Optimal Hedging via Deep Reinforcement Learning with Soft Actor-Critic

by

Zhihao Shan

An honors thesis submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science

Business and Economics Honors Program

NYU Shanghai

May 2024

Professor Marti G. Subrahmanyam
Professor Christina Wang
Professor Ye Jin

Faculty Advisers

Professor Shengjie Wang

Thesis Adviser

Preface

In the rapidly evolving world of finance, the ability to mitigate risks effectively is more crucial than ever. My journey of double majoring in Data Science and Business&Finance has been driven by a fascination with the intersection of finance and technology. With a strong academic background in both disciplines, I embarked on a project to explore how cutting-edge machine learning techniques can revolutionize financial hedging strategies.

This thesis, "Optimal Hedging via Deep Reinforcement Learning with Soft Actor-Critic," is inspired by the potential of deep reinforcement learning (DRL) to provide dynamic and efficient solutions to complex financial problems. My curiosity was piqued by the Soft Actor-Critic (SAC) algorithm's success in various complex tasks, which led me to investigate its application in financial hedging.

The target audience for this work includes academics in finance and machine learning, financial practitioners looking for innovative risk management strategies, and anyone interested in the practical applications of artificial intelligence in the financial industry. This thesis demonstrates the potential of DRL in optimizing hedging strategies, which is vital for managing financial risks and making informed investment decisions.

Acknowledgements

I extend my deepest gratitude to my faculty advisers, Professors Marti G. Subrahmanyam, Christina Wang, and Ye Jin, and my thesis advisor Professor Shengjie Wang, whose guidance and insights were invaluable throughout this research journey. Their expertise and encouragement have been pivotal in shaping this thesis.

I also wish to thank my peers and the academic community at NYU Shanghai for fostering a collaborative and intellectually stimulating environment. Their support and constructive feedback have been instrumental in refining my work.

Lastly, I acknowledge the resources provided by NYU Shanghai, which were essential for conducting the research presented in this thesis. Their commitment to academic excellence and research innovation has greatly enriched my scholarly endeavors.

Abstract

The thesis addresses the problem of optimal hedging in financial markets using deep reinforcement learning, specifically through the Soft Actor-Critic (SAC) algorithm. It explores the interesting challenge of applying sophisticated machine learning techniques to automate and potentially optimize hedging strategies in the volatile financial trading landscape. The approach involves leveraging the SAC algorithm's stability and efficiency in learning policies within a highly uncertain environment. The paper provides evidence of the efficacy of reinforcement learning in managing complex financial objectives and offers a comparative analysis between SAC-based hedging strategies and other existing methods. It aims to bridge the gap between advanced machine learning techniques and practical financial applications, contributing to both academic discourse and practical financial risk management.

Keywords

Optimal Hedging; Deep Reinforcement Learning; Soft Actor-Critic; Financial Markets; Derivatives; Risk Management;

Contents

1 Introduction

In the realm of financial markets, the concept of hedging is pivotal to managing risk and ensuring portfolio stability. It refers to the practice of strategically opening positions to offset potential losses in investments, which is a fundamental aspect of modern financial theory and practice. Traditionally, to hedge against the inherent risks in derivatives trading, traders keep a close watch on certain metrics known as "Greek letters." Among Greek Letters, Delta holds significant importance, representing the sensitivity of a derivative's price to changes in the underlying asset's value. Traders often aim to achieve positions that are delta-neutral or nearly so.

The dynamic nature of an option's delta throughout its duration necessitates regular adjustments to the trader's stance. In an ideal scenario without transaction costs or other hurdles, the most effective strategy would involve continuous rebalancing. Adhering to the hypothesized asset price model, such continuous adjustments ensure the expected hedging expenses equaling the option's theoretical valuation [3]. However, real-world factors like transaction expenses and various trading slippage (collectively termed "trading costs") necessitate a deviation from this ideal strategy. Additionally, adjustments to the underlying asset's position are made at intervals, rather than in a continuous fashion.

Machine learning, particularly reinforcement learning (RL), stands at the forefront of this revolution, offering dynamic and adaptive approaches to investment strategies that traditional statistical models struggle to match. RL aims to learn optimal actions through interaction with an environment to maximize some notion of cumulative reward. Among RL algorithms, the Soft Actor-Critic (SAC) algorithm has garnered significant attention due to its stability and efficiency in learning policies. The SAC, an off-policy algorithm that optimizes a stochastic policy in an entropy-augmented reinforcement learning framework, has demonstrated state-of-the-art performance in continuous action spaces, making it an attractive choice for financial applications where decision spaces are often complex and multi-dimensional.

This paper explores the application of the SAC algorithm to hedging strategies within the context of hedging a call option. By leveraging the robustness of SAC, the research

aims to unearth insights into the viability of reinforcement learning as a means to automate and potentially optimize hedging. The investigation is grounded in the hypothesis that SAC can outperform traditional hedging techniques and other RL algorithms like Deep Deterministic Policy Gradient, providing a more reactive and informed approach to risk mitigation in the volatile landscape of financial trading.

The contribution of this paper is threefold. First, it demonstrates the implementation of SAC in a financial hedging scenario, which, to the best of my knowledge, has not been explored extensively. Second, it provides evidence of the efficacy of reinforcement learning in managing complex financial objectives without passing the required knowledge to the RL agent. Third, it offers a comparative analysis between SAC-based hedging strategies and other existing methods, thus underscoring the advantages and potential limitations of the proposed approach.

As such, this paper aims to bridge the gap between sophisticated machine learning techniques and practical financial applications. It seeks to not only contribute to academic discourse but also provide a blueprint for financial practitioners considering the integration of advanced AI methodologies into their risk management arsenals. In doing so, it paves the way for further innovations in the intersection of machine learning and finance.

2 Related Work

2.1 Traditional Hedging Under Transaction costs

The exploration of hedging strategies in the presence of transaction costs has been a critical area of research in quantitative finance. Leland (1985) pioneered this field by developing an option hedging strategy with the presence of transaction costs, emphasizing the need for adjusted hedging intervals, despite that the method performs poorly when the option is at the money and close to expiration (which causes a high gamma) [14]. Boyle and Vorst (1992) furthered this work by searching into option pricing under transaction costs [4].

Grannan and Swindle (1996) developed strategies to minimize transaction costs in hedg-

ing, using the Mean-Variance Criterion and evaluated the strategies using simulations, focusing more on the computational perspective[9]. Toft (1996) also explored the mean-variance trade-off in hedging with transaction costs, highlighting the balance between risk and cost [21]. Whalley and Wilmott (1997) conducted an asymptotic analysis of optimal hedging models, offering insights into the long-term impacts of transaction costs on hedging strategies [22]. Lastly, Martellini (2000) proved the efficiency of his option hedging strategy by both a better hedge with the same transaction cost and a lower transaction cost with the same hedge. [16].

A common feature that these papers share is that they all considered proportional trading costs. Nevertheless, trading cost increases faster than linearly with respect to the size of the trade. Several works have been conducted to manage this non-linearity. Rogers and Singh (2010) were the first to consider option hedging under illiquidity and formed it into an optimal control problem [19]. Almgren and Li (2016) pioneered option hedging under non-linear transaction costs caused by both temporary and permanent market impact [1]. Bank, Soner, and Voß (2017) assumed quadratic trading costs caused by the market impact of larger orders and also used optimal control to solve the problem [2]. All these works are based on traditional mathematical approaches such as optimal control. Note that these approaches heavily rely on market assumptions such as modeling stock price as Geometric Brownian Motion and there are no easy ways to improve given real market data. Nevertheless, option hedging can be treated as serial decision-making, which can be addressed by state-of-the-art deep reinforcement learning algorithms.

2.2 Reinforcement Learning and Deep Reinforcement Learning

Reinforcement learning (RL) has emerged as a powerful methodology in the realm of machine learning, enabling agents to make serial decisions and learn optimal policies based on environmental feedback. Markov Decision Processes (MDPs) provide a mathematical framework for modeling decision-making in such situations. MDPs are characterized by a set of states S, a set of actions A, a transition function $P(s'|s,a)$ defining the probability of transitioning to state s' from state s after taking action a, and a reward function

$R(s, a, s')$ that assigns a reward for each transition [20]. There are multiple algorithms and they all seek to learn a policy that maximizes the total reward over all successive steps. Such algorithms typically use a function $Q(s, a)$ to map current state s and action taken a to the expected sum of future rewards. However, when state space and action space become larger or continuous, it is no longer feasible. Neural networks are then used to approximate the Q-function and other functions needed in training to form a family of Deep Reinforcement Learning (DRL) algorithms.

Actor-critic methods are a cornerstone of DRL, balancing the strengths of value-based and policy-based approaches. In this framework, the "actor" proposes actions based on the current policy, while the "critic" assesses these actions using a value function, guiding the actor towards better decision-making. One of the key advancements in this area is the Deep Deterministic Policy Gradient (DDPG) algorithm, which combines the strengths of policy gradient methods with the stability of Q-learning, suitable for continuous action spaces [15].

Soft Actor-Critic (SAC) is another significant development in actor-critic methods, emphasizing maximum entropy RL. This approach encourages exploration by not just aiming to maximize the expected return but also rewards the entropy of the policy. This leads to more robust and sample-efficient learning, especially in environments with complex and high-dimensional action spaces [10]. Both algorithms proved to handle sophisticated problems such as robotics and gaming.

2.3 Hedging with Deep Reinforcement Learning

As mentioned in former sections, option hedging can be treated as serial decision-making. Various research has been done at the intersection of reinforcement learning and financial hedging strategies. Halperin (2017) first introduces the Q-Learning in the Black-Scholes(-Merton) Worlds (QLBS), which utilizes Q-Learning, a reinforcement learning technique, to price options through portfolio replication in an environment without transaction costs [12]. Buehler et al. (2018) proposed "Deep Hedging," a method that employs deep reinforcement learning to hedge a call option with transaction costs. Buehler used hedging

error and CVaR to evaluate his policies on the Black-Scholes Model, Heston Model and S&P 500 index [5].

Kolm and Ritter (2019) further expanded the application of reinforcement learning in hedging through a practical lens. Their work aims to maximize the utility of the investor using DQN algorithm because market actors are often risk-averse rather than risk-neutral [13]. Du et al. (2020) developed their work to employ PPO algorithm on the task, which proved to be more sample efficient when training [8]. Cao et al. (2020) delve into DDPG algorithm for hedging, highlighting their adaptability to different asset-price processes and conditions, including changing volatility. This research emphasizes the computational advantages and flexibility of deep learning in adapting to complex market dynamics [7].

Canelli et al. (2023) approached hedging through a different type of reinforcement learning technique-contextual k-armed bandit problem. Their research contrasts with traditional Q-learning by focusing on reducing hedging errors and improving efficiency, offering a nuanced view of how RL hedging policies can be improved during the process of hedging [6].

Halperin, Kolm, Du and Canelli's works were all based on assumptions about price and volatility processes such as Brownian Motion. Mikkilä and Kanniainen (2021) provide a real-world perspective by applying deep reinforcement learning to hedge S&P500 index options using real intra-day data. Their results showcase the practical effectiveness of deep reinforcement learning in managing real-world options, highlighting its potential for real-time, data-driven decision-making in financial hedging [17]. Lastly, all the above works were about delta hedging and all used delta hedging as a benchmark, while Cao et al. (2023) introduced deep distributional reinforcement learning for gamma and vega hedging, considering the effects of evolving volatility[6].

Du et al. (2020) found that when the entropy of the policy of PPO was rewarded large enough, the resulting policy was more likely to be more financially intuitive and stable than when it is not [8]. Nevertheless, PPO, as an on-policy algorithm, has its disadvantages. Off-policy algorithms are prized for their flexibility, allowing them to learn from various sources and separate exploration from policy optimization. Furthermore, their stability,

aided by replay buffers, ensures robust learning in noisy environments like finance. Additionally, these algorithms efficiently use computational resources, facilitating rapid model development through parallelized training. Particularly in fields like finance, where data is precious, off-policy methods excel by maximizing learning from limited interactions, making them highly suitable for real-world applications. SAC is a state-of-the-art off-policy DRL algorithm that integrates policy entropy maximization into its objective function. The entropy bonus in SAC is not just an add-on for encouraging exploration but also directly influences the policy improvement step. These policies encourage exploration and prevent premature convergence to sub-optimal policies, a trait particularly advantageous in the stochastic and unpredictable financial market. With a policy of larger entropy, the agent is expected to behave well in scenarios that are less similar to training data, ensuring the stability of the resulting policy.

3 Methodology

3.1 Problem Setting

State Space

The state must encapsulate the information pertinent to arriving at the best possible decision. Details irrelevant to the task, or those inferable from existing state variables, are extraneous and thus omitted. When considering the replication of European options with various exercise prices, the appropriate state space is formulated as follows:

$$S := \mathbb{N} \times \mathbb{R}_+^3 = \{(n, S, \tau, C) \mid n \in \mathbb{N}, S \in \mathbb{R}^+, \tau \in \mathbb{R}^+, C \in \mathbb{R}^+\}$$

Herein, at any given moment t, the observer is presented with a four-dimensional state vector $s_t = (S_t, \tau, n_t, C)$, where S_t represents the stock's price at the instance t; $\tau := T - t > 0$ denotes the option's time to maturity; n_t represents the current number of shares possessed; C denotes the price of the call option (for one share of the underlying). It is crucial to note that the state need not comprise the 'Greeks' of the option, for these are derivative functions of the variables already within the agent's reach through the state.

With a robust dataset for training, it is anticipated that the agent will autonomously learn such complex functions as required.

Action Space

After observing the state, the agent takes an action a_t by choosing the position of the underlying asset to hold after trading from the action space:

$$A := \{(H)| - 1 \leq H \leq 1\}$$

The action is then scaled according to L which represents the number of shares of the underlying asset that one option contract corresponds to. It is designed to be between -1 and 1 for easier training of the DRL agent.

Transition Function

With the input state and action of the former step, the environment will pass out a new state according to certain probability distributions. The transition of the stock price is first set to be a Geometric Brownian Motion. The price of the call option is computed using the BSM formula [3]. Below is the transition function for each component of the state.

$$n_{t+1} = At * L$$

$$S_{t+1} = S_t \exp\left(\left(\mu - \frac{\sigma_t^2}{2}\right) + \sigma\sqrt{\Delta t} Z_t\right)$$

$$\tau_{t+1} = \tau_t - \Delta t$$

$$C_{t+1} = S_{t+1}\Phi(d_1') - S_0 e^{-r\tau_{t+1}}\Phi(d_2')$$

where

$$d_1' = \frac{\log\left(\frac{S_{t+1}}{S_0}\right) + \left(r + \frac{\sigma^2}{2}\right)\tau_{t+1}}{\sigma\sqrt{\tau_{t+1}}}, d_2' = d_1' - \sigma\sqrt{\tau_{t+1}}$$

Following the practice of Cao et al. (2023), the volatility of a stock is modeled to be another geometric Brownian motion as they experimented with vega hedging, which would not be necessary with fixed volatility. The transition function for n and τ would be the same as before while the transition of stock price and option price is as follows:

$$S_{t+1} = S_t \exp\left(\left(\mu - \frac{\sigma_t^2}{2}\right) + \sigma_t\sqrt{\Delta t}Z_1\right)$$

where

$$\sigma_{t+1} = \sigma_t \exp\left(-\frac{v^2}{2}\Delta t + v\sqrt{\Delta t}Z_2\right)$$

and

$$Z_2 = \rho Z_1 + \sqrt{1-\rho^2}Z_2'$$

Z_1 and Z_2' are independent standard normal distributions, ρ is the correlation coefficient between Z_1 and Z_2, and v is the volatility of volatility.

The price of the call option is the same but uses the implied volatility σ_{imp} to replace the constant volatility in the last setting. The implied volatility is calculated using a special case of the SABR model developed by Hagan et al. (2002) [11]. Given the inputs S (spot price), τ (time to maturity), σ (volatility), K (strike price), r (risk-free rate), v (volatility of volatility), ρ (correlation), and β (default value of 1), the SABR volatility model can

be expressed as follows:

$$
\begin{aligned}
F &= S \cdot e^{r\tau} \\
x &= (F \cdot K)^{\frac{1-\beta}{2}} = 1 \\
y &= (1-\beta) \cdot \log\left(\frac{F}{K}\right) = 0 \\
A &= \frac{\sigma}{x \cdot \left(1 + \frac{y^2}{24} + \frac{y^4}{1920}\right)} = \sigma \\
B &= 1 + \tau \left(\frac{(1-\beta)^2 \cdot \sigma^2}{24 \cdot x^2} + \frac{\rho\beta v\sigma}{4x} + \frac{v^2(2-3\rho^2)}{24}\right) \\
\Phi &= \frac{v \cdot x}{\sigma} \cdot \log\left(\frac{F}{K}\right) \\
\chi &= \log\left(\frac{\sqrt{1 - 2\rho\Phi + \Phi^2} + \Phi - \rho}{1-\rho}\right) \\
\sigma_{imp} &= \begin{cases} \frac{\sigma \cdot B}{F^{1-\beta}} = \frac{\sigma \cdot B}{F} & \text{if } F = K, \\ \frac{A \cdot B \cdot \Phi}{\chi} = \frac{\sigma \cdot B \cdot \Phi}{\chi} & \text{otherwise.} \end{cases}
\end{aligned}
$$

Reward Function

Automatic hedging is described as utilizing trained reinforcement learning (RL) agents to manage the hedging activities for specific derivative positions. The agents maintain a non-tradable short option position, but they are permitted to engage in trading other non-option assets for replication purposes. In an ideal scenario without trading frictions and with the ability for continuous trading, a perfect hedging dynamic portfolio could exist, rendering the net portfolio variance null. However, this paper considers the practical scenario of trading frictions and discrete trading, aiming to minimize both variance and costs.

The article aims to formulate the reward function for the RL agent, assuming it operates under a quadratic utility framework. The optimal portfolio for the agent aligns with a mean-variance optimization model, encapsulated by risk aversion factor κ, as follows:

$$
\max\left(E[w_T] - \frac{\kappa}{2}V[w_T]\right)
$$

In this model, w_T represents the aggregate of wealth increments δw_t,

$$w_T = w_0 + \sum_{t=1}^{T} \delta w_t$$

where $E[w_T]$ equals w_0 plus the sum of expected increments $E[\delta w_t]$. Assuming the increments of wealth are independent over time, meaning $\text{cov}(\delta w_t, \delta w_s) = 0$ for $s \neq t$,

$$V[w_T] = \sum_{t} V[\delta w_t].$$

Options in a complete market scenario are redundant as they can be precisely replicated by a continuous trading strategy. Nevertheless, in the practical world, the variance of the difference between an option and its replicating portfolio is non-zero. Our agent seeks to balance between cost and variance by optimizing the objective:

$$\min \sum_{t=0}^{T} \Big(E[-\delta w_t] + \frac{\kappa}{2} V[\delta w_t] \Big)$$

Assuming a random walk for the log price process, wealth increments can be broken down into $q_t - c_t$, where q_t is the random walk component, and c_t represents the total trading costs. Thus, the expected increment in wealth is simply the negation of the expected cost $E[-\delta w_t] = E[c_t]$, leading to a balancing act between cost and variance. More frequent hedging can reduce variance but at the cost of increased trading expenses.

Ritter (2017) demonstrated that by selecting an appropriate reward function, maximizing $E[u(w_T)]$ could be transformed into an RL problem [18]. The periodical reward, approximated by the equation below, facilitates a mean-variance optimization through RL training:

$$R_t := \delta w_t - \frac{\kappa}{2} (\delta w_t)^2$$

Furthermore, trading cost is defined by a quadratic function with respect to trading volume to capture the increasing market impact of larger orders:

$$\text{trading cost}_t = \text{trading cost parameter} \cdot S_t \cdot (|\Delta n_{t+1}| + 0.01 \cdot {\Delta n_{t+1}}^2)$$

where at time t:

$$w_t = -L * (C_{t+1} - C_t) + n_{t+1} * (S_{t+1} - S_t) - \text{trading cost}_t$$

Implementing this reward mechanism effectively trains the agents to maximize expected utility, translating into automatic hedgers proficient in managing the balance between hedging costs and variance.

Benchmark Performance

The performance of the trained SAC agent is compared to the Delta hedging policy and trained DDPG agent, which has been proved to perform well in Cao et al. (2020). With the current state denoted by S_t, delta hedging policy is defined by:

$$\pi_{DH}(S_t) = -\Delta(S_t)$$

where $\Delta(S_t)$ is the Black-Scholes-Merton delta of the current option position. In our setting, we are always at a constant short position of the call option, so the delta would be the opposite number of the delta of the call option, which is bounded between 0 and 1. Note that the action will be scaled automatically in the environment.

3.2 Solution

Environment

The environment described in the Problem Setting section is implemented in Gymnasium using Python (see https://gymnasium.farama.org). In training and simulations, the environment is set with the following parameters: risk-free rate $r = 0$, drift term of stock price $\mu = 0$, volatility $\sigma = 0.15$ (annual), time to maturity $T = 10$ and 30 (days), trading frequency $D = 5$ (times per day), trading cost parameter $= 0.003$, shares of underlying corresponding to the call option $L = 100$, risk-averse level $\kappa = 0.1$. When the stock price is set to have stochastic volatility, volatility of volatility $v = 0$ and $v = 0.5$ (annual), and

correlation between Z_{1t} and Z_{2t}, $\rho = -0.4$, which captures the leverage effect. In each simulation period, there are $T \times D$ timesteps. At each timestep, the agent will observe the current state and pass the resulting action to the environment, before which the environment returns the next state and the reward of the current step until the final step is reached. Note that the drift term of stock price μ is intentionally set to zero because when trained with large samples, the agent might learn that the stock price is expected to increase and simply takes the maximum long position of the stock as if it is an arbitrage opportunity. When training the agent with real-world data, stock returns should be residuals whose mean is zero to avoid this issue. Note that since the risk-free rate is set to be zero, the price of the call option is lower than in real-world cases.

Algorithms

The DRL algorithm employed in the research is Soft Actor-Critic, which belongs to the Actor-Critic methods. Actor-Critic methods blend the characteristics of value-based and policy-based strategies. These methods employ two components: an actor, which dictates the policy of actions, and a critic, which evaluates these actions. The actor, represented by the policy function $\pi(a|s, \theta)$, maps the current state to action, with θ being the parameters that define this policy. The actor's objective is to optimize this policy to maximize the sum of future rewards.

The critic, on the other hand, assesses the actions taken by the actor using the value function $V(s, w)$, where w are the parameters of the value function. This function estimates the expected return from a given state, providing feedback on the quality of the actions chosen by the actor. The interaction between actor and critic is mediated through the Temporal Difference (TD) error, $\delta = r+\gamma V(s', w)-V(s, w)$, which quantifies the difference between the estimated returns and the actual returns, where r is the immediate reward, s' is the subsequent state, and γ is the discount factor that balances the importance of immediate and future rewards.

During the learning process, the actor updates its policy to favor actions that lead to higher rewards, based on the guidance from the critic. This is achieved by adjusting the policy parameters θ in the direction that maximizes the expected return, utilizing

the gradient ascent method. The update rule for the actor's parameters is driven by the gradient of the policy's performance, expressed as $\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \log \pi(a|s,\theta) \cdot \delta]$.

Simultaneously, the critic updates its value function parameters w to better estimate the true returns, minimizing the squared TD error. This is typically done using gradient descent, with the update rule $\nabla_w L(w) = -\mathbb{E}[\delta \nabla_w V(s,w)]$. Through this continuous feedback loop, the critic helps to refine the actor's policy decisions, leading to a more effective learning process in DRL environments. The symbiotic relationship between the actor and the critic facilitates a more stable and efficient approach to learning optimal policies in complex environments.

Building on the foundation of Actor-Critic methods, Soft Actor-Critic (SAC) emerges as a more advanced approach that incorporates the principles of entropy maximization to achieve more robust and efficient learning [10].

In SAC, the policy is trained to maximize not only the expected sum of future rewards but also the entropy of the policy itself. This entropy term, which measures the randomness of the policy, encourages the actor to explore a wider range of actions, thus preventing premature convergence to suboptimal policies. The objective function in SAC includes an entropy term, H, and is given by:

$$J(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \alpha H(\pi(\cdot|s_t)) \right) \right]$$

Here, α is a temperature parameter that controls the relative importance of the entropy term against the reward, allowing a balance between exploration (entropy maximization) and exploitation (reward maximization).

SAC uses two Q-functions, or critics, to reduce the overestimation bias common in Q-learning methods. These Q-functions are updated using the Bellman equation, and the policy is updated by minimizing the expected KL-divergence, leading to a policy that is more stable and less prone to local optima. The Python package "stable-baseline3" (see https://stable-baselines3.readthedocs.io) is used to implement the SAC algorithm in our environment. The pseudo-code of the SAC algorithm is shown in Algorithm 1.

Model Architecture and hyperparameters

Algorithm 1 Soft Actor-Critic [10]

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1, \phi_{\text{targ},2} \leftarrow \phi_2$
repeat
    Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
    Execute a in the environment
    Observe next state s', reward r, and done signal d to indicate whether s' is terminal
    Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
    If s' is terminal, reset environment state.
    if it's time to update **then**
        for j in range(however many updates) **do**
            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
            Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \tilde{a}' \sim \pi_\theta(\cdot|s')$$

            Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \quad \text{for} \quad i = 1, 2$$

            Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}(s)) - \alpha \log \pi_\theta(\tilde{a}(s)|s)\right),$$

            Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \quad \text{for} \quad i = 1, 2$$

        end for
    end if
until convergence

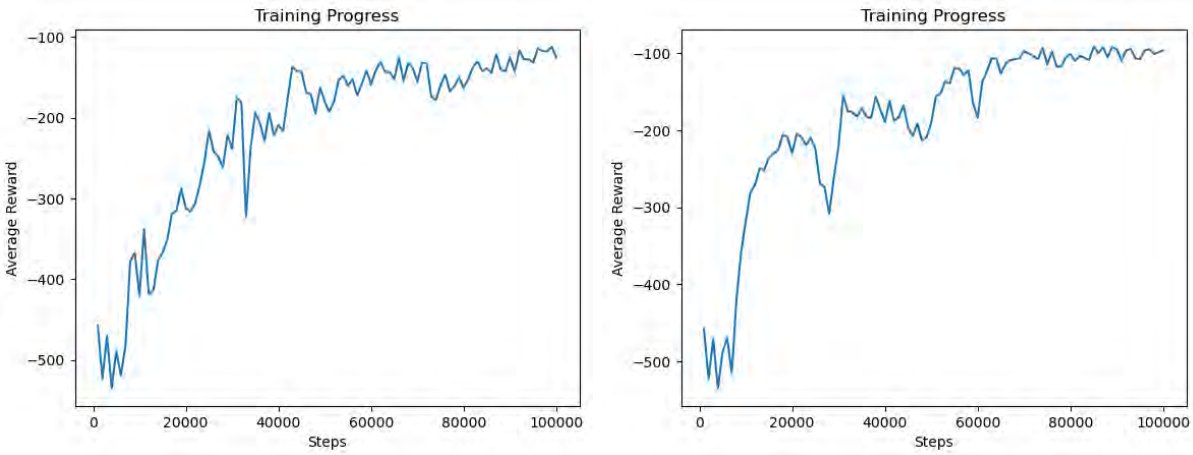

Figure 1: Average Reward versus Training Time for SAC and DDPG

There are five neural networks involved in the SAC algorithm, one for policy network, two for target Q-networks, and two for local Q-networks. All five neural networks are set to be a standard Multi-Layer Perceptron which has 3 hidden layers with 64 neurons on each layer and uses ReLU as an activation function. Such model architectures proved to balance training time and performance well in my tuning. The input size of the policy network is the dimension of state space which is 4 and the output size is 2, for the mean and the standard deviation of the action distribution. The resulting action would be sampled from this distribution. The input size of the Q-networks is 5, 4 for state space and 1 for action space, and the output size is 1. The optimizer to update the network parameters is Adam with the learning rate = 0.0001.

For the SAC algorithm itself, hyperparameters are tuned to the following: learning starts from 5000 timesteps, replay buffer size is 16384, batch size is 1024, discount rate $\gamma = 1$. All the other hyperparameters are the default values from the stable-baseline3 implementation. The training lasts for 10^6 timesteps whereas the policy is evaluated using 100 simulations every 1000 timesteps of training. DDPG algorithm is also employed to train an agent as another benchmark performance to assess the efficacy of the SAC agent.

4 Main Results

Both baseline and DRL (SAC and DDPG) policies are evaluated by running 10,000 simulations in the hedging environment. Their total rewards, total PnL, and total trading costs are stored for each simulation. Training and evaluation are done for both Geometric Brownian Motion stock price and Stochastic Volatility stock price.

4.1 GBM Stock Price Scenario

Figures 2 and 3 demonstrate the performance of different hedging strategies in 10,000 out-of-sample simulations. Rewards are the sum of rewards at each timestep in one episode which lasts for 10 trading days. P&L is the sum of w_t in one episode and costs are the sum of trading costs in one episode. From the KDE of rewards, SAC performs better than DDPG and Delta Hedging, which proves that the SAC agent is capable of learning better and more robust policy through its policy entropy maximization feature. Looking into details in its hedging behavior, the total P&L is less skewed compared with DDPG and far closer to zero than Delta Hedging. The DDPG agent hedges the least while the Delta Hedging agent hedges the most. From Table 1, it is shown that the SAC agent can achieve only 4% more std of PnL of Delta Hedging using only approximately half of the costs. While the DDPG agent generates even less trading costs, the std of PnL is far higher, proving that the SAC agent manages to achieve a better balance of cost and risk.

4.2 Stochastic Volatility Stock Price Scenario

Figures 4 and 5 demonstrate the performance of different hedging strategies in 10,000 out-of-sample simulations under stock price stochastic volatility. From the KDE of rewards, SAC still performs better than DDPG and Delta Hedging, which proves the robustness of our findings in an even more volatile scenario. Yet, the performance edge of the SAC agent is not as large as it was in the GBM case. Looking into details in its hedging behavior, the DDPG agent hedges roughly as much as the SAC agent proved by similar trading costs. Intuitively, investors should hedge less when volatility is stochastic since the benefits from

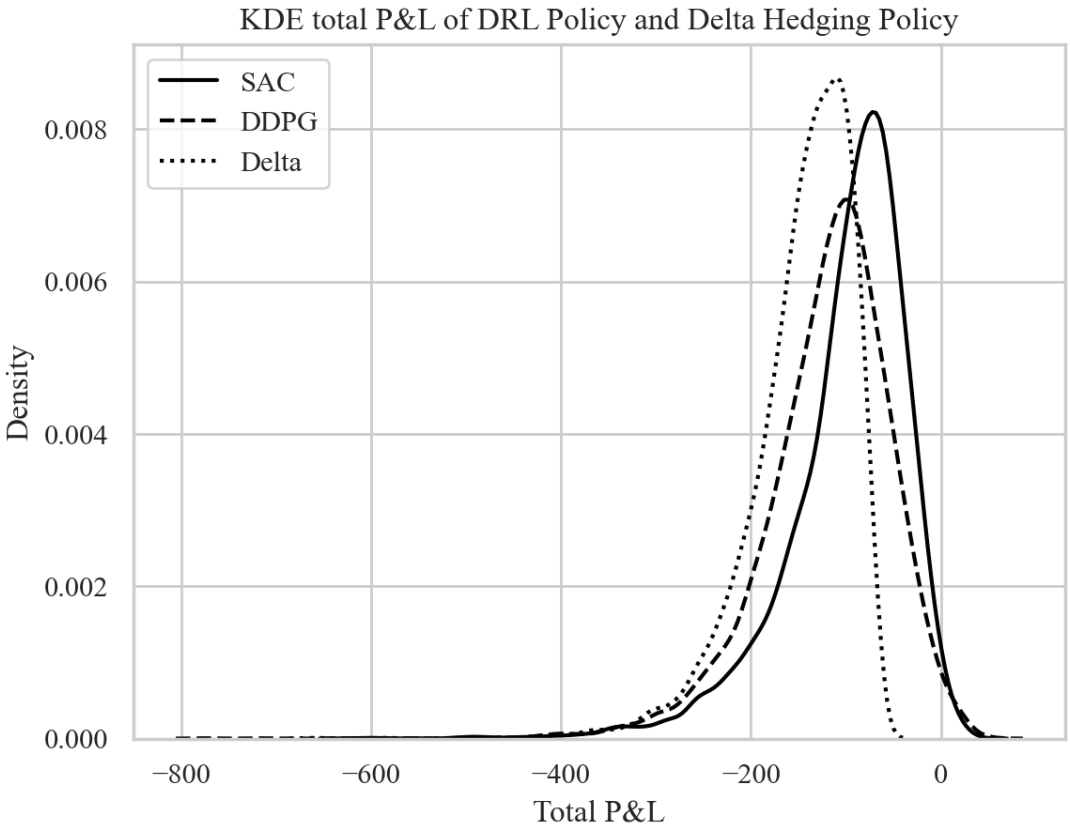

Figure 2: Kernel Density Estimate of Simulation Rewards (GBM stock price)

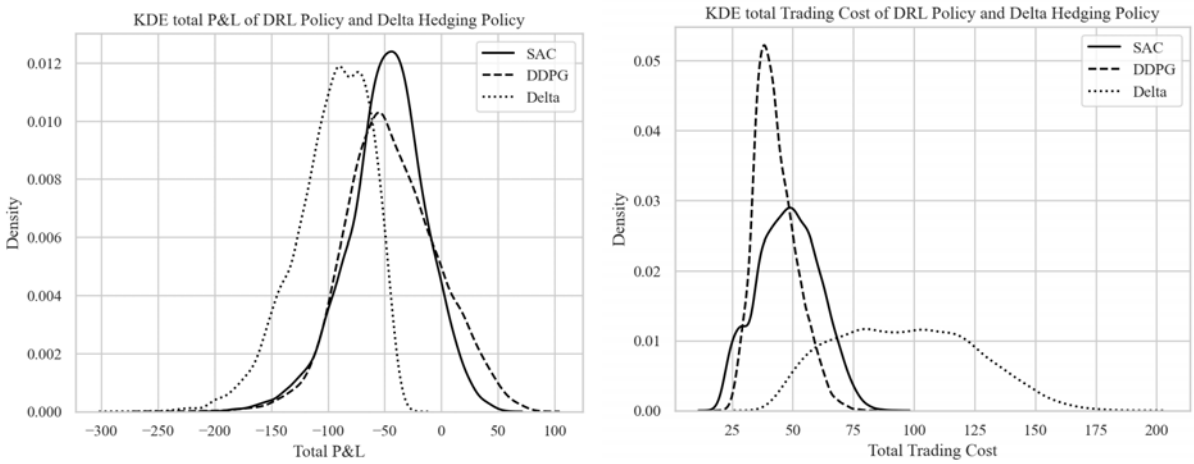

Figure 3: Kernel Density Estimate of Simulation P&L and Trading Cost (GBM stock price)

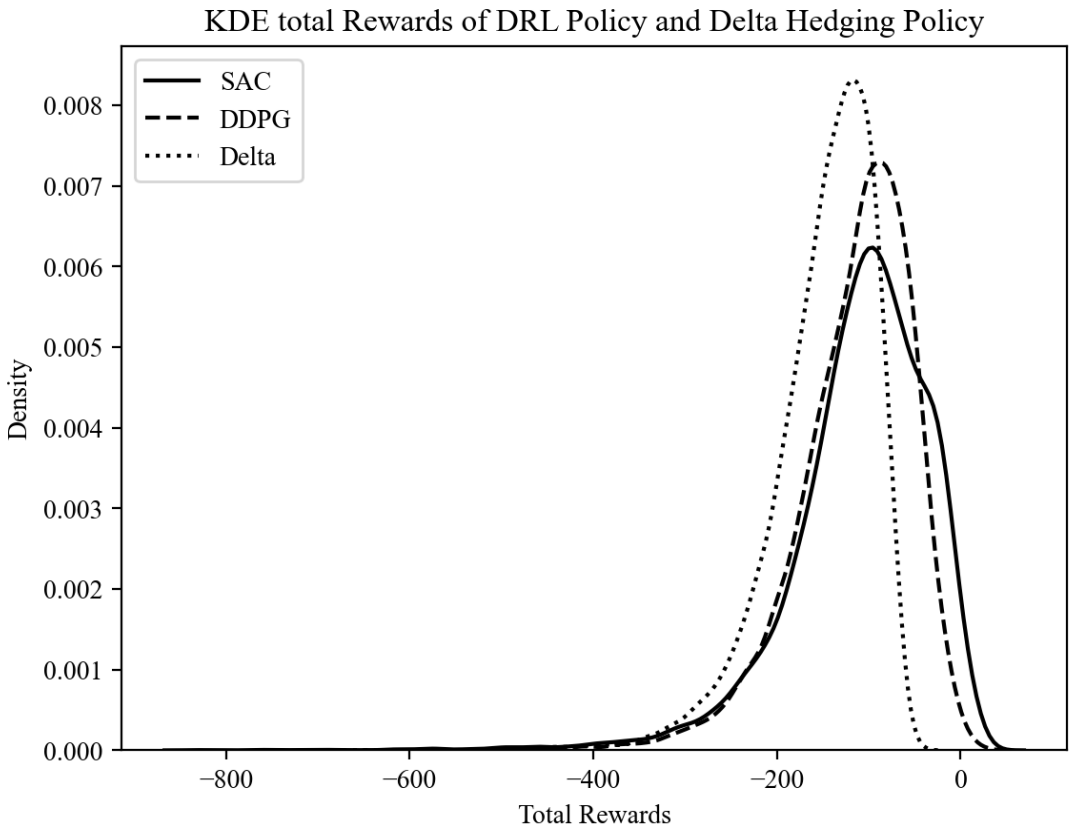

Figure 4: Kernel Density Estimate of Simulation Rewards (Stochastic Volatility stock price)

hedging have more uncertainties compared to the constant volatility case. The SAC agent manages to learn a policy that hedges less than the GBM case.

5 Discussion

Though the SAC agent appears to perform well in our stock price and option pricing models, it is not guaranteed to perform well or beat baseline strategies in real-world applications. There are a few techniques to integrate with the SAC algorithm that could potentially improve the transferability of DRL policies.

First, Bayesian methods could also be integrated into the SAC algorithm to handle uncertainty more effectively and prevent overfitting on training data, thus benefiting performance in real-world settings. By treating the parameters of the SAC model as probability distributions rather than fixed values, Bayesian SAC introduces a natural measure of uncertainty into the decision-making process. This approach is not only crucial for managing

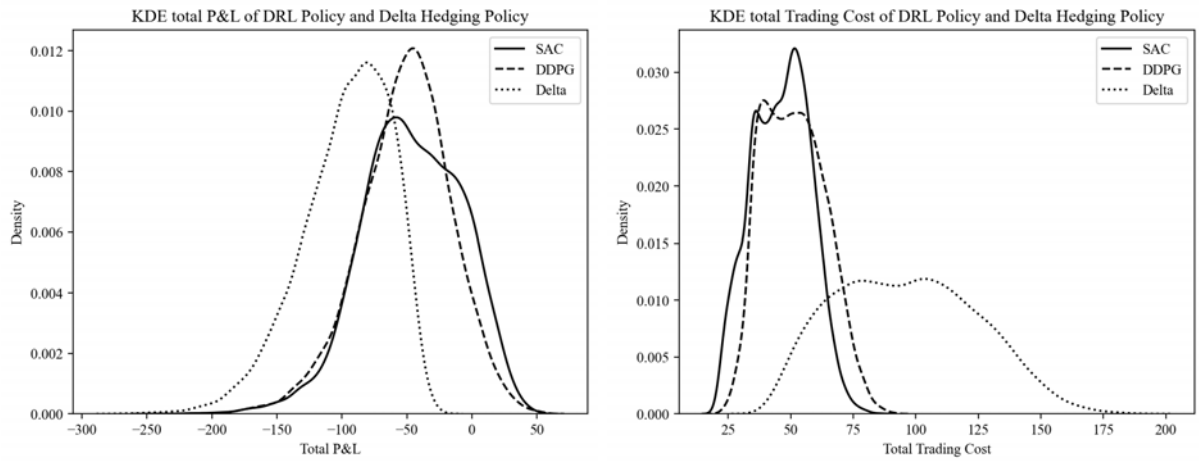

Figure 5: Kernel Density Estimate of Simulation P&L and Trading Cost (Stochastic Volatility stock price)

the inherent unpredictability in financial markets but also provides a regularization effect that prevents the model from fitting too closely to the training data. The probabilistic nature of Bayesian methods encourages exploration by adding an extra layer of stochasticity to the policy's actions, thus potentially discovering more robust hedging strategies. Additionally, Bayesian SAC can continuously update its belief about model parameters as more data becomes available, which is ideal for adapting to evolving market conditions and for achieving a balance between exploration and exploitation in policy development.

Second, Curriculum learning also offers a structured approach to progressively enhance the training of the SAC algorithm, making it well-suited for complex hedging tasks. By starting with simpler market scenarios and gradually introducing more complex and volatile conditions, curriculum learning mimics the human learning process, where foundational knowledge is built first before advancing to more challenging tasks. This approach can improve the SAC model's generalization capabilities and prevent it from getting stuck in suboptimal solutions early in the training process. For financial hedging, curriculum learning can help the model develop a robust understanding of basic hedging dynamics before tackling high-risk financial instruments or erratic market behaviors. Such a structured learning progression ensures that the SAC algorithm optimizes its policy across a continuum of increasingly challenging environments, enhancing its effectiveness and reliability in practical financial applications.

		GBM Stock Price		SABR Stock Price	
Method		Mean	Std	Mean	Std
SAC					
	PnL	-48.61	34.88	-46.11	38.39
	Cost	48.23	12.63	46.13	11.52
	Rewards	-98.94	65.25	-109.98	77.52
DDPG					
	PnL	-43.70	40.90	-51.43	35.43
	Cost	42.97	8.59	51.32	12.30
	Rewards	-119.16	67.13	-114.97	64.07
Delta Hedging					
	PnL	-96.29	33.55	-96.19	34.64
	Cost	96.44	28.21	96.39	28.09
	Rewards	-146.30	55.43	-149.17	57.09

Table 1: Performance comparison for SAC, DDPG, and Delta Hedging methods

Furthermore, it would be preferred to train the model with real-world data when applying the model in real markets. However, the training in this study requires large amounts of data (10^6 timesteps which is 20,000 trading days. This amount of data is typically not available for options on a single underlying asset. Mikkilä and Kanniainen (2021) used option data from 419 different underlying assets to train their DRL agent.

One way to modify the SAC algorithm to be more sample-efficient is multi-step learning. The SAC algorithm, which employs an off-policy, entropy-regularized, actor-critic approach, traditionally utilizes one-step returns for updating the critic. By extending this to n-step returns, one can more directly associate actions with their cumulative future rewards, thereby enhancing the algorithm's ability to learn effective policies from fewer interactions with the environment. This modification necessitates a recalibration of the critic's update mechanism to accommodate the temporal dimension introduced by n-step returns. Such an adjustment not only accelerates the convergence of the value function through the propagation of reward information but also optimizes the trade-off between bias and variance in the reward estimates. Effective implementation requires precise tuning of the n-step horizon and the discount factor to align with specific environmental dynamics, ensuring that the algorithm remains robust and converges efficiently.

If this modification still cannot train the DRL agent with real-world data available, an

alternative would be to pre-train the DRL agent using synthetic data and train the DRL agent again using real-world data. This approach benefits from the unlimited amount of synthetic data with less compromise on the transferability of the trained agent to real-world scenarios.

6 Conclusion

In this paper, the application of the Soft Actor-Critic (SAC) algorithm in financial hedging strategies has proved to be effective in various settings. The research demonstrated that SAC outperforms traditional hedging methods and other reinforcement learning algorithms, offering a more nuanced and robust approach to managing financial risks. The SAC's ability to balance risk and transaction costs effectively underscores its potential as a robust tool in financial risk management.

The implications of this study extend beyond academia, bridging the gap between sophisticated machine learning techniques and practical financial applications. For financial practitioners, the findings present a compelling case for integrating advanced AI methodologies into hedging strategies, potentially enhancing the efficacy and efficiency of risk management processes. However, while the research offers promising insights, it is not without limitations. The study's reliance on specific market models and data sets may affect the generalizability of the findings, suggesting a need for further exploration across diverse market conditions and financial instruments.

Future research could expand on the current work by testing the SAC algorithm in different financial environments and real-world data, with varying degrees of market volatility and trading constraints. Such studies could further refine the algorithm's capabilities, ensuring its applicability and robustness in real-world financial settings. Additionally, modifying the SAC algorithm such as utilizing multi-step learning, could yield even more transferable hedging strategies, pushing the boundaries of what is currently achievable in financial risk management.

In conclusion, this thesis not only contributes valuable insights into the fields of finance and machine learning but also sets the stage for future innovations in financial hedging

strategies. The integration of deep reinforcement learning, particularly through the Soft Actor-Critic algorithm, offers a promising avenue for developing more dynamic and responsive hedging mechanisms to help investors better manage their risks.

References

[1] R. Almgren and T. M. Li. Option hedging with smooth market impact. *Market Microstructure and Liquidity*, 2(1):1650002, 2016.

[2] P. Bank, H. M. Soner, and M. Voß. Hedging with temporary price impact. *Mathematics and Financial Economics*, 11(2):215–239, 2017.

[3] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[4] P. P. Boyle and T. Vorst. Option replication in discrete time with transaction costs. *The Journal of Finance*, 47(1):271–293, 1992.

[5] Hans Bühler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging, 2018.

[6] J. Cao, J. Chen, S. Farghadani, J. Hull, Z. Poulos, Z. Wang, and J. Yuan. Gamma and vega hedging using deep distributional reinforcement learning. *Frontiers in Artificial Intelligence*, 6, 2023.

[7] J. Cao, J. Chen, J. Hull, and Z. Poulos. Deep hedging of derivatives using reinforcement learning. *Journal of Financial Data Science*, 3(1):10–27, 2021.

[8] Jiayi Du, Muyang Jin, Petter N. Kolm, Gordon Ritter, Yixuan Wang, and Bofei Zhang. Deep reinforcement learning for option replication and hedging. *The Journal of Financial Data Science*, 2(4):44–57, Sep 2020.

[9] E. R. Grannan and G. H. Swindle. Minimizing transaction costs of option hedging strategies. *Mathematical Finance*, 6(4):341–364, 1996.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[11] Patrick Hagan, Deep Kumar, Andrew S. Lesniewski, and Dennis E. Woodward. Managing smile risk. *Wilmott*, pages 84–108, 2002.

[12] Igor Halperin. Qlbs: Q-learner in the black-scholes(-merton) worlds, 2017.

[13] P. N. Kolm and G. Ritter. Dynamic replication and hedging: a reinforcement learning approach. *Journal of Financial Data Science*, 1(1):159–171, 2019.

[14] H. E. Leland. Option pricing and replication with transactions costs. *The Journal of Finance*, 40(5):1283–1301, 1985.

[15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[16] L. Martellini. Efficient option replication in the presence of transactions costs. *Review of Derivatives Research*, 4(2):107–131, 2000.

[17] O. Mikkilä and J. Kanniainen. Empirical deep hedging. *Quantitative Finance*, 23(1):111–122, 2023.

[18] G. Ritter. Machine learning for trading, 2017. Available at SSRN 3015609.

[19] L. C. G. Rogers and S. Singh. The cost of illiquidity and its effects on hedging. *Mathematical Finance*, 20(4):597–615, 2010.

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[21] K. B. Toft. On the mean–variance tradeoff in option replication with transactions costs. *Journal of Financial and Quantitative Analysis*, 31(2):233–263, 1996.

[22] A. E. Whalley and P. Wilmott. An asymptotic analysis of an optimal hedging model for option pricing with transaction costs. *Mathematical Finance*, 7(3):307–324, 1997.