations. This action increases profit by 18%, 24%, and 49% in the of S&P500, KOSPI, and EuroStoxx50, respectively, compared with the prior model, NENDI3. This suggests that, when the market is uncertain, postponing the decision to buy or sell—that is to say, to *hold the stock*—is the most efficient way to increase total profit. The HOLD action trains the network to be stable when the model is confused while making decisions.

This method can also be expanded to reflect a trader's confidence in the market and the trader's propensity. In terms of the model's practical application, if a trader feels confident that the stock price will increase, the trader can choose the BUY strategy in confusing situations. Conversely, if a trader predicts that the stock price will decrease, as in a financial crisis, the trader can choose the SELL strategy. Moreover, if a trader is highly risk-averse, the trader can increase the threshold.

5. Conclusion

As we highlight in our study, challenges arise when a stock is traded using reinforcement learning. Financial data can be scarce and highly volatile; therefore, they include uncertain patterns. Consequently, using such data to make decisions is difficult. To reflect real-world decision making, we show that trading actions should avoid a discrete form, as in a fixed-share model. Instead, trading models should imitate the real world. We address this issue by proposing three new methods using four different stocks: the S&P500, KOSPI, HSI, and EuroStoxx50.

First, we use the DNN regressor with stepwise training to change the number of trading shares. We add a separated branch to the DQN to determine the number of shares. To reduce the searching space, we use stepwise training. The regression model then determines the number of shares to trade. We predict the number of shares regardless of the trading action. This method in-

Fig. 11. Comparison of the action strategies using S&P500.

creases trading total profit by four times in S&P500, five times in KOSPI, 12 times in HSI, and six times in EuroStoxx50 compared with the fixed-number trading system. We not only increase total profit, but also improve the model's ability to adapt to the real market.

Second, we propose various action strategies to prepare for an uncertain market, and in particular, the HOLD action with a threshold of 0.2. Postponing the decision to buy or sell controls a confusing situation effectively and increases total profits by 18% in S&P500, 24% in KOSPI, and 49% in EuroStoxx50. This method can be explained by a simple formula. As a result, compared with the original reinforcement learning model, all three methods increase total profits for the test period by 13 times in S&P500, 24 times in KOSPI, 30 times in HSI, and 18 times in EuroStoxx50.

Third, we choose several component stocks that have some relationship with the index stock according to the predicting error of the neural network to pretrain the model. When the amount of component stocks is small, we choose six component stocks that are highly related to their index. When the amount of component stocks is large, we choose ten component stocks with low relationship to their index. This approach prevents overfitting and provides more information to enable a greater understanding of the financial data. This method increases trading total profits twofold in S&P500, 3 times in KOSPI, twofold in HSI, and 2.5 times in EuroStoxx50.

Further, the problem of insufficient financial data can be handled through the transfer-learning approach. This solution shows how we can use financial data efficiently for deep learning. Predicting the number of shares can facilitate decisions in the real market, where traders regularly deal with differing numbers of shares. This method can also be extended to a hedge strategy, where the number of shares traded is important. The action strategy also shows how various investors' propensities can be reflected in the automated algorithm.

We believe that our model has many applications—for example, it could be combined with other financial models, such as the multi-asset portfolio optimization model, to identify hedge strategies. Overall, our model has clear implications for traders, who can improve their total profits. In addition, the model extends the literature on operations research and decision making in the financial domain and provides a foundation for further, in-depth analysis.

Nonetheless, our study has limitations. For instance, our proposed model only analyzes data of closing trade prices. Further, other market factors, such as volatility, should be considered to predict the number of shares and to select trading actions more effectively. Moreover, we only consider one stock. Thus, the study could be extended to a portfolio of stocks. We leave these tasks for future research.

Acknowledgments

This work was supported by the Technology Advancement Research Program funded by the Korean Ministry of Land, Infrastructure, and Transport (grant number 17CTAP-C129782-01).

CRediT author statement

Gyeeun Jeong: Writing–Original Draft, Software, Methodology, Data Curation, Formal Analysis, Investigation, Visualization **Ha Young Kim:** Writing–Review & Editing, Supervision, Conceptualization, Methodology, Formal Analysis, Investigation, Project Administration, Resources, Funding Acquisition.

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. doi:10.1016/j.eswa.2017.06.023.

Ang, K. K., & Quek, C. (2006). Stock trading using RSPOP: A novel rough set-based neuro-fuzzy approach. *IEEE Transactions on Neural Networks, 17*(5), 1301–1315. doi:10.1109/TNN.2006.875996.

Aquino, C. F., Chamhum Salomão, L. C., & Azevedo, A. M. (2016). High-efficiency phenotyping for vitamin A in banana using artificial neural networks and colorimetric data, 75(3), 268–274. 10.1590/1678-4499.467

Bellman, R. (1957). A Markovian decision process. *Journal Of Mathematics And Mechanics*. doi:10.1007/BF02935461.

Bergstra, J., & Yoshua Bengio, U. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research, 13*, 281–305. doi:10.1162/153244303322533223.

Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy, 81*(3), 637–654. doi:10.1086/260062.

Brock, W., Lakonishok, J., & LeBaron, B. (1992). Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance, 47*(5), 1731–1764. doi:10.1111/j.1540-6261.1992.tb04681.x.

Brownlees, C. T., Cipollini, F., & Gallo, G. M. (2011). Intra-daily volume modeling and prediction for algorithmic trading. *Journal of Financial Econometrics, 9*(3), 489–518. doi:10.1093/jjfinec/nbq024.

Campbell, J. Y., Grossman, S. J., & Wang, J. (1993). Trading volume and serial correlation in stock returns. *The Quarterly Journal of Economics, 108*(4), 905–939.

Chaboud, A. P., Chiquoine, B., Hjalmarsson, E., & Vega, C. (2009-2014). Rise of the machines: Algorithmic trading in the foreign exchange market. *The Journal of Finance, 69*(5), 2045–2084. doi:10.1111/jofi.12186.

Chavarnakul, T., & Enke, D. (2008). Intelligent technical analysis based equivolume charting for stock trading using neural networks. *Expert Systems with Applications, 34*(2), 1004–1017. doi:10.1016/j.eswa.2006.10.028.

Chen, G. M., Firth, M., & Rui, O. M. (2001). The dynamic relation between stock returns, trading volume, and volatility. *Financial Review, 36*(3), 153–174. doi:10.1111/j.1540-6288.2001.tb00024.x.

Cotter, A., Shamir, O., Srebro, N., & Sridharan, K. (2011). Better mini-batch algorithms via accelerated gradient methods. In *Advances in neural information processing systems* (pp. 1647–1655).

DeLong, J. B., & Magin, K. (2006). A short note on the size of the dot-com bubble. *NBER Working Paper No. 12011*, 1–14. 10.3386/w12011

Dempster, M. A. H., & Leemans, V. (2006). An automated FX trading system using adaptive reinforcement learning. *Expert Systems with Applications, 30*(3), 543–552. doi:10.1016/j.eswa.2005.10.012.

Dempster, M. A. H., Payne, T. W., Romahi, Y., & Thompson, G. W. P. (2001). Computational learning techniques for intraday FX trading using popular technical indicators. *IEEE Transactions on Neural Networks, 12*(4), 744–754. doi:10.1109/72.935088.

Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2017). Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems, 28*(3), 653–664. doi:10.1109/TNNLS.2016.2522401.

Domowitz, I., & Yegerman, H. (2006). The cost of algorithmic trading. *The Journal of Trading, 1*(1), 33–42. doi:10.3905/jot.2006.609174.

Duchi, J., & Singer, Y. (2009). Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research, 10*, 2899–2934. doi:10.1561/2400000003.

Dymova, L., Sevastianov, P., & Bartosiewicz, P. (2010). A new approach to the rule-base evidential reasoning: Stock trading expert system application. *Expert Systems with Applications, 37*(8), 5564–5576. doi:10.1016/j.eswa.2010.02.056.

Fama, E. F. (1995). Random walks in stock market prices. *Financial Analysts Journal, 51*(1), 75–80. doi:10.2469/faj.v51.n1.1861.

Fama, E. F., & Blume, M. E. (1966). Filter rules and stock-market trading. *The Journal of Business, 39*(1), 226–241.

Feyzmahdavian, H. R., Aytekin, A., & Johansson, M. (2016). An asynchronous mini–batch algorithm for regularized stochastic optimization. *IEEE Transactions on Automatic Control, 61*(12), 3740–3754.

Flood, M. D. (2012). A brief history of financial risk and information. In M. Brose, M. Flood, D. Krishna, & W. Nichols (Eds.). In *Handbook of financial data and risk information: 1* (pp. 1–32). Cambridge University Press. 2014Retrieved from:. doi:10.2139/ssrn.2150987.

Ghosh, S., Laksana, E., Morency, L.-P., & Scherer, S. (2016). Representation learning for speech emotion recognition, (September), 3603–3607. 10.21437/Interspeech.2016-692

Gorse, D., & Street, G. (2011). Application of stochastic recurrent reinforcement learning to index trading. In *Proceedings of the european symposium on artificial neural networks, computational intelligence and machine learning ESANN 2011* (pp. 27–29). Retrieved from:. http://discovery.ucl.ac.uk/id/eprint/1339314 .

Heaton, J. B., Polson, N. G., & Witte, J. H. (2017). Deep learning for finance: Deep portfolios. *Applied Stochastic Models in Business and Industry, 33*(1), 3–12. doi:10.1002/asmb.2209.

Hendershott, T., Jones, C. M., & Menkveld, A. J. (2011). Does algorithmic trading improve liquidity. *Journal of Finance, 66*(1), 1–33. doi:10.1111/j.1540-6261.2010.01624.x.

Hendershott, T., & Riordan, R. (2011). *Algorithmic trading and information*. Berkeley: University of California *Working Paper*. doi:10.2139/ssrn.1472050.

Hu, Y., Liu, K., Zhang, X., Su, L., Ngai, E. W. T., & Liu, M. (2015). Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review. *Applied Soft Computing, 36*, 534–551. doi:10.1016/j.asoc.2015.07.008.

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

Karpoff, J. M. (1987). The relation between price changes and trading volume: A survey. *Journal of Financial and quantitative Analysis, 22*(1), 109–126.

Kaelbling, L., Littman, M., & Moore, A. (1996). Reinforcement learning: A Survey. *Journal of Artificial Intelligence Research, 4*, 237–285. doi:10.1613/jair.301.

Kindleberger, C. P., & Aliber, R. Z. (2005). *Manias, panics and crashes*. John Wiley *3rd. Ed*. doi:10.1057/9780230628045.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *ArXiv, 1412.6980*, 1–15. https://doi.org/http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503.

Kuo, R. J., Chen, C. H., & Hwang, Y. C. (2001). An intelligent stock trading decision support system through integration of genetic algorithm based fuzzy neural network and artifial neural network. *Fuzzy Sets and Systems, 118*(1), 21–45. doi:10.1016/S0165-0114(98)00399-6.

Levich, R. M., & Thomas, L. R. (1993). The significance of technical trading-rule profits. *Journal of International Money and Finance, 12*(5), 451–474. doi:10.1016/0261-5606(93)90034-9.

Li, M., Zhang, T., Chen, Y., & Smola, A. J. (2014). Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 661–670). ACM. August.

Lin, X., Yang, Z., & Song, Y. (2011). Intelligent stock trading system based on improved technical analysis and Echo State Network. *Expert Systems with Applications, 38*(9), 11347–11354. doi:10.1016/j.eswa.2011.03.001.

Lu, Z., Zhu, Y., Pan, S. J., Xiang, E. W., Wang, Y., & Yang, Q. (2014). Source free transfer learning for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 122–128). Retrieved from. https://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8361 .

Mastinšek, M. (2006). Discrete-time delta hedging and the Black-Scholes model with transaction costs. *Mathematical Methods of Operations Research, 64*(2), 227–236. doi:10.1007/s00186-006-0086-0.

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. doi:10.1038/nature14236.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. et al. (2013). Playing atari with deep reinforcement learning. *ArXiv*, 1312.5602, 1–9. 10.1038/nature14236

Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks, 12*(4), 875–889. doi:10.1109/72.935097.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning* (pp. 807–814). 10.1.1.165.6419.

Nevmyvaka, Y., Feng, Y., & Kearns, M. (2006). Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning* (pp. 673–680). ACM. June.

Oh, K. J., & Kim, K. J. (2002). Analyzing stock market tick data using piecewise nonlinear model. *Expert Systems with Applications, 22*(3), 249–255. doi:10.1016/S0957-4174(01)00058-6.

O, et al. (2006). Adaptive stock trading with dynamic asset allocation using reinforcement learning. *Information Sciences, 176*(15), 2121–2147. doi:10.1016/j.ins.2005.10.009.

Pan, S., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering, 22*(10), 1345–1359. doi:10.1109/TKDE.2009.191.

Pendharkar, P. C., & Cusatis, P. (2018). Trading financial indices with reinforcement learning agents. *Expert Systems with Applications, 103*, 1–13. doi:10.1016/j.eswa.2018.02.032.

Scholtus, M., Van Dijk, D., & Frijns, B. (2014). Speed, algorithmic trading, and market quality around macroeconomic news announcements. *Journal of Banking and Finance, 38*(1), 89–105. doi:10.1016/j.jbankfin.2013.09.016.

Shalev-Shwartz, S., Singer, Y., & Ng, A. Y. (2004). Online and batch learning of pseudo-metrics. In *Proceedings of the twenty-first international conference on machine learning - ICML '04* 94.. doi:10.1145/1015330.1015376.

Sudo, a, Sato, a, & Hasegawa, O. (2009). Associative memory for online learning in noisy environments using self-organizing incremental neural network. *Neural Networks, IEEE Transactions On, 20*(6), 964–972. doi:10.1109/TNN.2009.2014374.

Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks, 9*(5). 1054–1054. doi: 10.1109/TNN.1998.712192.

Taghavifar, H., Taghavifar, H., Mardani, A., & Mohebbi, A. (2014). Modeling the impact of in-cylinder combustion parameters of di engines on soot and NOx emissions at rated EGR levels using ANN approach. *Energy Conversion and Management, 87*, 1–9. doi:10.1016/j.enconman.2014.07.005.

Tan, Z., Quek, C., & Cheng, P. Y. K. (2011). Stock trading with cycles: A financial application of ANFIS and reinforcement learning. *Expert Systems with Applications, 38*(5), 4741–4755. doi:10.1016/j.eswa.2010.09.001.

Vijayakumar, S., D 'souza, A., & Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Computation, 17*, 2602–2634. doi:10.1162/089976605774320557.

Wang, Y., Wang, D., Zhang, S., Feng, Y., Li, S., & Zhou, Q. (2017). *Deep Q-trading*. http://cslt.riit.tsinghua.edu.cn.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3-4), 279–292. doi:10.1007/BF00992698.

Wu, D., Lance, B. J., & Parsons, T. D. (2013). Collaborative filtering for brain-computer interaction using transfer learning and active class selection. *PLoS ONE, 8*(2). doi:10.1371/journal.pone.0056624.

Yadav, Y. (2015). How algorithmic trading undermines efficiency in capital markets. *Vand. L. Rev., 68*, 1607.

Yang, S., Paddrik, M., Hayes, R., Todd, A., Kirilenko, A., Beling, P., et al. (2012). Behavior based learning in identifying high frequency trading strategies. In *Proceedings of the 2012 IEEE conference on computational intelligence for financial engineering & economics (CIFEr)* (pp. 1–8). doi:10.1109/CIFEr.2012.6327783.