Deep hedging with reinforcement learning

School year 2022/2023

Table of contents

Table of figures

Deep hedging using reinforcement learning

Introduction

Quantitative finance studies extensively the hedging of derivative products, one of the first efforts in this area being the Black-Scholes model by Black and Scholes (1973). This model assumes a market devoid of arbitrage opportunities and continuous hedging without accounting for transaction costs. However, in the real world, these idealistic assumptions do not hold, prompting traders to rely on their experience to craft an optimal hedging strategy. This strategy often involves adjusting positions at intervals that strike a balance between minimizing hedging errors and managing trading costs. Of particular importance among these strategies is Delta hedging, an ongoing process aimed at mitigating unwanted risk stemming from adverse price movements in the underlying asset.

The Delta-hedging strategy, being a sequential decision-making process, presents an exciting opportunity for the application of Reinforcement Learning techniques. In this context, the agent learns to maximize the expected profit and loss whilst at the same time minimize its variance at maturity. The agent operates within an environment that is continuously changing and decides on the holding amount in the underlying asset at each time period to make the position Delta-neutral. The reward in this paradigm is the periodic profit and loss made due to each of the agent's holding decisions.

In this project, our situation is that of a trader hedging a short position in a call option. She can rebalance her portfolio at specific time intervals and is subject to trading costs. We will first start with a brief literature review on reinforcement learning and the use of it for hedging, we will then move on to detail the methodology and the approach, then we present the results and a conclusion.

Literature review

Reinforcement learning (RL)

Reinforcement learning is closely linked to machine learning, primarily because the agent's learning process involves striving to achieve a goal through the assimilation of knowledge from prior actions and feedback provided by the environment in the form of rewards. These rewards serve as a means of conveying to the agent the quality of its actions at specific moments. Ultimately, the agent's ultimate objective is to discover an optimal policy.

Using information from the current environment state and past reward received, the agent takes an action, which induces the transition of the market to a new state, resulting in a new reward. The agent's overarching objective is to maximize the cumulative sum of discounted rewards over the long term.

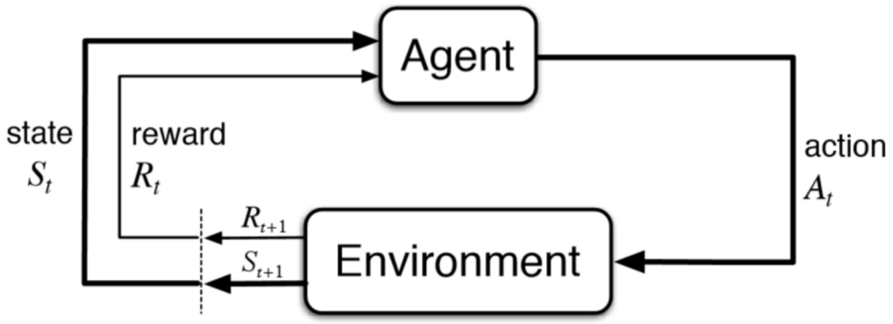

Figure 1. General representation of RL (Sutton and Barto (1998))

For a full formalism of an RL learning process as defined by Sutton and Barto (1998), we refer to appendix A.

The RL training process can be implemented using multiple algorithms. Some of them are tabular, i.e. they work with finite and discretized action and state spaces like Q-learning and SARSA. These algorithms are not applicable in our case since our state space, which contains the underlying spot prices, is continuous. Furthermore, we can also choose to not discretize the action space. In this case, the need arises to use some function approximation method for the action value and policy functions. Deep neural networks with non-linear functions in the hidden layers are a universal function approximator and may therefore be used. The drawback, however, is that global convergence in this case is not always guaranteed. One way to deal with this is by introducing exploration noise, whether to the action space or to the actor's parameter space.

Deep reinforcement learning for delta hedging derivatives

The literature on the topic is diverse in terms of the choices made with the environment, the reinforcement learning algorithm, the reward function and the underlying's diffusion model. Particular attention will be given to the choice of the reward function, as its role is crucial in the learning process of the RL agent.

Kolm and Ritten (2019) hedge a 100 short positions in European call options and define the state at time step t as the holding of the underlying at that time step, the underlying spot price and the time to maturity. They hedge in a Black-Scholes market in discrete time with quadratic transaction costs. They use the SARSA algorithm, meaning that their action space is discretized. Du et al.

(2020) uses this same framework with a deep Q-learning (DQN) algorithm and several of its variations and a Proximal Policy Optimization (PPO) algorithm, keeping the action space discrete. Giurca and Borovkova (2021) also use this framework and extend it to other diffusion models and the additional RL algorithm Deep Deterministic Policy Gradient (DDPG). Cao et al. (2020) approach the problem differently but use a similar environment with proportional transaction costs and the DDPG algorithm to train the agent. Furthermore, they extend the framework to a Heston stochastic volatility model, and discuss the possibility of using different models to diffuse and price the options by using the Heston model to diffuse the underlying and the Black-Scholes model to price the option, concluding that it bears no effect on the agent's ability to learn from the environment.

When it comes to the reward function, Kolm and Ritten (2019) assume that the agent is risk averse with a given (concave and increasing) utility function, and the optimal hedging strategy is the solution to the mean-variance problem where the agent maximizes the expected wealth and minimizes its variance. The wealth increments from $t-1$ to t are none other than the profit and loss over this same period, $PL_t := \delta W_t$. Given this, the reward function defined as (with λ the risk aversion parameter):

$$r_t(s_t, a_t) = PL_t(s_t, a_t) - \frac{\lambda}{2} PL_t(s_t, a_t)^2$$

allows the agent to find the policy that solves the mean-variance problem:

$$\arg\max_{\pi} \sum_{t=0}^{T} \mathbb{E}_{\pi}[PL_t] - \frac{\lambda}{2} Var_{\pi}[PL_t]$$

The profit loss at time t is formulated as $PL_t(s_t, a_t) = \delta V_t + N_t(\delta S_t) - c(\delta N_t)$ where N is the holding in the underlying, V is the option value, S the underlying's price process and c a cost function.

Cao et al. (2020) on the other hand formulate the problem as a cost minimization problem, with the objective function given by $Y(t) = \mathbb{E}[C_t] + c\sqrt{\mathbb{E}[{C_t}^2] + \mathbb{E}[C_t]^2}$ where c is the risk aversion term and C_t the total hedging cost from time t to maturity. The aim is then to minimize $Y(0)$. The reward in their case is simply $r_t(s_t, a_t) = PL_t(s_t, a_t)$. However, to accommodate the form of the objective function, they use two Q-functions, where one approximates $\mathbb{E}[C_t]$, and the other approximates $\mathbb{E}[{C_t}^2]$. This results in a DDPG agent with two critics and one actor.

Methodology

In this project, we recreated the two approaches of Cao et al. (2020) and Giurca and Borovkova (2021). The code implementation is on my GitHub repository. The environments and DDPG architecture are different for each implementation, and in this section, we will only present the theoretical framework of hedging a European call option using a DDPG agent as presented in Giurca and Borovkova (2021) for the sake of brevity.

The hedging problem revisited

We are short a European call option with strike $K = 100$ and maturity T of 1 month. The underlying follows a GBM process such that $dS_t = rdt + \sigma dW_t$. For the numerical values of the parameters, we take the initial spot $S_0 = K$, the risk-free rate $r = 0.02$ and the volatility $\sigma = 0.2$. Indeed, any other diffusion model can be used, and the authors themselves use the Heston and Bates models in addition to the Black-Scholes model.

The rebalancing of the portfolio is done with a specified frequency Δt, for instance once a day. Indeed, the price process is discretized with a different time step with a chosen precision.

The profit and loss from t to $t+1$ is given by

$$PL_{t+1} = V_{t+1} - V_t + N_t(S_{t+1} - S_t) - \kappa\, |S_{t+1}(N_{t+1} - N_t)|$$

for $0 \leq t \leq T-1$ where N_t is the holding between the time t and $t+1$, V_t is the price of the option at time t given by the Black-Scholes formula, $\kappa = 0.01$ is the proportional trading cost. Note that with this formulation of the reward, a perfect hedge will result in a zero P&L increment in each period.

The environment

The state variable is given at each rebalancing time $t \in \{0, \dots, T\}$ as $s_t \coloneqq (S_t, N_t, V_t, \Delta_t)$ where Δ_t is the option's Delta value at the beginning of the period t. We compute it using the closed Black-Scholes formula.

The action a_t is given by the agent at each rebalancing time $t \in \{0, \dots, T\}$ and signifies the holding for the next period, i.e. N_t.

The reward given to the agent after each rebalancing time $t \in \{0, \dots, T\}$ is:

$$r_t(s_t, a_t) \coloneqq PL_t(s_t, a_t) - \frac{\lambda}{2}\, PL_t(s_t, a_t)^2$$

as stated above, with $\lambda = 0.1$.

Using these variables, the transitions that will then be stored and used to train the agent are defined as $t := (s_t, a_t, r_t, s_{t+1})$.

The DDPG agent

The Deep Deterministic Policy Gradient (DDPG) algorithm is a reinforcement learning technique introduced by Lillicrap et al. (2015) that addresses continuous action space problems. It is an extension of the original Deep Q-Network (DQN) algorithm, adapted to solve problems where the action space is continuous. It follows an actor-critic architecture. The actor network approximates the policy function π, and the critic network evaluates the quality of the actions chosen by the actor by estimating the Q-value (expected cumulative reward) associated with taking a specific action from a given state $q(s, a)$.

Essentially, we have two learning processes happening:

- The agent learns the best approximation of the optimal action-value function $q_*(s, a)$, also called the Q-learning process, by using off-policy data (data obtained by exploring actions different from the actions recommended by the current policy) and the Bellman equation.
- Given the optimal action-value function for any state s, then the optimal policy is learned by approximating the policy function π such that $q\big(s, \pi(s)\big) = max_a q_*(s, a)$.

The Bellman equation giving the optimal Q-value function is:

$$q_*(s, a) = \mathbb{E}_\pi[\ r(s, a) + \gamma\ max_{a'} q_*(s', a')\ \mid\ s, a\ ]$$

As stated, the optima Q-value for each state s is approximated using a neural network $q_\theta(s, a)$ with parameter vector θ, and the optimal policy is approximated using a neural network $\pi_\omega(s)$ with parameter vector ω. The off-policy data points are stored in a replay buffer in the form of transitions. The replay buffer $\mathcal{B}$ should contain both old and new transitions (called experiences), and the size of it matters to prevent both slow convergence and overfitting.

The Q-value is learned by the critic network by setting the loss as the mean-squared Bellman error:

$$Loss(\theta, \mathcal{B}) = \mathbb{E}\left[\left(q_\theta(s, a) - (\ r + \gamma q_\theta(s', \pi_\omega(s'))\ )\right)^2\right]$$

As for the actor, its aim is to learn a policy $\pi_\omega(s)$ that maximizes the expected return $q_\theta(s, \pi_\omega(s))$. Since the Q-value function is differentiable with respect to a, this is done by a simple gradient ascent with respect to policy parameters to solve:

$$\max_\omega \mathbb{E}[q_\theta(s, \pi_\omega(s)) \mid s\ in\ \mathcal{B}]$$

The term $r + \gamma q_\theta(s', \pi(s'))$ in the critic loss is called the *target*, when the critic network minimizes the loss, it gets closer and closer to this target. In practice, the DDPG network uses two separate actor and critic networks in addition to the actor and critic networks, called target networks, and which are used to compute the target term and consequently the loss. The reason for this as stated in Lillicrap et al. (2015) is to stabilize the learning process; the target depends on the parameters of the networks. The target networks' parameters are updated using soft updates choosing $\rho \to 1$ and setting:

$$\begin{cases} \theta_{target} \leftarrow \rho\theta_{target} + (1-\rho)\theta \\ \omega_{target} \leftarrow \rho\omega_{target} + (1-\rho)\omega \end{cases}$$

Exploration noise

The dilemma of exploration and exploitation is a fundamental challenge in reinforcement learning. It revolves around finding the right balance between two conflicting objectives:

- Exploration: This refers to the agent's need to try out new actions or states to discover potentially better strategies. It involves taking uncertain or untried actions that may lead to higher rewards in the long run but are risky in the short term. Without exploration, the agent might get stuck in suboptimal policies and miss out on more rewarding ones.
- Exploitation: Exploitation involves making decisions that the agent believes are the best based on its current knowledge and experience. It means choosing actions that are known to yield high rewards based on past interactions with the environment. Exploitation can lead to the agent consistently following a known policy, but it might miss out on potentially better strategies if it doesn't explore.

In the context of the DDPG algorithm, the target networks introduce a degree of exploitation by using older, more stable estimates during updates, which helps in learning more reliable policies. On the other hand, DDPG being an off-policy algorithm, uses some form of noise to pick actions that the agent can learn from.

The exploration strategy introduced in the original work of Lillicrap et al. (2015) is to either add temporally correlated noise to the actions given by the actor by using an Ornstein-Uhlenbeck process, or uncorrelated noise in the form of simple additive gaussian noise. Essentially, in both cases, the exploration is realized through adding noise to the *action space*.

This approach has its merits, but as remarked by Plappert et al. (2018), since the noise process is independent from the state space, for a given state s_t, even if that state were to be exactly sampled again, we would obtain a different action. This introduces a rift in the agent's ability to learn the relationship between a state and the action picked for it. To remedy this, they introduce *parameter space noise* for exploration. As the name suggests, the noise it introduced directly to the actor's parameters. Indeed, a separate noisy actor network is introduced, whose parameters are sampled anew at the beginning of each episode and reset to be equal to the actor's parameters after each parameter update.

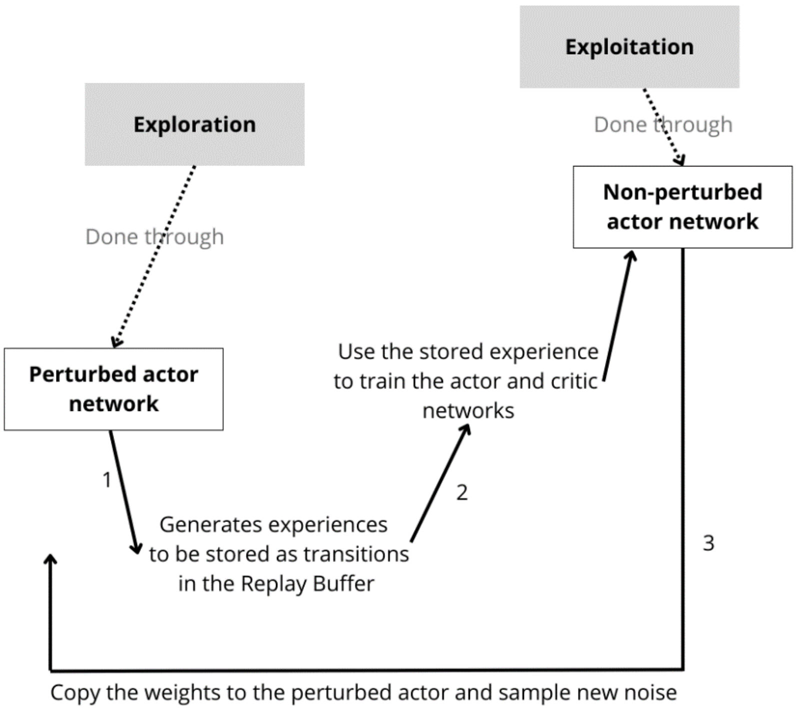
Exploitation
Exploration
Done through
Done through
Non-perturbed actor network
Perturbed actor network
Use the stored experience to train the actor and critic networks
1
2
3
Generates experiences to be stored as transitions in the Replay Buffer
Copy the weights to the perturbed actor and sample new noise

Figure 2. The mechanism of parameter space exploration noise

Indeed, picking a proper scale for noise in the parameter space is much harder than picking it for the action space. The authors' solution to this is to proceed with an adaptive noise scaling; they relate the scale of the parameter space noise to the variance it induces in the action space. Essentially, they define a distance measure between perturbed and non-perturbed policy in action space and adaptively increase or decrease the noise scale denoted by σ. For each new episode $k + 1$, we set the noisy actor parameters as $\widetilde{\omega} = \omega + \mathcal{N}(0, \sigma_{k+1}^2)$ where:

$$\sigma_{k+1} = \begin{cases} \alpha\sigma_k \,, if \; d(\pi, \tilde{\pi}) \leq \varepsilon \\ \dfrac{1}{\alpha}\sigma_k \,, otherwise \end{cases}$$

α being the scaling factor and ε our desired action space standard deviation, and the distance measure $d(.,.)$ defined for the DDPG algorithm is:

$$d(\pi, \tilde{\pi}) = \sqrt{\mathbb{E}[(\pi(s) - \tilde{\pi}(s))^2]}$$

This expectation is estimated from a batch of states, meaning each time the perturbed actor is used to pick an action, we compute and store the action picked by the non-perturbed actor to compute this distance and decide at the beginning of each episode if the noise to the parameter space should be reduced or increased.

Since $d\left(\pi, \pi + \mathcal{N}(0, \sigma^2)\right) = \sigma$, setting $\varepsilon = \sigma$ as the threshold results in an *effective* action space noise that has the same standard deviation as regular gaussian action space noise.

Results

Unfortunately, the DDPG agent fails to converge in both approaches. For instance, we plot the cumulative reward given by the target actor network at the end every episode for the case of no trading costs and daily hedging as a function of the episode:

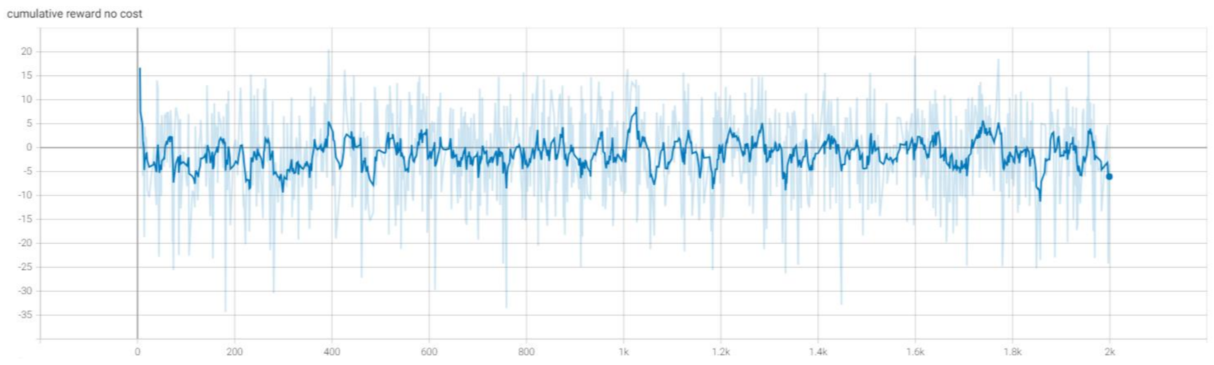

Figure 3. DDPG rewards as a function of episode number

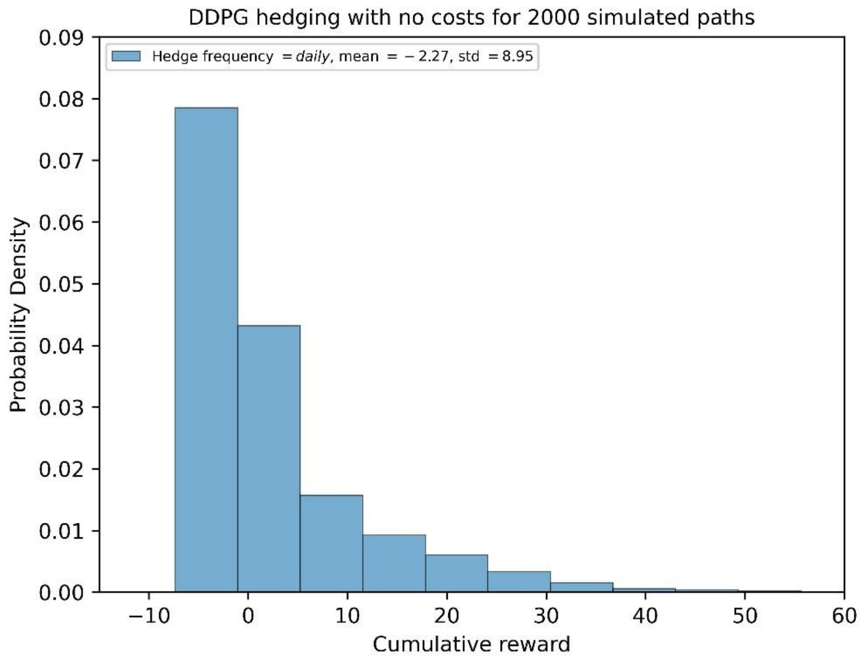

Figure 4. Cumulative reward distribution

We expect the reward in this case to converge to 0, but the reward continues to vary wildly regardless of how long it trains.

Indeed, the variance of the cumulative reward is not being minimized, thus that of the P&L is also not minimized during training. This doesn't make for a reliable hedging strategy, as it is too risky, which is clear from the agent failing to converge.

Multiple things have been tried to try to resolve this issue. I experimented with multiple network architectures; number of hidden layers and their depth, adding and removing normalization layers, trying multiple noise levels, trying different output activation functions and hidden layer activation functions. I tried multiple learning rates for both the actor and critic networks. Ultimately, the problem persisted, and the agent was unable to learn. Furthermore, I even tried supplementing the agent with transitions using a delta hedging policy. Multiple trading cost and risk aversion levels have also been tried, to no avail.

Conclusion

Using reinforcement learning to hedge options is a rich and diverse field of study. In my project I attempted to recreate the approach of hedging a European call option in the presence of trading costs using the DDPG algorithm.

Although I haven't been able to recreate good results, working on this project has been an extremely valuable experience. This has been my first experience working on a reinforcement learning project. The task of creating an RL training environment from scratch allowed me to deepen my understanding and expertise of the python language, especially object-oriented programming. Invaluable lessons have been learned when it comes to working on a project where changes to the code need to be tracked to try to find the root cause of the problem while staying organized.

Indeed, the more general question is that of the global convergence of a DDPG agent, which is not always guaranteed. For this reason, changing the training algorithm could be the only viable solution. One such algorithm could be the Dueling DQN algorithm, which would necessitate a discretization of the action space, but which has better convergence guarantees.

Appendices

Appendix A

An RL learning process can be represented as a discrete-time stochastic process, particularly as a finite Markov Decision Process (MDP). An MDP is characterized by:

Its state set $\mathcal{S} = \{s_0, \dots\}$ which fully describe the environment at each time-step.

Its action set $\mathcal{A} = \{a_0, \dots\}$ where every action a_t is taken after observing the state s_t.

In addition, an MDP is defined by the transitions of the environment at each time-step. A transition includes the reward that the agent receives after taking an action. The reward $\mathcal{R}$ at time $\mathrm{t} + 1$ is a function of the state and action at time t :

$$\mathcal{R}_s^a = \mathbb{E}[r_{t+1} | s_t = s, a_t = a]$$

The cumulative reward is the sum of all the rewards that the agent receives at each time-step. We can analogously define the future reward at a given time step t. Since the MDP process is stochastic, this future sum is discounted to account for the fact that the agent is not concerned in an equal manner in each time-step's reward. Particularly, in the case of financial applications, the agent is interested in a more immediate reward. This sum then can be defined as

$$G_t = \sum_{k=0}^{\infty} \gamma^k r(s_{t+k+1}, a_{t+k+1}) \text{ ,} where\ \gamma \in (0,1]$$

In many applications, such as that of hedging derivative products, the sum only extends up until a final time T.

Given a state, an action, we can define the state transition probability $\mathcal{P}^a$ matrix of an MDP as

$$\mathcal{P}_{s,s'}^a = \mathbb{P}[s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a]$$

Which is the probability that the environment will move to state s' with a reward $r = r(s,a)$ given that the agent took action a in state s. If this transition probability defines the dynamics of the process.

The objective of the RL agent is to learn a policy function, which is essentially a probabilistic guideline dictating the agent's action choices based on the current state:

$$\pi : \mathcal{S} \times \mathcal{A} \to [0,1], \qquad \pi(a|s) = \mathbb{P}(a_t = a | \, s_t = s)$$

Such that the cumulative reward is maximized. This policy fully characterizes an agent's behavior, does not depend on the history of states but only on the current state, and is stationary in the sense that a given state produces the same action regardless of the time the agent encounters this state. Given this policy, we can write:

$$\mathcal{P}^{\pi}_{s,s'} = \sum\nolimits_{a \in \mathcal{A}} \pi(a|s)\mathcal{P}^{a}_{s,s'}$$

$$\mathcal{R}^{\pi}_{s} = \sum\nolimits_{a \in \mathcal{A}} \pi(a|s)\mathcal{R}^{a}_{s}$$

Sutton and Barto (1998) define two value functions for a given policy π. The state-value function $v_{\pi}(s) = \mathbb{E}_{\pi}[G_t |\, s_t = s]$ which is the expected future reward the agent hopes to get starting from state s and following policy π, and the action-value function $q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t |\, s_t = s, a_t = a]$ which is the expected future reward the agent hopes to get starting from state s, then taking the action a given by the policy π.

Bibliography

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

Cao, J., Chen, J., Hull, J. C., & Poulos, Z. (2020). Deep hedging of derivatives using reinforcement learning. *The Journal of Financial Data Science*, *3*(1), 10–27. https://doi.org/10.3905/jfds.2020.1.052

Du, J., Jin, M., Kolm, P. N., Ritter, G., Wang, Y., & Zhang, B. (2020). Deep reinforcement learning for option replication and hedging. *The Journal of Financial Data Science*, *2*(4), 44–57. https://doi.org/10.3905/jfds.2020.1.045

Giurca, A., & Borovkova, S. (2021). Delta Hedging of Derivatives using Deep Reinforcement Learning. *Social Science Research Network*. https://doi.org/10.2139/ssrn.3847272

Kolm, P. N., & Ritter, G. (2019). Dynamic Replication and Hedging: *A Reinforcement Learning Approach*. *The Journal of Financial Data Science*, *1*(1), 159–171. https://doi.org/10.3905/jfds.2019.1.1.159

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv (Cornell University)*. http://export.arxiv.org/pdf/1509.02971

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., & Andrychowicz, M. (2018). Parameter Space noise for exploration. *International Conference on Learning Representations*. https://openreview.net/pdf?id=ByBAl2eAZ

Sutton, R. S., & Barto, A. (1998). Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5), 1054. https://doi.org/10.1109/tnn.1998.712192