Generalization and Adaptation in Option Hedging Strategies Using Deep Learning[*]

Seiya Kuno[†]
Faculty of Commerce, Doshisha University

Abstract

Deep Reinforcement Learning (DRL) has been actively studied and applied in practice in recent years as an effective approach for constructing option hedging strategies in incomplete and frictional financial markets. DRL provides a data-driven framework that can learn optimal hedging strategies directly from simulations or past market trajectories, unlike traditional model-based methods that rely on specific processes related to market dynamics. On the other hand, because the strategies are derived through deep learning using complex neural networks, they are treated as black boxes, and various issues remain regarding the explainability and interpretation of the model, as well as generalization performance and range of application. In this study, we investigate the robustness and generalizability of DRL-based option hedging strategies under various market conditions and provides insight into the explainability of the models required in practice.

1 Introduction

In recent years, reinforcement learning, a branch of machine learning, has been actively applied in the financial field, primarily in areas such as portfolio optimization, options trading, and algorithmic trading. Reinforcement learning aims to develop a strategy that maximizes cumulative rewards by an agent, which determines its actions over time to achieve a goal, to learn while interacting with the environment. Furthermore, model-free (reinforcement learning) approaches, particularly those using neural networks, have been explored and implemented to derive more appropriate (practical) option hedging strategies. Deep reinforcement learning (especially DQN, which we use here) faces several issues: stability, learning efficiency and convergence speed, and behavioral appropriateness. We investigate whether frameworks that improve these capabilities (especially Double DQN, Dueling Network, and Prioritized Experience Replay) are more generalizable and adaptability to a wider range of applications. These are thought to primarily represent improvements in learning techniques, learning structures, and sampling methods. While these approaches improve generalization, they also require a balance with adaptation, particularly for Dueling Networks, due to their limited adaptability. In this paper, we examine the usefulness of these DQNs in deriving hedging strategies for options, particularly European call options.

[*]This research was supported in part by the Research Institute for Mathematical Sciences, a Joint Usage/Research Center, Kyoto University in 2025 and the Grant-in-Aid for Scientific Research (No. 23K01460) of the Japan Society for the Promotion of Science.

[†]Address: Karasuma-higashi-iru, Imadegawa-dori, Kamigyo-ku, Kyoto, 602-8580 Japan
E-mail address: skuno@mail.doshisha.ac.jp

The remainder of this paper is organized as follows. We provide a general discussion on deriving option hedging strategies using reinforcement learning in Section 2, and Section 3 introduces existing research on applications to deep learning. Section 4 discusses improvements to option hedging problems using DQNs. Finally, we conclude this paper.

2 Option hedging using reinforcement learning

Throughout this paper, an agent is defined as an entity that makes decisions about actions over time to achieve a goal. Reinforcement learning involves an agent learning as it interacts with the environment, constructing a policy that maximizes the value of its cumulative reward. While supervised learning provides the agent with the correct action or answer based on data, reinforcement learning does not instruct the agent on what action to take; instead, the agent finds the action that maximizes the reward through trial and error. Hereinafter, the current state of the environment will be denoted as $s(S)$, the action selected by the agent as $a(A)$, and the feedback received from the environment based on the agent's actions as $r(R)$. Reinforcement learning can be divided into model-based approaches, in which the agent learns a model of the environment's behavior (e.g., state transition function and reward function) to predict future states and rewards, and model-free approaches, in which the agent does not learn a model of the environment's behavior but instead learns optimal actions through actual interaction with the environment. Furthermore, there are value-based algorithms, which predict how much reward each state will bring in the future and select actions with the highest value. There are also policy-based algorithms, in which the agent learns a specific behavioral strategy (policy) and optimizes that policy to maximize the cumulative reward obtained from the environment. There is also a hybrid approach called actor-critic. In this paper, we consider value-based algorithms, in which the agent learns the value of each state and state-action pair.

2.1 Value-Based algorithm

The cumulative reward from time t through T is expressed as

$$G_t = \sum_{k=0}^{T-t} \gamma^k R_{t+k+1}, \tag{2.1}$$

where $\gamma \in (0, 1)$ is the discount rate for future rewards.

The state value $V(s)$ at time t is the expected cumulative reward obtained by the agent when the initial state is s, and is expressed as follows:

$$V(s) = \mathbb{E}[G_t | S_0 = s]. \tag{2.2}$$

Action value $Q(s, a)$ indicates the expected cumulative reward obtained when action a is selected in state s, and is expressed as follows:

$$Q(s, a) = \mathbb{E}[G_t | S_0 = s, A_0 = a]. \tag{2.3}$$

Both the state value function and the action value function assume the Markov property, so only the initial state and initial action are conditions(The evaluation begins at time $t = 0$). In what follows, we define the Q-learning agent and SARSA agent, with α as the learning rate.

The Q-learning agent uses the maximum Q-value for the next state and updates it using,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right). \tag{2.4}$$

In other words, it takes an action that transitions to the state with the highest value.

On the other hand, agents following the SARSA method (SARSA agents) use the Q-value obtained from the actual action chosen in the next state and update their Q-values using,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right). \tag{2.5}$$

In this case, actions are determined according to the strategy.

The learning rate α represents the influence of new information in updating the Q-value. A high value significantly reflects new information in the evaluation, but it also directly affects the update, resulting in poor generalization. Furthermore,

$$\Delta_t := R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \tag{2.6}$$

is called the temporal difference error (TD error) because it is the difference between the predicted value of the state at different time steps. In addition, since Q-learning and SARSA are learning methods that use TD errors, they are both called TD learning, and the process of learning the optimal policy and updating the Q function is called TD control. Therefore, Q-learning is an off-policy TD control because it updates by optimizing regardless of the policy, while SARSA is an on-policy TD control.

Next, the ϵ-greedy method is a method of searching for the optimal action by probabilistically selecting the probability that the agent will select it, and then selecting the action considered optimal based on the remaining probability. This method is used to prevent the agent from falling into a local optimum, thereby improving learning stability by balancing "exploration" and "exploitation." The ϵ-greedy policy π is expressed as follows:

$$\pi_\epsilon = \begin{cases} \tilde{a} & (u < \epsilon) \\ \underset{a}{\operatorname{argmax}} Q(S, a) & (u \geq \epsilon) \end{cases} \tag{2.7}$$

where u is a random variable ($u \sim U(0, 1)$) uniformly distributed between 0 and 1, and $\epsilon \in (0, 1)$ is a real number (hyperparameter). It is often scheduled to decay from an initial value of 1 at a specific decay rate as it learns.

2.2 Option Hedging Strategy

Based on [3], we present the derivation of the most basic Q-learning options hedging strategy (a strategy that hedges risk by using options trading to take positions to prevent losses due to price fluctuations). This strategy can be used to respond to large declines in asset prices and

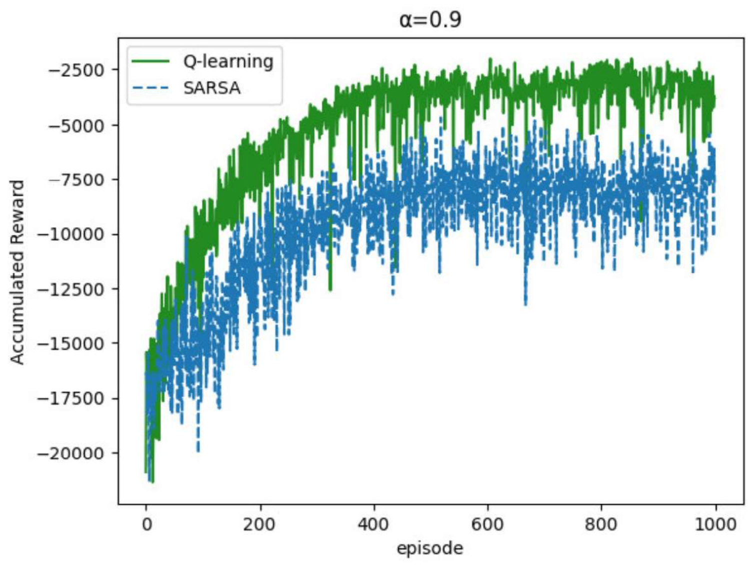

Figure 1: Stability and Convergence of Q-learning and SARSA Learning at a Learning Rate of $\alpha = 0.9$

sudden changes in volatility. We assume the dynamics of the underlying asset price P to follow the geometric Brownian motion,

$$dP_t = \mu P_t dt + \sigma P_t dW_t \tag{2.8}$$

where the constants μ and σ represent drift and volatility, and W represents standard Brownian motion. In the BSM model, under the assumption that transactions are continuous and cost-free, the price of a European call option with maturity T at time t is explicitly obtained, and delta is given as a delta hedging strategy,$\frac{\partial C}{\partial P} = N(d_1)$, where N is the standard normal distribution function and $d_1 = \frac{\ln(\frac{P_t}{K}) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}$, moreover K and r represent the call option strike price and risk-free interest rate, respectively. As an example, Figure 1 shows the difference in stability and convergence of learning between Q-learning and SARSA. For simplicity, the initial value of the underlying asset price following naturalized Brownian motion is set to 100, and the system is trained 500 times to hedge once per day for 250 days. With a large learning rate like the one shown in the figure, learning progresses quickly but becomes unstable.

3 Options hedging strategy with neural network

In this section, we present a European call option hedging strategy based on the Deep Q Network(DQN) by [2] and Deep Hedging by [1].

3.1 DQN

A technique called Experience Replay is often used, in which samples are stored in a buffer and multiple samples are extracted from there to perform mini-batch learning (see PER), because the samples obtained by the agent have a strong correlation over time. For other various ideas, see [8]. As a formula, let $L(\theta)$ be the loss function, and find a that satisfies the following:

$$L(\theta) = \mathbb{E}\left[\frac{1}{2}\left(R_{t+1} + \gamma \max_a Q_{\theta'}(S_{t+1}, a) - Q_\theta(S_t, A_t)\right)^2\right]. \tag{3.1}$$

3.2 Deep hedging

Deep hedging measures the cost associated with hedging using a scale based on convex risk and approximates a function that minimizes this using a neural network. Although it is not strictly speaking a reinforcement learning method, since the research by [1] in 2019, it has been widely referenced in both academic and practical fields as a problem for deriving hedging strategies using neural networks. Generally, learning proceeds by setting the underlying asset price process and simulating. As for the price model followed by the underlying asset, [1] uses a stochastic volatility model (Heston model), [7] uses a rough volatility model, and [6] propose a deep hedging model that does not use the underlying asset price process. For a single underlying asset, transactions are possible at discrete time points $0 < \cdots < t < \cdots < T$, with the asset price denoted as P.

Let $\delta = \{\delta_t\}_{0 \le t < T}$ denote the agent's holdings of the asset at each time step, with the current time set as 0. The value of a European call option with strike price K at maturity is given by the max function: $Z = max\{P_T - K, 0\}$. Furthermore, we consider using only the underlying asset as a hedging instrument. Under this setting, the profit or loss PL_T from hedging at maturity when taking a short position in this option is:

$$PL_T(Z_T), P, \delta) = -Z_T + p_0 + \sum_{t=0}^{T-1} \delta_t (P_{t+1} - P_t) - \sum_{t=0}^{T-1} c_t(\delta_t - \delta_{t-1}) \tag{3.2}$$

where p_0 is the selling price of this call option. The third term on the right-hand side of this equation (3.2) represents the cumulative hedging profit/loss, while the fourth term represents transaction costs in response to the trading volume at each time point, moreover we assume $\delta_T = \delta_{-1} = 0$.

Next, the loss function $l(\delta)$ is given by

$$l(\delta) = \rho(PL_T(Z_T, P, \delta)), \tag{3.3}$$

where ρ is a convex risk measure that has the following characteristics: 1) monotonically decreasing, 2) convexity, and 3) cash invariance. Deep Hedging is a problem of finding a hedging strategy that is the solution to the convex risk minimization problem $\delta^* = \underset{\delta}{\operatorname{argmin}} \rho((PL_T(Z_T, P, \delta))$.

While various convex risk measures can be considered, here we use Conditional VaR (CVaR) as the loss function. This CVaR represents the expected value of large losses occurring with a probability of $100 \times \beta\% (\beta \in (0,1))$. Let X be the random variable representing profit/loss, and let $VaR_\beta(X)$ as the lower $100 \times \beta\%$ percentile of its distribution,

$$\rho(X) = \mathbb{E}[-X \mid -X \ge VaR_\beta(X)] \tag{3.4}$$

is defined.

In order to align the profit and loss indicators with DQN, a simulation was performed for Deep Hedging using the profit and loss amount directly rather than CVaR. Deep Hedging tended to be slightly more accurate, but no major differences were observed. On the other hand, DQN requires a considerable amount of time for calculations as the number of epochs increases and the calculations become larger in scale, so improvements in computational efficiency are desirable.

4 Improving option hedging strategies using DQN

From here, we will consider improving the performance of the option hedging strategy by improving DQN, as well as generalization and adaptation. In [5], an application of Q-learning using neural networks is introduced, which simultaneously incorporates seven improved DQN, Double DQN, Dueling Network, Prioritized Experience Replay(PER), Noisy Network, Multi-step learning, and Categorical(Distributional) DQN, and then performance improvements are reported. Here, in order to clarify the difference from the perspective of option generaliz ation and adaptation, we will consider the hedging error when considering the original DQN, especially "Double DQN", "Dueling Network", and "PER". When DQN seeks training accuracy, the calculation time required increases as the number of epochs increases and the calculation scale becomes larger, so it is desirable to improve the computational efficiency. In other words, it is necessary to balance training accuracy and training efficiency, or to improve one without sacrificing the performance of the other.

4.1 Improved method for DQN

- Double DQN

Double DQN is a method to speed up convergence by eliminating the bias (overestimation) that causes the value function estimate to be overestimated, and was proposed by [4] in 2016. This bias (rather than increasing all estimates uniformly) is an overestimation that varies depending on the action, which changes the action selected by the policy and leads to a decrease in performance. The calculation of the maximum Q value is separated into ① the selection of the action a that gives the maximum Q, and ② the calculation of Q for action a, as shown in the following equation

$$\max_a Q(s,a) = Q\left(s, \operatorname*{argmax}_a Q(s,a)\right). \tag{4.1}$$

The TD error approximated by the neural network is defined as

$$R_{t+1} + \gamma \max_a Q_{\theta'}\left(S_{t+1}, \operatorname*{argmax}_{a'} Q_\theta(S_{t+1}, a')\right) - Q_\theta(S_t, A_t). \tag{4.2}$$

Since Double DQN reduces bias of Q and overestimation, making it more stable and improving generalization than DQN. However, because it has the same structure as DQN, it is unlikely that improvements in adaptability can be expected.

- Dueling Network

Table 1: Hedging error

	RMSE	MAE	Avg.Cost
QLBS	0.93	0.71	0.63
DQN	0.81	0.61	1.11
Dueling +Double	0.62	0.60	0.56

The Dueling Network was proposed by [10] in 2016 and is a method that uses an innovative network structure to improve learning efficiency and accuracy by separating state value and action advantage. For tasks where there are many states where the results are similar regardless of the action, it is more efficient to learn by dividing the learning into ① the goodness of the state: $V(s)$ and ② the relative increase or decrease of each action: $A(s, a)$ (Advantage), rather than directly calculating the Q for each action. Specifically, it is expressed as

$$Q(s,a) = V(s) + A(s,a) - \frac{\sum_{a'} A(s,a')}{|A|} \tag{4.3}$$

where $|A|$ is the number of selectable actions. As mentioned in [10], it is generally said that there are restrictions on the range of adaptation of Dueling Networks. By separating the value structure and action difference, the Dueling Network can learn the market background and trading details separately, and while its robustness to noisy samples improves generalization, V does not follow sudden changes. In option hedging strategies, the market can change overnight due to volatility jumps or gamma surges, so the inertia of V may have a negative effect, which may cause problems with adaptability. Table 1 shows the results of calculating the hedging error for QLBS, DQN, and Dueling+Double, with an initial price of 100, an exercise price of 100, one year (one transaction per day, 250 transactions), two hidden layers, an activation function of relu (similar to Deep Hedging), and the number of episodes (the period from the start to the end of the task to be solved) of 1500, using RMSE, MAE, and Average Cost as evaluation criteria. From this, we will see that by combining the Dueling Network and Double DQN, we can see improved performance for option hedging strategies.

- Prioritized Experience Replay(PER)

In DQN, mini-batches are created by random selection using the mini-batch method, which makes it inefficient at learning important (valuable) events. For this reason, to improve learning efficiency, it is desirable to prioritize sampling highly unexpected transitions from the buffer, and then Prioritized Experience Replay was proposed by [9]. Specifically, the larger the TD error Δ(Equation(2.6)) of transition information, the more unexpected it is, and the more weight is given to the probability of sampling $P(i)$. Here,

$$P(i) = \frac{(|\Delta_i| + \epsilon)^c}{\sum_k^N (|\Delta_k| + \epsilon)^c} \tag{4.4}$$

where c is a hyperparameter ($0 \leq c \leq 1$), and $c = 0$ indicates random sampling, furthermore, ϵ represents an appropriate small value such that the probability is not strictly zero. Because

PER emphasizes important past trends, it has the advantage of being able to easily capture common market structures and is therefore thought to have good generalization ability.

In addition, since TD error jumps in suddenly changing markets, it is prioritized for learning, and learning from risky past memories comes into play, resulting in very fast tracking and excellent adaptability.

4.2 Numerical examples

In this subsection, we compare the performance of each option hedging problem using machine learning in a simulated environment. All activation functions in the intermediate layers use relu, and the output layer uses a linear function. The rest of the setup is the same as in deep hedging, with 30 nodes in each hidden layer. Figure 2 compares the performance (hedging error) of DQN, Double DQN, Dueling Network, PER, and an improved version that simultaneously considers Double+Dueling+PER when there are two hidden layers and 500 episodes, while Figure 3

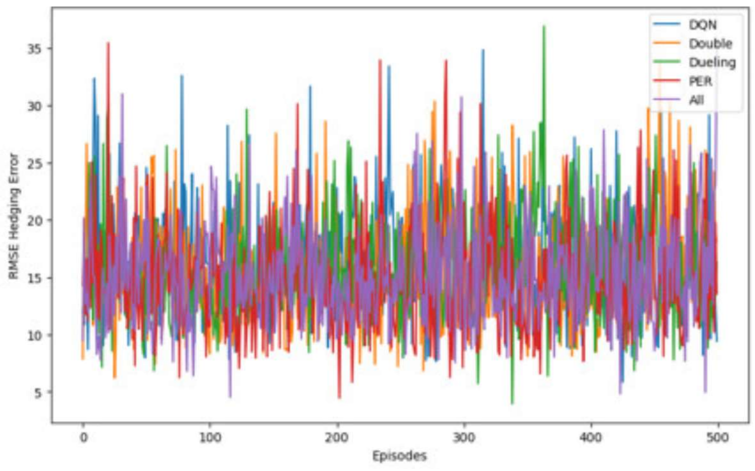

Figure 2: Hedging error for 2 hidden layers and 500 episodes

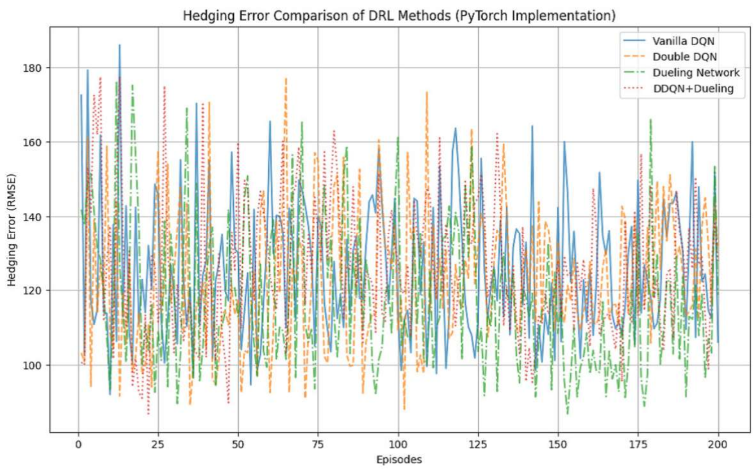

Figure 3: Hedging error for 5 hidden layers and 200 episodes

compares the performance when there are five hidden layers and 200 episodes. The error in both cases is measured using RMSE. Since there is no significant difference between DQN and Deep Hedging in this setting, we will use Deep Hedging in DQN. In Figure 2, with shallow hidden layers, no difference in learning efficiency is observed even when the number of episodes is increased. However, as shown in Figure 3, when there are deep hidden layers, DDQN works to suppress overestimation, but the Dueling Network tends to accelerate learning and demonstrate better performance due to improved learning efficiency of state value. However, due to the setting in [3] and the small reward used in this simulation, the difference is only slight. Also, in Figure 3, PER is used for all methods. Deep networks enable more complex state representations, so the advantage of the Dueling Network approach, which learns by separating state value and action advantage, becomes more apparent. When there are few hidden layers, including all of them does not improve performance because it does not improve overestimation. On the other hand, as the layers become deeper, it can be seen that the Dueling Network improves generalization and the Double DQN improves (complements) adaptation.

5 Conclusion

In this study, we investigated the generalization performance and adaptation of options hedging problems using deep reinforcement learning with neural networks by considering various algorithms and schemes. Regarding the option hedging problem, it is thought that the adaptation can be resolved by using a dueling network. Also, as the number of hidden layers and the number of training increases, performance generally improves in the order of a PER-based combination of Double and Dueling, Dueling Network, and Double DQN. However, in this setting, only slight differences were observed. Furthermore by fundamentally reviewing the structure and methods, even in the case of the Dueling Network, which has problems with adaptability, by combining it with Double DQN, it is possible to achieve learning that shows improvements in both generalization and adaptation. But if the layers are made quite deep, it takes a considerable amount of time and is not realistic for practical application. In such cases, improving the sampling method such as Experience Replay will contribute to improving performance rather than improving the structure or methods. Furthermore, slight changes in conditions result in unstable performance with respect to hedging errors, so further consideration of evaluation criteria is required.

References

[1] Buehler, H., Gonon, L., Teichmann, J., and Wood, B. (2019), "Deep hedging," *Quantitative Finance*, **19**, 8, 1271–1291.

[2] Du, J., Jin, M., Kolm, P. N., Ritter, G., Wang, Y., and Zhang, B. (2020), "Deep reinforcement learning for option replication and hedging," *The Journal of Financial Data Science*, **2**, 4, 44–57.

[3] Halperin, I. (2017), "Qlbs: Q-learner in the black-scholes (-merton) worlds," *arXiv preprint arXiv:1712.04609.*

[4] Van Hasselt, H., Guez, A., and Silver, D. (2016), "Deep reinforcement learning with double q-learning," *Proceedings of the AAAI conference on artificial intelligence*, **30**, 1.

[5] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018), "Rainbow: Combining improvements in deep reinforcement learning," *Proceedings of the AAAI conference on artificial intelligence*, **32**, 1.

[6] Hirano, M., Minami, K., and Imajo, K. (2023), "Adversarial deep hedging: Learning to hedge without price process modeling," *Proceedings of the Fourth ACM International Conference on AI in Finance*, 19–26.

[7] Horvath, B., Teichmann, J., and Žurič, Ž. (2019), "Deep hedging under rough volatility," *Risks*, **9**, 7, 138.

[8] Plaat, A. (2022), *Deep reinforcement learning*, Singapore: Springer.

[9] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015), "Prioritized experience replay," *arXiv preprint arXiv:1511.05952.*

[10] Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Frcitas, N.(2016), "Architectures for Deep reinforcement learning," *International conference on machine learning*, PMLR, 1995–2003.