Applied Analysis and Optimization
Volume 9, Number 3, 2025, 373–387

Yokohama Publishers
ISSN 2432-1664 Online Journal

STOCK TRADING PREDICTION BY EPISODIC MEMORY DEEP Q-NETWORKS

ZIYI WU, DONGHAO XIN, XUECHUAN XIAO, AND XIANCHAO ZHU

Abstract. This paper proposes a stock prediction method based on Episodic Memory Deep Q-Network (EMDQN), which aims to improve the accuracy and speed of stock price prediction while reducing the convergence time of the DQN loss function by incorporating episodic memory. Contrast experiments were conducted on the historical daily price and trading volume data of U.S. stocks and ETFs using reinforcement learning algorithms, including DQN, Dueling Double Deep Q Network (D3QN), Proximal Policy Optimization (PPO), and EMDQN. The results verified the effectiveness of EMDQN in short-term stock price prediction. The experimental results show that the outputs of EMDQN in both training and testing phases are highly consistent with actual stock prices, and it outperforms other algorithms in several key indicators, including prediction accuracy, runtime efficiency, and loss control. Therefore, the method proposed in this paper provides investors with a more valuable decision-making basis, further exploring the potential and application prospects of reinforcement learning in stock market prediction.

1. Introduction

Stock price fluctuations in financial markets resemble complex, ever-changing waves, capturing the attention of investors and financial analysts alike. An effective stock-trading strategy offers investors as much profit and as little risk as possible [23]. However, this remains a highly challenging task [19]. Stock prices exhibit high volatility, and the factors influencing them are intricate and complex. Economic indicators, such as consumer supply and demand fluctuations or the commodity price index, are already complex. With global conditions and investor behavior, stock price changes become even more unpredictable. At the same time, the influence of global conditions and investor behavior makes stock price changes even more elusive [4]. Interwoven factors with unclear correlations make identifying key influences on stock prices and building effective prediction models challenging.s garnered widespread attention and applied across multiple domains. Value investment theory posits that the company's market value determines a stock's actual value, but stock prices often deviate from rational expectations. The efficient market hypothesis even posits that predicting stock values is impossible, and historical data predictions usually fail due to the volatility and dynamic nature of stock prices [5,8]. However, technical analysis indicates that historical stock price movements are crucial for predicting future stock prices [10, 21]. The advent of machine learning technology has brought new hope for stock price prediction [1,6]. Research on algorithmic trading based on machine learning has been increasing in recent years [11].

2010 *Mathematics Subject Classification.* 68Q32, 68T05.
Key words and phrases. Stock trading prediction, reinforcement learning, episodic memory.

Machine learning can identify valuable data and inductively derive patterns [12], especially adept at handling nonlinear problems. Deep learning excels in extracting key information from nonlinear time series, and its integration in stock prediction is of significant importance, enhancing prediction accuracy [3]. However, some traditional machine learning models, such as feedforward and recurrent neural networks, still fall short when dealing with unstable time series and long-term autoregressive data [13]. Stock trading strategies pose challenging machine learning applications for significant commercial yields in the finance industry [9]. As a subfield of deep learning, reinforcement learning stands out due to its unique strategic advantages. Many studies have shown the effectiveness of using deep reinforcement learning to learn profitable trading strategies from financial market data [7]. Unlike other fields, it aims to seek profit maximization and adapt to environmental changes. In stock price prediction, reinforcement learning's sample efficiency categorization provides various options for different methods, with applications in solving discrete region optimization problems and computing parameterized strategies [20]. Reinforcement learning (RL) has great potential and advantages in the feasibility of financial markets, stock market prediction, and intelligent decision-making mechanisms [14]. Numerous studies have applied reinforcement learning to stock trading prediction and trading model construction [17]. For example, when working with financial time series data, DQN can identify and exploit hidden patterns and trends in historical information to predict future market dynamics [2]. However, it still faces challenges like long convergence times for loss functions and tremendous loss values. This paper proposes an episodic memory-based DQN stock prediction method, the Episodic Memory Q Network [15], which combines the advantages of episodic memory and DQN, significantly improving sample efficiency and learning speed. This method not only quickly identifies and adopts good action strategies but, compared to DQN, requires fewer interactions to achieve industry-leading performance.

Furthermore, EMDQN provides a direct method to alleviate the problem of Q-value overestimation, a common issue in Q-learning-based agents [16]. Integrating scenario memory information into the parameterized model combines the generalization ability of DQN with the fast convergence characteristics of scenario control. This method aims to reduce the algorithm's loss function convergence time, enhance the accuracy and speed of stock trading volatility prediction, address the challenges in volatile markets, and provide more valuable decision-making references for investors. It also seeks to explore further the potential and application prospects of reinforcement learning in stock market prediction.

2. Background

2.1. **Reinforcement Learning.** As an essential branch of machine learning, RL studies how agents learn through interactions with the environment to make optimal decisions, thereby maximizing cumulative rewards [24]. This problem can be formalized as a Markov Decision Process (MDP), which includes a state space S, an action space A, a reward space R, and a transition probability function $P(S'|S, A)$ that describes the probability of transitioning from one state to another [22]. According to Figure 1, the state space of the environment is represented by S, encompassing all possible states that the agent may occupy; the action space A defines the set of

actions the agent can take in each state; the reward space R quantifies the feedback the agent receives from the environment after performing an action. At the time step t, the agent observes the environmental state $s \in S$ and then selects an action a from the action space A. The environment then transitions from the agent's current state to the next state $s' \in S$ based on the action. This action leads to the agent receiving a scalar reward $r_t \in R$. This process can be represented as: $s \xrightarrow{a} (r_t, s')$, The ultimate goal of the agent is to maximize long-term rewards, which are defined as: $R_t = \sum_{k=0}^{\infty} \gamma r_{t+k}$, where $\gamma \in [0, 1)$ is a discount factor that determines the importance of future rewards relative to current rewards. The closer the discount factor γ is to 1, the more inportant the future rewards are, the closer γ is to 0, the less important the future rewards are. To find the optimal policy, the agent needs to learn a policy $\pi : S \to A$ that specifies an optimal action $a \in A$ for each state $s \in S$ to maximize long-term returns starting from that state.

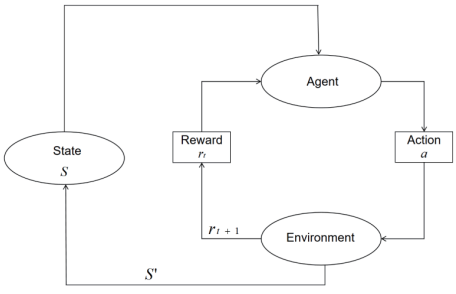


FIGURE 1. The flowchart of reinforcement learning.

2.2. **Deep Q-Networks (DQN).** In the field of RL, DQN has made significant breakthroughs, using deep neural networks to approximate the Q-function $Q(s, a; \theta)$, where θ represents the network parameters, and (s, a) denotes state-action [25]. DQN employs two networks with the same structure but different parameters. One is the main network (MainNet) for optimization, while the other is the target network (TargetNet) used to compute the Q target. In Figure 2, the target network calculates the target Q-value through its parameters θ'. The main network produces the Q-value prediction $Q(s, a; \theta)$ when the agent takes actions a in a given state s. In contrast, the target network is used to compute the $\max_{a'} Q(s', a'; \theta')$, where s' is the next state, and parameters θ' correspond to the target network's parameters. The parameters of the target network are periodically copied from the main network, a mechanism designed to enhance learning stability. Specifically, this approach helps prevent training instability that could arise from rapid fluctuations in network parameters during the calculation of the loss function. DQN stores the agent's experience ((s, a, r, s') in the form of tuples) in a replay buffer and randomly samples these experiences during training to update the network. This helps decorate samples and prevents rapid forgetting. The parameters of the neural network are optimized using stochastic gradient descent to minimize the loss $(r + \gamma \max_{a'} Q_{\theta'}(s', a') - Q_{\theta}(s, a))$. Here, θ represents the parameters of the target network, and θ' represents the parameters of the main network. r is the

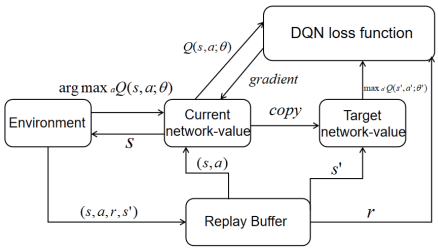


FIGURE 2. The algorithm flow of DQN.

immediate reward obtained after taking an action at time step t. γ is the discount factor used to measure the importance of future rewards in current decision-making. $Q_{\theta'}(s,a)$ is the Q-value prediction when the main network takes action a in state s. $\max_{a'} Q(s',a';\theta')$ represents the maximum Q-value estimate for taking any action a' in the next state s', using the target network parameters θ. By continuously adjusting the network parameters θ, the predicted Q-values gradually approach the target Q-values, thereby improving the agent's decision-making ability [18].

2.3. **Episodic Memory Deep Q-Networks.** In stock trading prediction using DQN, existing gradient-based optimization methods like stochastic gradient descent are inefficient, causing long training times and slow parameter-updating from experiences, thus low data efficiency. Improving training efficiency and accuracy of DQN algorithm is crucial. We introduce Episodic Memory Q Network (EMDQN). Its core is using episodic memory to aid algorithm learning. MDQN accelerates DQN's learning, especially in solving problems such as slow reward propagation, single learning mode, and low sample efficiency. Our goal is to address the following issues in DQN by introducing episodic memory:

Addressing the issue of slow reward propagation:

EMDQN overcomes this by using the maximum return from episodic memory to spread rewards from the best trajectory to parameters without high variance. Combined with traditional TD targets, it can rapidly propagate unbiased MC returns while taking advantage of TD targets' low variance, thus solving the slow reward propagation problem and speeding up stock prediction.

Regarding the issue of single learning mode:

EMDQN integrates DQN, which mimics the striatum, with episodic memory that imitates the hippocampus, establishing two learning systems for the agent: the inference target S and the memory target H. By leveraging a novel loss function to balance S and H, EMDQN flexibly applies both learning methods during training. This approach better aligns with the brain's mechanisms, enhancing the learning process and improving the accuracy of stock predictions.

Efficiency Problem:

EMDQN introduces a new mechanism that extracts the best returns from episodic memory during training and integrates them into the neural network. It alters the update targets of each sample, making sample utilization more efficient. Unlike prioritized experience replay, which changes the priority of sample updates, it handles

non-zero rewards in samples more effectively, improving sample efficiency, reducing the number of interactions with the environment, accelerating convergence, and speeding up stock prediction.

2.4. **Mechanism of Episodic Memory Deep Q Network (EMDQN).** EMDQN is inspired by the decision-making mechanisms of the human brain, partially simulating the functions of the striatum and hippocampus. The system simulating the striatum estimates action values through neural networks, providing the agent with an inference goal S, calculated as: $S(s_t, a_t) = r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$. Here, r_t is the immediate reward obtained after taking action a_t at time t, γ is the discount factor used to measure the importance of future rewards in current decision-making, $Q_\theta(s_{t+1}, a')$ representing the action value estimate when different actions a' are taken in state s_{s+1}. The system simulating the hippocampus provides the memory goal H, defined as: $H(s_t, a_t) = \max_i R_i(s_t, a_t), \quad i \in \{1, 2, \cdots, E\}$, where E denotes the number of episodes the agent has experienced, $R_i(s_t, a_t)$ is the future return after taking an action a_t in state s_t during the i episode. The memory goal H is constructed as an ever-growing table indexed by state-action pairs. Throughout each episode, transition tuples (s, a, r) along the agent's trajectory are cached. Once the episode concludes, these transition tuples are processed reversely to update H. To integrate these two learning systems, EMDQN proposes a new loss function:

$$L = \alpha(Q_\theta - S)^2 + \beta(Q_\theta - H)^2 \tag{2.1}$$

The value function Q_θ, determined by the neural network parameters θ, plays a key role in the agent's decision-making process. α and β are weight parameters adjusting relative importance to the two learning objectives. Further, $\lambda = \frac{\beta}{\alpha}$ is defined as the relative weight of S and H, which rewrites the loss function as:

$$\min_\theta \sum_{(s_i, a_i, r_i, s_{i+1}) \sim D} [(Q_\theta(s_i, a_i) - S_\theta(s_i, a_i))^2 - \lambda(Q_\theta(s_i, a_i) - H_\theta(s_i, a_i))^2] \tag{2.2}$$

where D indicates a mini-batch of experience data sampled from the experience replay buffer.

During training, certain state-action pairs (s, a) may not yet have been added to the scenario memory table, meaning the value of $H(s, a)$ cannot be obtained. To address this, EMDQN adopts a simple and reasonable approach. When the corresponding (s, a) pair cannot be found in the scenario memory table, the contribution of the memory target H to the loss function is ignored. This approach aligns with human cognitive logic, as people generally do not form memories of events they have not experienced. Therefore, it is reasonable to disregard these missing memory targets during training.

The EMDQN network architecture: The state s is input into a convolutional neural network for feature extraction and then passed through two fully connected layers to compute $Q_\theta(s, a)$. This architecture is consistent with the original DQN. When calculating $Q_\theta(s, a)$, the state s is multiplied by a matrix randomly sampled from a Gaussian distribution and projected into a low-dimensional vector h. This vector h is used to search for the corresponding memory target value $H(s, a)$ in

the memory table, which regularizes $Q_\theta(s, a)$ and guides the learning process of the neural network. To improve the search efficiency of the memory table, EMDQN uses a kd-tree data structure to organize the memory table. During each episode, all experienced tuples $(\phi(s), a, r)$ are caught. Like the target network in DQN, EMDQN maintains a target memory table, which is updated every K training step using the previously cached transition data. This ensures relatively stable memory target values and helps prevent training instability.

EMDQN employs the random projection technique to represent the state compactly, projecting the original high-dimensional state space into a lower-dimensional vector space. In practical applications, EMDQN, according to the Johnson - Lindenstrauss lemma, uses random projection operation only the vectors after random projection for exact match search, rather than using methods like NEC and MFEC for k-nearest neighbor (KNN) search to estimate value. This exact match search method allows the state to be projected into a lower-dimensional vector space, greatly accelerating the lookup speed of the memory table and improving training efficiency. For newly appearing state-action pairs, EMDQN adds their corresponding key-value pairs to the memory table; for state-action pairs already in the memory table, it updates their values based on the future return $R(s_t, a_t)$ (Monte Carlo return) calculated from the current episode, with the update rule being.

$$H(s_t, a_t) = \begin{cases} \max\{H(s_t, a_t), R(s_t, a_t)\}, & \text{if } (s_t, a_t) \in H \\ R(s_t, a_t) \end{cases} \tag{2.3}$$

In this way, the memory table continuously records and updates the optimal return value for each state-action pair, providing more valuable information for the agent ' s learning.

During the training process, EMDQN calculates the gradient of the loss function concerning the neural network parameters θ through backpropagation, thereby updating the parameters to optimize network performance. When computing the gradient, the contributions of both the value-guided target (inference target S) and the memory target H to the loss function are considered, and their gradient information is backpropagated into the neural network simultaneously, enabling the network to learn and adjust its parameters from both learning objectives. To prevent issues such as gradient explosion or vanishing, which could lead to unstable training, EMDQN clips the gradients of the two terms in the loss function:

$$(Q(s_t, a_t) - S(s_t, a_t))^2 \ (Q(s_t, a_t) - H(s_t, a_t))^2$$

limiting them to the range of $[-1, 1]$. This gradient clipping operation helps stabilize the training process, ensuring that the neural network parameters are updated within a reasonable range, thereby improving training stability and convergence.

3. The design of the stock forecasting system

3.1. **Dataset Source.** This dataset is sourced from the "Huge Stock Market Dataset" published by Boris Marjanovic on the Kaggle platform and is licensed under the CC0 Public Domain license. It contains historical data for U.S. stocks and Exchange-Traded Funds (ETFs) traded on the New York Stock Exchange (NYSE), NASDAQ,

and NYSE MKT. For this experiment, we selected market data for financial products covering the period from April 11, 2014, to November 10, 2017 (This is the dataset used in the algorithm comparison experiment, along with additional datasets employed to verify the generalization ability of EMDQN). The dataset includes daily information on the opening trade, highest price, lowest price, closing price, trading volume, and open interest.

To evaluate the generalization ability of the trading decision algorithm, we randomly select stock data from different time periods to create the test datasets. (Profit is calculated by subtracting the buy price from the closing price at the time of sale, and the result is accumulated)

3.2. **Stock Model Design.** The stock price prediction model is based on discrete time series data, where stock prices are influenced by multiple factors. A multidimensional time series can be constructed using daily transaction prices and volumes. The goal of the model is to predict future stock prices using historical data.

3.3. **Reinforcement Learning Environment Design.** The Environment class simulates the stock trading environment, allowing the model to learn optimal trading strategies through interaction with the environment.

Initialization: The constructor init takes stock price data (data) and the length of the historical time window (history_t). The default value of history_t is 90, which means the last 90 days of price changes are considered.

Reset environment: The reset method initializes the environment state, including the time step t, done flag, profits, positions, position value, and historical price changes.

Execute trade: The step method executes a trade based on the input action (act) and updates the environment state:

If act = 1, it represents a buy action, and the current closing price is added to positions.

If act = 2, it represents a sell action, calculating the total profit from the stocks in the positions and updating profits and positions.

Reward mechanism: The reward is calculated based on the trade result: 1. The buy action does not generate an immediate reward. 2. The sell action calculates profit based on the difference between the sell and buy prices. If profit is made, reward = 1; if no profit or a loss occurs, reward = -1.

State update: The time step t, position_value, and historical price changes (history) are updated after each trade. The environment stops if the end of the datasets is reached or the done flag is true.

Profit: Profit is calculated by subtracting the buy price from the closing price at the time of sale, and the result is accumulated.

3.4. **Hyperparameter Design.** To ensure the validity of the experiment, all hyperparameters remain unchanged, and only the network architecture is modified. This approach follows the controlled-variable method, ensuring that any differences in the experimental results are primarily due to the algorithms ' intrinsic characteristics, thereby facilitating an objective comparison of their performance. We set the hyperparameters as shown in Table 1.

Table 1 Introduction to hyperparameters of the experiment.

hyperparameters	Value	Parameter description
gamma	0.98	discount factor
epoch_number	50	training iterations
memorysize	300	experience replay buffer capacity
batchsize	25	samples per training batch
hiddensize	100	number of hidden layer neurons
epsilon	1.0	initial exploration rate
epsilondecrease	5e-4	exploration rate decayr
epsilonmin	0.1	minimum exploration rate
startreduceepsilon	200	steps before exploration rate decay
updateqfreq	20	target network update frequency
trainfreq	10	training frequency

3.5. **DQN Training.** In stock model design, we use the DQN algorithm to train the agent's trading strategy. DQN is a reinforcement learning algorithm that combines Q-learning with deep neural networks, enabling it to learn strategies in complex environments.

Q-Network structure: A QNetwork class is defined as a three-layer feedforward neural network with two hidden layers. The input layer size is env.historyt + 1, representing the historical time steps plus position value; the hidden layer size is 100, and the output layer size is 3, corresponding to three possible actions (buy, hold, sell).

Training process: Initialize the QNetwork and target network Qast, as well as the Adam optimizer. Set up a memory buffer to store state transition tuples (pobs, pact, reward, obs, done). Through interaction with the environment, the agent executes actions and collects experiences, which are stored in the memory buffer. Periodically, a random batch of samples is drawn from memory to train and update the QNetwork's weights. Experience replays and target networks are used to stabilize and accelerate the training process.

Training results: After each episode, the accumulated reward, loss, and training time of the agent are recorded and printed. After multiple iterations, the agent learns the strategy of selecting the optimal action in different states to maximize the accumulated reward.

3.6. **D3QN Training.** In designing the stock model, we adopted the D3QN (Double Dueling DQN) algorithm to evaluate the training of the agent's trading strategy. This algorithm introduces two key improvements over DQN: a dueling network architecture and a double Q-learning mechanism. Decoupling state value estimation from action advantage and separating action selection from value evaluation enhances the stability and efficiency of policy learning in complex market environments.

The hyperparameters and training process of D3QN are kept identical to those of DQN:

Experience Replay: The memory buffer capacity is set to memory_size=300, and the batch training size is batch_size=25. Historical experiences are randomly sampled at regular intervals to reduce data correlation.

Training Frequency: Experience replay training occurs every 10 steps, and target network parameters are synchronized every 20 steps, ensuring that both algorithms learn at the same frequency of environmental interaction.

3.7. **PPO Training.** In the experimental design of the stock model, we introduce the PPO (Proximal Policy Optimization) algorithm to train the agent's trading strategy. As a policy gradient-based reinforcement learning method, PPO balances the magnitude of policy updates by optimizing the "proximal policy objective," demonstrating superior performance in continuous action spaces and complex sequential decision-making tasks. This makes it well suited for the high-volatile environment and the multi-step dependency characteristics of the stock market.

Input Layer: The input size is set to env.history_t + 1, receiving environmental state information that incorporates both historical time steps and position values.

Hidden Layers: These are composed of two fully connected layers, each with 100 neurons, using the ReLU activation function for nonlinear transformation. This configuration is identical to the hidden layer settings used in previous experiments, ensuring experimental controllability.

Policy Network (Actor): The output layer contains three neurons, which generate action probability distributions using the Softmax function. These correspond to three trading actions: buy, hold, and sell.

Value Network (Critic): The output layer has one neuron, which estimates the value of the current state.

3.8. **EMDQN Training.** Considering no influence from other factors, the Q-Network structure is largely the same as DQN, and EMDQN also uses a three-layer feedforward neural network.

Training process: Initialize the Q_Network and target network Q_target and the Adam optimizer. Set up a memory buffer to store state transition tuples $(s, a, r, s', done)$ and introduce episodic memory based on episodes (episodic memory) to store the maximum reward for each state-action pair. Through interaction with the environment, the agent executes actions and collects experiences stored in the memory buffer. Periodically, a random batch of samples is drawn from the memory buffer for training, updating the Q_Network's weights, and optimizing the learning process by incorporating episodic memory.

4. Experimental results

In Figure 3, The rewards of all algorithms fluctuate throughout the training process. Among them, EMDQN's reward shows a consistent upward trend and is higher than the rewards of DQN, D3QN, and PPO in most cases. Specifically, in the later stages of training, EMDQN's reward advantage becomes more pronounced, indicating that EMDQN outperforms the other algorithms in terms of both prediction accuracy and reward acquisition.

In Figure 4, The loss comparison chart shows that the losses of DQN and D3QN exhibit significant peaks and large fluctuations over multiple epochs. In contrast,

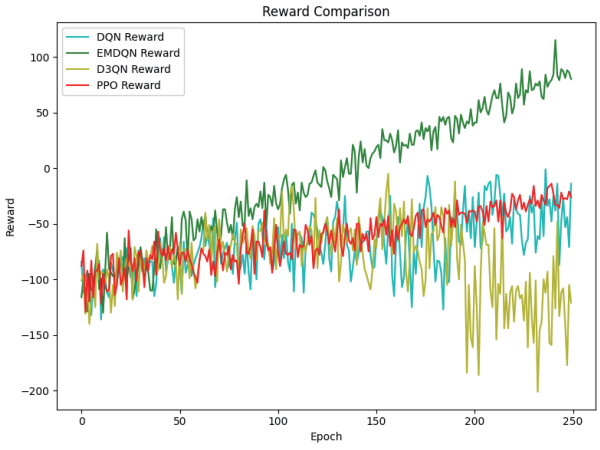
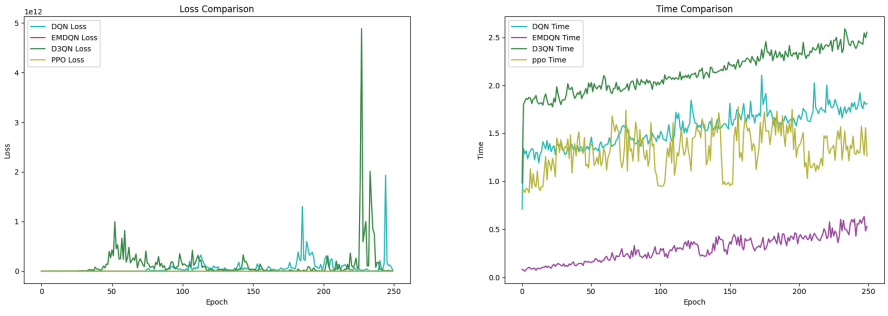


FIGURE 3. Comparison of Reward for different methods.

EMDQN's loss displays relatively smaller fluctuations, while PPO's loss remains consistently low with minimal variation. This suggests that, in terms of loss control, EMDQN has a clear advantage over DQN and D3QN, as it is better able to manage prediction error loss. The time comparison chart reveals that the runtime per epoch for D3QN is relatively long, while the runtimes for DQN and PPO fluctuate significantly and tend to be higher. In contrast, the runtime for EMDQN remains consistently low with only small variations. This demonstrates that EMDQN has a clear advantage in runtime efficiency, completing each training epoch more efficiently than the other algorithms.



(a) Loss (b) Runtime

FIGURE 4. Comparison of runtime and loss for different methods.

In Figure 5, EMDQN demonstrates higher rewards and profits on training and testing data than other algorithms, further proving that EMDQN excels in overall prediction performance.

In Figures 6, Figure 7, and Figure 8, the blue curve represents a simple long-term investment strategy, where stocks are initially purchased, held throughout

(a) D3QN (b) DQN

(c) EMDQN (d) PPO

FIGURE 5. Training and testing comparison results for different methods.

the investment period without being sold, and then sold at the end. By comparing the curves of different algorithms, it is evident that, in both the training and testing phases, the relative returns of EMDQN, as indicated by the corresponding curves, exceed those of other algorithms. This marked difference clearly illustrates the effectiveness of EMDQN in various stock prediction scenarios, highlighting its superiority in generating investment returns.

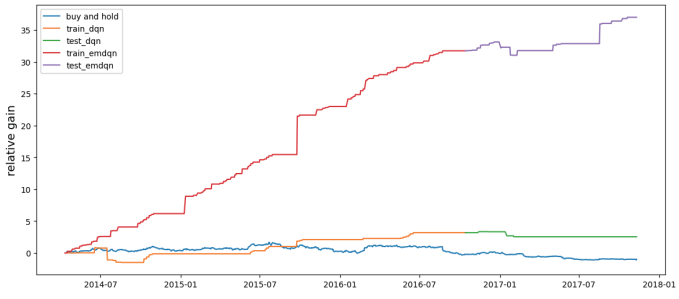


FIGURE 6. Comparative Results of DQN and EMDQN.

In terms of algorithmic structure optimization and strategy evaluation, EMDQN demonstrates significant advantages, which are fully verified by the experimental results in Figures 6, 7, and 8. At the level of algorithmic structure, EMDQN innovatively integrates specific components into the strategy network for in-depth optimization. This design endows it with excellent information extraction and integration capabilities when dealing with complex stock data. From the comparison of the curves in the experimental charts, it can be seen that, whether facing the changes of various data features in the stock market or complex data environments, the relative return curves of EMDQN during both the training and testing phases are always significantly higher than those of DQN, D3QN, and PPO. In contrast,

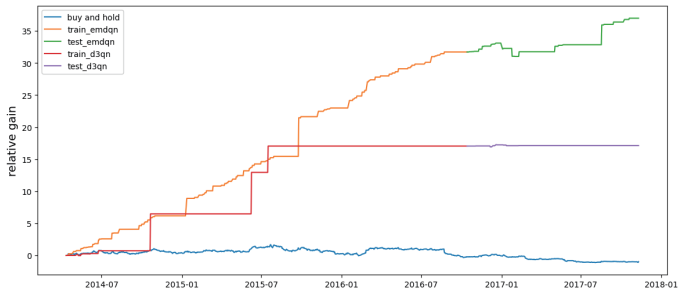


FIGURE 7. Comparative Results of D3QN and EMDQN.

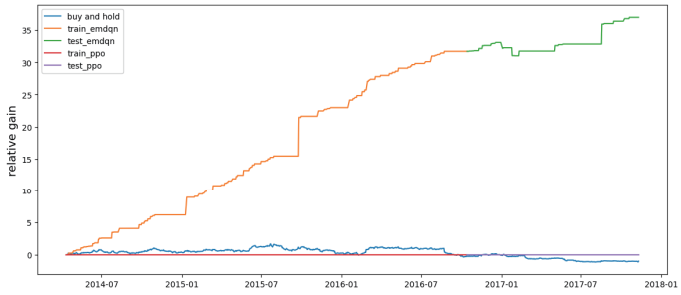


FIGURE 8. Comparative Results of PPO and EMDQN.

DQN adopts a more traditional network architecture, and both D3QN and PPO lack the integrated optimization approach of EMDQN. This leads to problems such as information processing delays or inaccuracies when they handle complex data. As reflected in the charts, the relative return curves of these three algorithms are obviously lower than that of EMDQN, and they fluctuate more violently, showing poor adaptability and scalability. In terms of the strategy evaluation mechanism, EMDQN uses the accumulated rewards obtained through the interaction between the agent and the environment as the fitness value of the strategy. Through efficient interaction with the environment and gradient updates during iterative cycles, it can adjust strategies more accurately when facing fluctuations in the stock market. The effectiveness of this mechanism is strongly supported by the experimental data: compared with DQN, D3QN, and PPO, EMDQN achieves higher relative returns during both the training and testing phases, and its curve continues to rise and remains stable, intuitively reflecting the effectiveness of strategy adjustment. On the contrary, DQN, D3QN, and PPO have relatively inefficient strategy evaluation and update mechanisms, making it difficult for them to quickly adapt to market changes, which results in poor performance in obtaining rewards. In the charts, their relative return curves not only have low values but also fluctuate irregularly during market fluctuations and fail to form a stable upward trend. Through a comprehensive analysis of multiple sets of experimental data, the advantages of EMDQN in key indicators are further highlighted. In terms of reward acquisition, EMDQN obtains higher rewards during both the training and testing phases, and its relative return curve far exceeds those of DQN, D3QN, and PPO, fully demonstrating its superior

prediction accuracy. In terms of runtime efficiency, EMDQN takes less time, which allows for more strategy iterations within the same time period and accelerates the optimization process. This also explains why its relative return curve can rise more quickly and remain smooth. In terms of loss control, EMDQN can stably control the prediction error loss at a lower level, effectively reducing the risk of model bias and ensuring the reliability of predictions, which is highly consistent with the stable performance of its relative return curve in the charts. Overall, the experimental results clearly show that EMDQN demonstrates remarkable superiority and reliability in the field of stock prediction and investment strategy optimization.

5. Conclusion

In stock trading prediction tasks, the EMDQN method proposed in this paper demonstrates superior performance over existing algorithms in most scenarios, owing to its distinct advantages. However, it also has limitations in terms of stability and resource requirements. Future research can explore solutions to address these shortcomings, such as further optimizing data processing strategies to enhance stability or developing efficient computational resource management methods to reduce resource consumption. This would thereby advance the continuous development of stock prediction algorithms and provide more accurate, reliable, and efficient technical support for financial market decisions.

Acknowledgements

This work is supported by the Key Scientific Research Projects of Higher Education Institutions in Henan Province under Grant No.24B520006. This work is also supported by the Research Foundation for Advanced Talents 2022BS073 of Henan University of Technology, the National Natural Science Foundation of China (61473114), the Natural Science Foundation of Henan (252300421806), the China Postdoctoral Science Foundation (2025M771592), the Scientific and Technological Innovation Teams of Universities in Henan Province (25IRTSTHN018) and Zhongyuan Science and Technology Innovation Leadership Talent Programme (254000510043), and Henan Provincial Key R&D Special Project (241111110200), the Science and Technology Project of Science and Technology Department of Henan Province, China (242102210016, 242102220121, and 212102210149). This work is also supported by the College Students ' Innovative Entrepreneurial Training Plan Program PX-38256057.

References

[1] F. Chih, *Stock price forecasting by hybrid machine learning techniques*, Proceedings of the International Multiconference of Engineers and Computer Scientists **1** (2009): 755.

[2] X. Chen, Q. Wang, C. Hu and C. Wang, *A stock market decision-making framework based on CMR-DQN*, Appl. Sci. **14** (2024): 6881.

[3] E. Chong, C. Han and F.C. Park, *Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies*, Expert Systems Appl. **83** (2017), 187–205.

[4] Y. Deng, Y. Kong, F. Bao and Q. Dai, *Sparse coding-inspired optimal trading system for HFT industry*, IEEE Trans. Ind. Info. **11** (2015), 467–475.

[5] R. Hafner and M. Riedmiller, *Reinforcement learning in feedback control*, Mach. Learn. **84** (2011), 137–169.
[6] K. A. Hassan, *A performance comparison of machine learning models for stock market prediction with novel investment strategy*, PLOS One **18** (2023): e0286362.
[7] Y. Huang, C. Zhou, K. Cui and X. Lu, *A multi-agent reinforcement learning framework for optimizing financial trading strategies based on TimesNet*, Expert Syst. Appl. **237** (2024): 121502.
[8] V. Konda and J. Tsitsiklis, *Actor-critic algorithms*, Adv. Neural Info. Process. Syst. **12** (1999), 1–12.
[9] Y. Kwon and Z. Lee, *A hybrid decision support system for adaptive trading strategies: Combining a rule-based expert system with a deep reinforcement learning strategy*, Decision Support Syst. **177** (2024): 114100.
[10] G. Lample and D.S. Chaplot, *Playing FPS games with deep reinforcement learning*, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2017.
[11] P. Liu, Y. Zhang, F. Bao, X. Yao and C. Zhang, *Multi-type data fusion framework based on deep reinforcement learning for algorithmic trading*, Appl. Intelligence **53** (2023), 1683–1706.
[12] J. W. Lee, *Stock price prediction using reinforcement learning*, in: SIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No.01TH8570), Pusan, South Korea, 2001, pp. 690–695.
[13] J.W. Lee, E. Hong and J. Park, *A Q-learning based approach to design of intelligent stock trading agents*, in: 2004 IEEE International Engineering Management Conference (IEEE Cat. No.04CH37574), Singapore, 2004, pp. 1289–1292.
[14] Y. Li, P. Ni and V. Chang, *Application of deep reinforcement learning in stock trading strategies and stock forecasting*, Computing **102** (2020), 1305–1322.
[15] Z. Lin, T. Zhao and G. Yang, *Episodic memory deep Q-networks*, in: Proceedings of th.e 27th International Joint Conference on Artificial Intelligence, 2018, pp. 2433–2439
[16] S. Mohammed and K. Amr, *On the reduction of variance and overestimation of deep Q-learning.*, arXiv preprint arXiv:1910.05983 2019.
[17] R. Sathya, P. Kulkarni, M.N. Khalil and S.C. Nigam, *Stock price prediction using reinforcement learning and feature extraction*, Int. J. Recent Technol. Eng. **8** (2020), 3324-3327.
[18] D. Sha, S. Hailong, *A stock prediction method based on deep reinforcement learning and sentiment analysis.*, Appl. Sci. **14** (2024): 8747.
[19] A. Shahrokh, *Hybridization of evolutionary Levenberg–Marquardt neural networks and data pre-processing for stock market prediction*, Knowledge-Based Syst. **35** (2012), 245–258.
[20] W. Sunye, *Machine learning and deep learning predictive models for the stock market*, in: SHS Web of Conferences, 2024.
[21] F. Weiss, *A numerical approach to solve consumption-portfolio problems with predictability in income, stock prices, and house prices*, Math. Meth. Oper. Res. **93** (2021), 33–81.
[22] D. J. White, *A survey of applications of Markov decision processes*, J. Oper. Res. Soc. **44** (1993), 1073–1096.
[23] B. Yang and T. Liang, *Deep reinforcement learning based on transformer and U-Net framework for stock trading*, Knowledge-Based Syst. **262** (2023): 110211.
[24] Y. Yang, L. Juntao, and P. Lingling, *Multi‐robot path planning based on a deep reinforcement learning DQN algorithm*, CAAI Transactions on Intelligence Technology **5** (2020), 177–183.
[25] B. Zhu, W. Zhu, W. Li, *An improved method of q-learning algorithm*, J. Physics. **2035** (2021): 012014.

Manuscript received May 29 2025
revised August 11 2025

Z. Wu
School of Artificial Intelligence and Big Data, Henan University of Technology, Zhengzhou 450001, China
E-mail address: 18303806390@163.com

D. Xin
School of Artificial Intelligence and Big Data, Henan University of Technology, Zhengzhou 450001, China
E-mail address: 3421948662@qq.com

X. Xiao
College of Management, Henan University of Technology, 450001, Zhengzhou, China
E-mail address: xxc.zxc@gmail.com

X. Zhu
(Corresponding Author) School of Artificial Intelligence and Big Data, Henan University of Technology, Zhengzhou 450001, China
E-mail address: xczhuiffs@163.com