M2 SFA - ENSAE Paris

Machine Learning in finance: theoretical foundations : Project report

SE310

Dynamic Replication and Hedging: A Reinforcement Learning Approach Petter N. Kolm and Gordon Ritter

Author:
Jiaqi Xia

May 2024

Contents

I Introduction

The paper discusses the challenge of replicating and hedging option positions in finance, going back to the foundational work of Black, Scholes, and Merton. It highlights the discrepancy between theory, which requires discrete adjustments for hedging strategies, and practice, where there are trading costs and market frictions. Various studies have addressed transaction costs and market impact, with recent efforts integrating reinforcement learning for optimal hedging. The proposed method uses reinforcement learning techniques to obtain the best trade-off between trading costs and variance. Notably, it surpasses the limitations of finite-state-space methods. The method's versatility extends to handling arbitrary portfolios of derivative securities, providing automated hedging solutions for traders facing complex scenarios without directional views.

II The paper's approach

II.1 The Problem to Solve

Hedging is an investment strategy used to offset potential losses from adverse movements in the price of an asset. Here, an agent holds a long option position that cannot be traded. The agent has the possibility of trading the underlying with a risk-free asset for replication purposes, thus hedging his position. In an ideal scenario with no trading friction and continuous trading, a dynamic replicating portfolio could perfectly hedge the option position, resulting in zero overall portfolio variance. However, in this paper, we place in a real life context, where there are discretization constraints (discrete-time trade, discrete-trade amount), and market friction (transaction cost, market impact, etc.). The objective shifts to minimizing both variance and trading costs.

II.2 Main Idea of the Paper: Reinforcement Learning

This article chooses to maximize the mean-variance of the final wealth, $\max\left(E(W_t) - \frac{\kappa}{2}V(W_t)\right)$, using a reinforcement learning approach.

II.2.1 Reinforcement Learning

Reinforcement learning (RL) is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards. It is widely used in various fields such as robotics, gaming, finance, and healthcare.

II.2.2 Basic Components

RL typically involves the following basic components:

- **Agent**: The entity that learns to perform actions based on observations and receives rewards from the environment.
- **Environment**: The external system with which the agent interacts. It provides feedback to the agent in the form of rewards.
- **State**: A representation of the current situation of the environment. It is used by the agent to make decisions.
- **Action**: The set of possible moves or decisions that the agent can take.
- **Reward**: A scalar feedback signal from the environment indicating the success or failure of the agent's actions.
- **Q function**: A function that associates a score to each pair of (State, action). This determine deterministically ($action = \arg\max_{a \in A} Q(s, a)$) or stochastically ($P(\text{action} = a) = Q(s, a)$) It's updated during learning.

II.3 Setup in the Paper

- **Agent:** The agent can trade $D = 5$ times a day.
- **Training:** 1 day, over five batches with 15,000 episodes per batch
- **Action:** The action involves trading from $-L$ to L underlying shares.
- **Environment:** The underlying follows a Black-Scholes dynamic with $r = 0$, daily volatility of 0.01, initial price $S_0 = 100$, and option maturity of 10 days.
- **Quadratic Trading Cost:** The cost function is quadratic:

  $$\text{cost}(n) = \text{multiplier} \times \text{TickSize} \times \left(|n| + 0.01 \times n^2\right)$$

  where TickSize = 0.1, and n is the amount of underlying traded.
- **State:** The state comprises the current price of the underlying, the remaining time until the option maturity, and the number of underlying shares.

- **Decision Function:** The decision function employs an epsilon-greedy strategy, where actions are chosen either uniformly randomly or determined according to the Q neural network.
- **Reward:** We have the final objective

  $$\max\Big(E(W_t) - \frac{\kappa}{2}V(W_t)\Big)$$

  As

  $$W_T = W_0 + \sum_{i=1}^{T} dw_i$$

  So

  $$E[W_T] = W_0 + \sum_{i=1}^{T} E[dw_i]$$

  Then supposing that the dw_i are independent, we have also

  $$V[W_T] = \sum_{i=1}^{T} V[dw_i]$$

  Thus, we derive the reward for each step t:

  $$R_t = \mathrm{d}w_t + \frac{\kappa}{2} \cdot \mathrm{d}w_t^2$$

  So $\sum_{i=1}^{T} V[dw_i]R_t =$ objective where $\mathrm{d}w_t$ represents the total wealth variation at time t, and κ the risk aversion coefficient.

III Paper's Results

III.1 Agent's behavior

Obviously, in each case, it's just a question of a particular trajectory, which is not enough to draw any statistical conclusions about performance. They are mainly used to illustrate the behaviour of agents on the main line and to make qualitative interpretations.

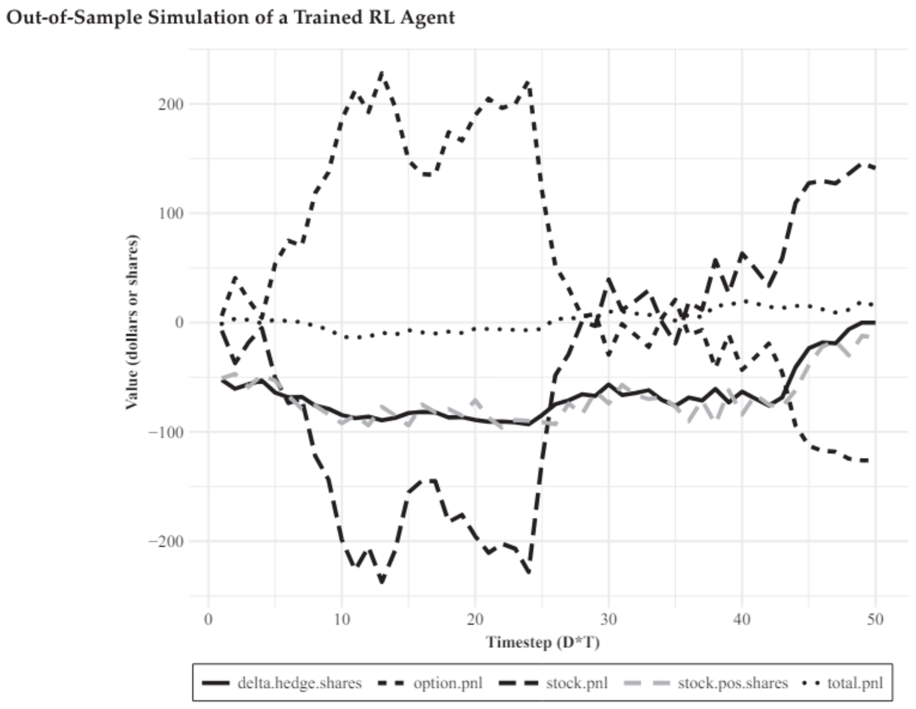


Figure 1: Out-of-Sample Simulation of a Trained RL Agent (case without trading cost)

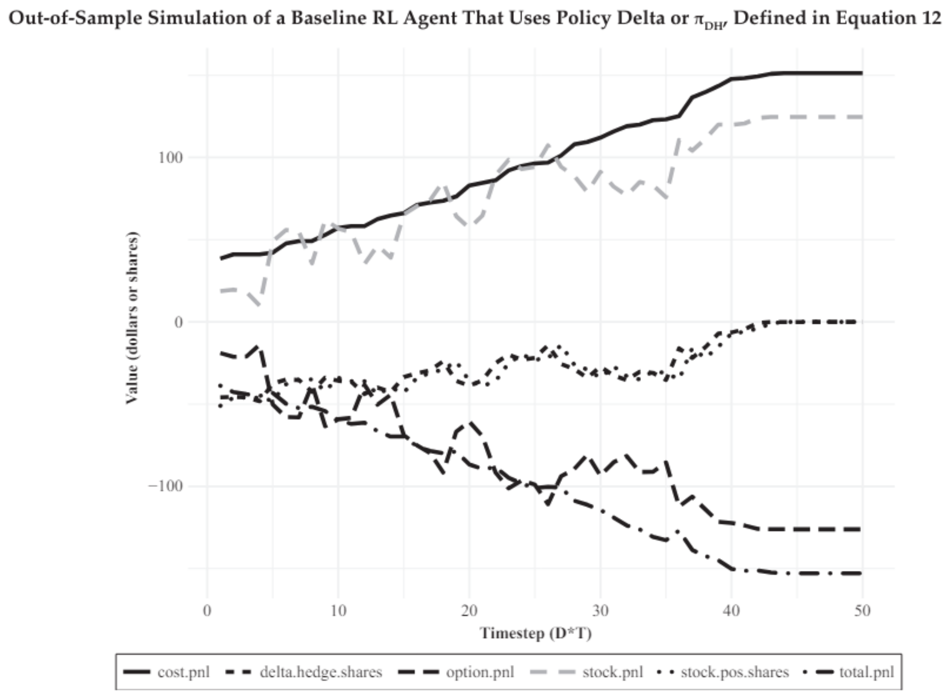


Figure 2: Out-of-Sample Simulation of a Baseline Agent using delta hedging strategy (case with trading cost)

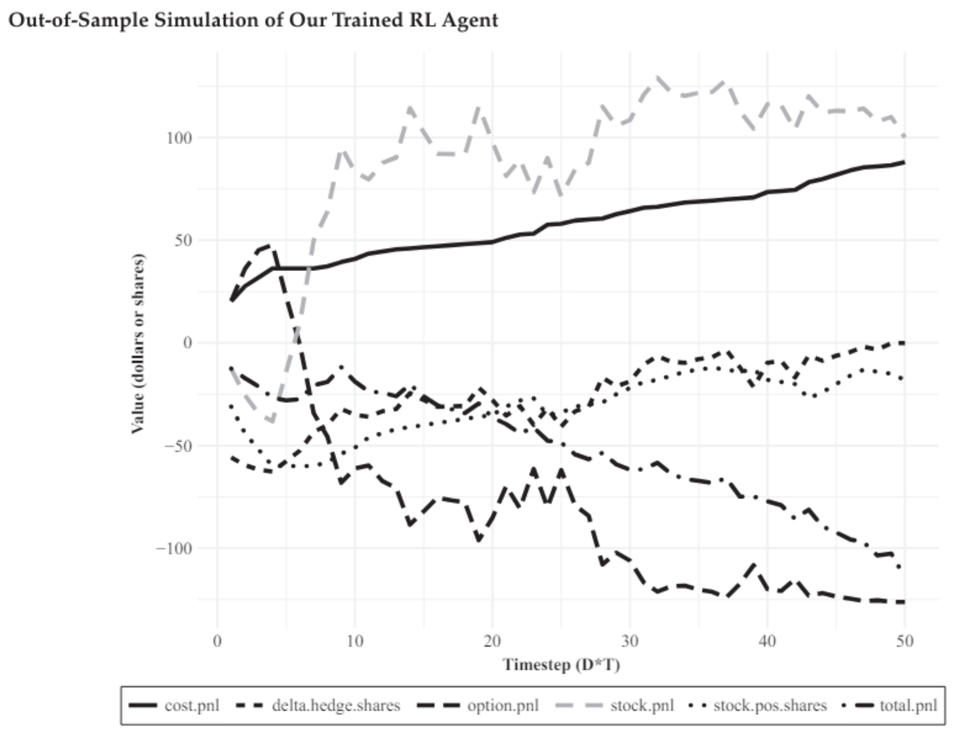


Figure 3: Out-of-Sample Simulation of a Trained RL Agent (case with trading cost)

In these illustrations, we observe that in the case without trading cost, the RL agent can offset the exposure to the underlying price movement and keep the total PnL close to 0. In the presence of trading costs, the RL agent has a smoother trading strategy, which means it avoids over-trading, even if it means not exactly following the underlying price. In the end, this gives him a better PnL than the baseline agent.

III.2 Performance Analysis

In order to do some statistically significant comparison between the RL agent and the baseline agent, they run a large number (N = 10000) out-of-sample simulations.

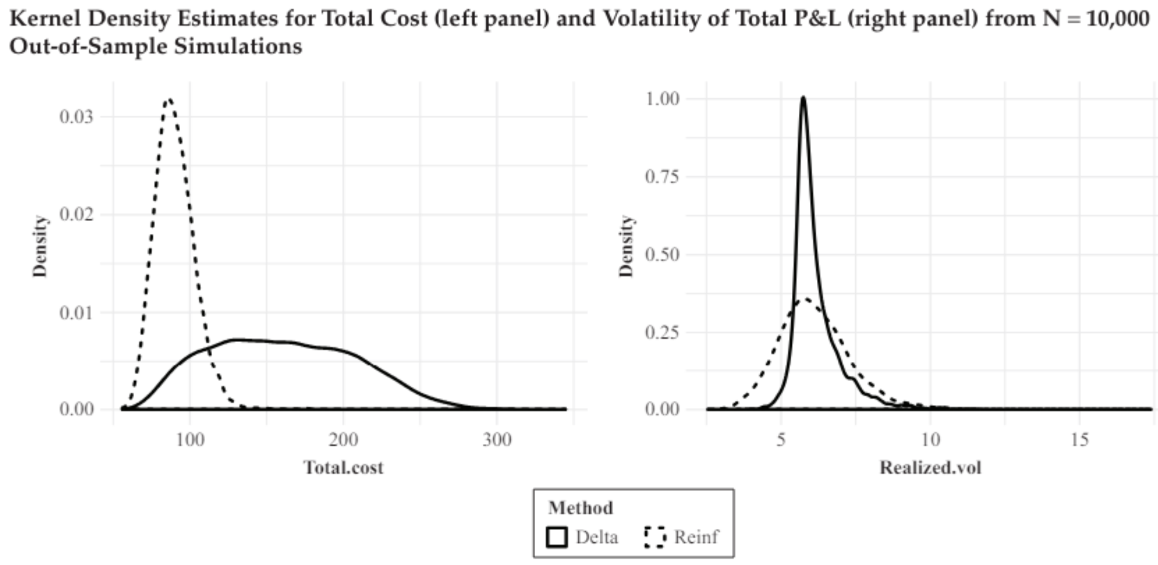


Figure 4: Kernel Density Estimates for Total Cost (left panel) and Volatility of Total PL (right panel) from N = 10,000 Out-of-Sample Simulations

We can conclude :
-The hedging cost is better (statistically significant) with the RL agent
-No statistical difference between 2 methods in terms of volatility

III.3 Other strength of Deep RL

- **Not limited on finite state space**: Unlike using Q table
- **No need to compute greeks**: The agent will learn these non-linear function by itself
- **Doesn't need a trading cost model**: The agent will learn it
- **No need for the existence of perfect dynamic replication**: Sometimes, perfect replication is impossible because of a lack of theoretical knowledge (in the case of an exotic option) or because of market imperfection. The agent would learn to balance variance and cost optimally.
- **can deal automatically with position-level constraints in a simple way.**:By simply removing the unwanted actions from the action space

IV Re-implementation

IV.1 Environment Setup

We tried to reproduce the results of the paper with the same environment setup for the case without trading costs.

For the case of trading cost, we supposed that the ticksize would be a quadratic function of the moment of the day.

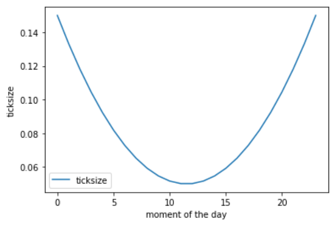


Figure 5: Ticksize(moment of the day)

Because of our limited computing power and memory, the training is done on only 1 batch of 100 episodes.

Improvement :

- **1**: We want the action space to cover the whole interval of $[-N_Option, N_Option]$. Especially at the beginning when the agent has 0 stocks, it could buy a lot of them in a short period of time.
- **2**: We want to minimize the action space to gain computing time
- **3**: we want to keep a good precision on little adjustments

So we added a transformation function on downstream of the Q network. Instead of trading $alpha_t$ stocks, the agent would trade $f(alpha_t)$ stocks. Its shape is designed to make the best trade-off between the 3 objectives.

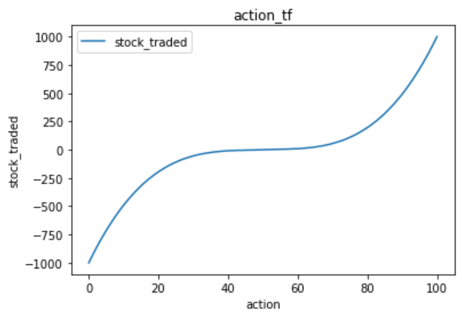
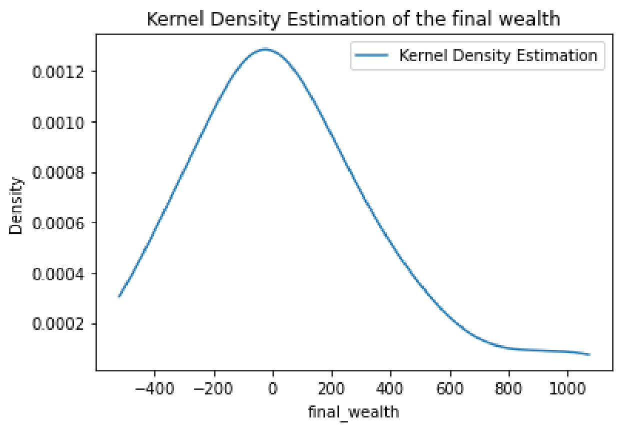


Figure 6:

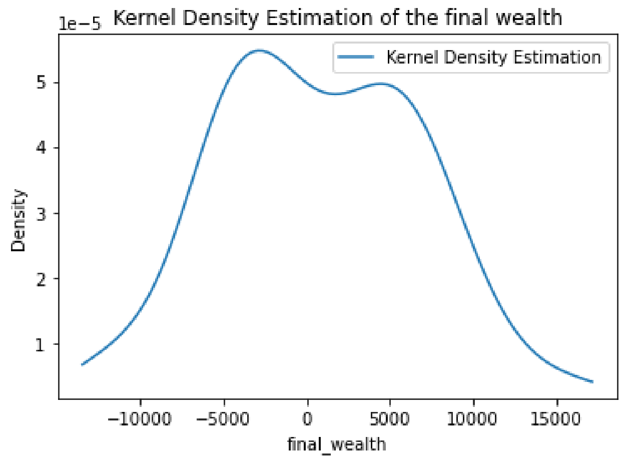
IV.2 My Results

IV.2.1 Without trading cost



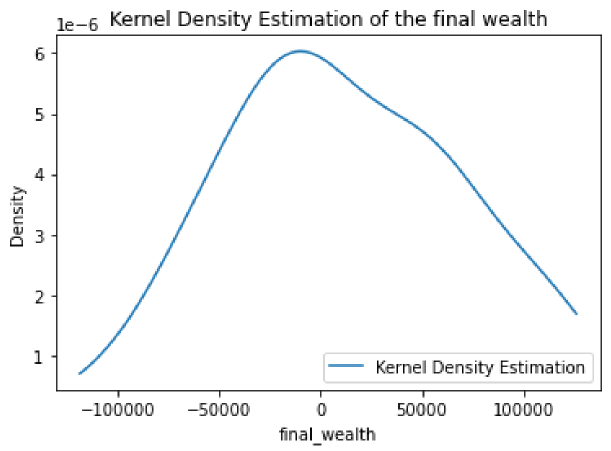
```
DH_mean = 40.365155333877965
DH_var = 102006.48170016974
DH mean var = -5059.95892967461
```

Figure 7: Kernel Density Estimates for final wealth, baseline agent, without trading cost



```
DQN_mean = 835.893470981369
DQN_var = 39375926.9191035
DQN_mean_var = -1967960.452484194
```

Figure 8: Kernel Density Estimates for final wealth, RL agent, trained on 20 episodes, without trading cost

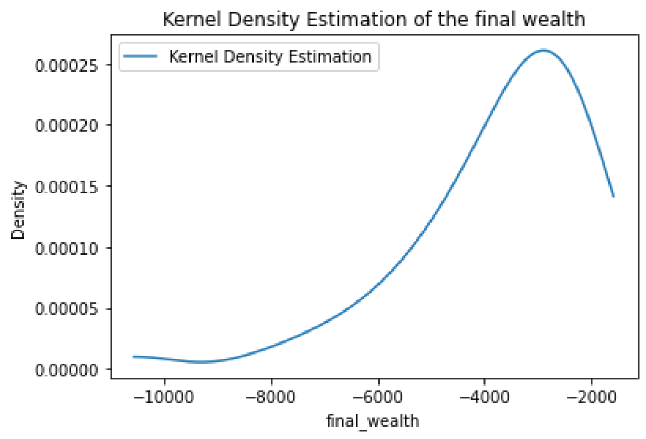


```
DQN_mean = 11790.164126916141
DQN_var = 3368770479.0516677
DQN_mean_var = -168426733.7884565
```

Figure 9: Kernel Density Estimates for final wealth, RL agent, trained on 100 episodes, without trading cost

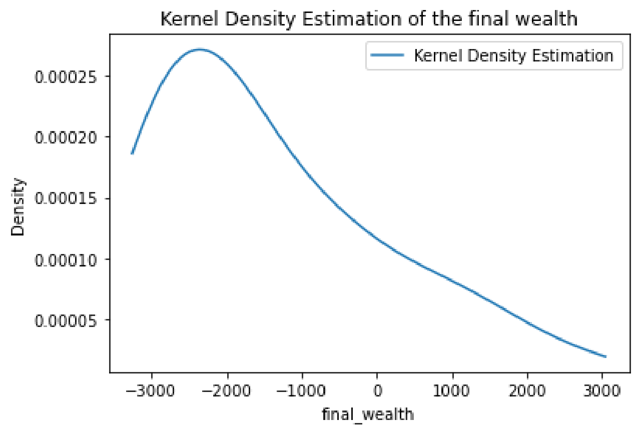
The RL agent outperforms the baseline agent in terms of expectation, but it's less good at minimizing the variance, as expected. These trends accentuate the longer we train. Finally, in terms of total objective, as defined in, II.2, our RL agent performs poorly compared to the baseline, as if its training makes it underestimate the $\frac{\kappa}{2} \cdot dw_t^2$ term

IV.2.2 With trading cost



```
DH_mean = -3720.0093955654306
DH_var = 3044776.5031250324
DH_mean_var = -155958.83455181707
```

Figure 10: Kernel Density Estimates for final wealth, baseline agent, with trading cost: multiplier = 5



```
DQN_mean = -1384.5664271573407
DQN_var = 2457055.6517155785
DQN_mean_var = -124237.34901293628
```

Figure 11: Kernel Density Estimates for final wealth, RL agent, trained on 100 episodes, with trading cost: multiplier = 5

Our RL agent performs way better than the baseline agent in both mean and variance. This is predictable because the baseline delta hedging strategy doesn't take into consideration trading cost. We can assume that the agent's occasional decision errors due to insufficient/inadequate training are more than compensated for by taking trading costs into account.

All agents are tested on 50 out-of-sample episodes

IV.3 Limits and Further Ideas

We can question the validity of the hypotheses

IV.3.1 Independent wealth increment?

As explained in II.3, the reward function is derived from the final objective by supposing that all the wealth increments are decorrelated. But by fitting these increments to an AR(1) model, for example, we observe for many trajectories, the auto-regressive coefficient is non-zero, which means there are temporal dependencies.

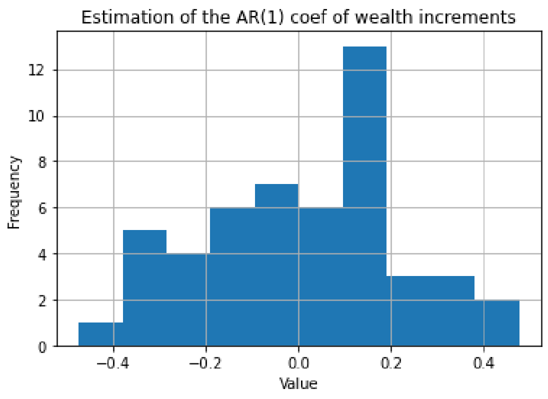


Figure 12:

In the case of dependencies,

$$V[W_T] >= \sum_{i=1}^{T} V[dw_i]$$

so if we want equality, we should use a $\kappa' > \kappa$ in the R_t. This could maybe explain why using the same κ in R_t and in the final objective make the training process to neglect variance minimization

An idea of the solution: Simply use a $\kappa' > \kappa$ But $\kappa' =$? Or fit a model to estimate temporal dependency while training, use the model to update the reward function during training, and use further wealth increment to update the model.

IV.3.2 It still needs a "good" derivative pricer

It's said that with this method, "any option that can be priced can also be hedged". But what is a "good" price? For example, in our case, the BS formula used to price the European option supposes continuous hedging in a frictionless market. So it tends to under-price the option. I would be better if the pricer itself integrated estimated future hedging cost into the option price.

An idea of the solution : Train alternatively the Q network and a pricing network. The pricing network for a time t would have a target C such as selling the options at t at price C and stopping trading until T would lead to the same final wealth on average than keeping the option and hedging it until T.

V Conclusion

Q-learning is widely applied in various domains, including robotics, gaming, and finance, for its simplicity, efficiency, and ability to learn optimal policies in complex environments. This paper illustrates the effectiveness of Deep Q-learning in hedging financial derivatives under real-world constraints. While previous studies like Leland (1985) and Almgren and Chriss (1999) addressed discrete hedging and market impact, these methods, in order to obtain analytical results, use simple models that could not reflect reality. RL methods offer greater flexibility by being less dependent on specific models. They can be applied to various derivatives without assuming perfect replication. We were able to reproduce approximately the method described in the paper. our trained agent adopts similar behaviors and improves as training progresses. Even with limited computing resources, our agent successfully exceed the delta hedging performance in the case with trading cost. However, the method's assumptions have limitations, and our tests focused solely on simple European options hedging. Although the model avoids the curse of dimensionality by not restricting to finite state space, the action space remains finite. This raises questions regarding its applicability to more complex scenarios like hedging basket options involving trading multiple underlying simultaneously.