Assessing reinforcement delta hedging

Haodong Zhang[a], Hirbod Assa[b,*], Chris Kenyon[c†]

September 6, 2021

[a] PING AN Insurance, China; [b] Kent Business School, UK; [c]MUFG Securities EMEA plc, UK

Abstract

Usual option pricing is based on replication assuming complete markets. Complete markets means that simple hedging strategies, like delta hedging, work exactly but real markets are not complete. This has motivated research on reinforcement learning to develop pricing for incomplete markets that are highly complex to deal with otherwise. However, the literature has studied approximations of reinforcement learning giving the agents hints, e.g. approximated payoffs before the actual final payoff, or by using a neural network for each timestep, increasing complexity.

Here, we investigate if a single agent can learn hedging without hints, i.e. by only knowing the final pay-off, using reinforcement learning. We consider the simplest possible option setup: a vanilla European equity option in a pure Black-Scholes world. We use an agent-critic-based method, Deep Deterministic Policy Gradient (DDPG), and show that even in this simplest case, DDPG alone is unable to properly hedge. However, by employing ensemble methods, i.e. average and median, we can significantly improve accuracy and stability. This emphasises the need for customization in application of deep learning in hedging rather than being purely driven by data and hyper-parameters.

1 Introduction

Neural networks (NN) have been used as a non-parametric method since the early 1990s in finance, particularly option pricing and hedging, Ruf and Wang (2020). However, there have been very few papers using deep reinforcement learning (hence RL)

*Corresponding Author, email: h.assa@kent.ac.uk

†This paper is a personal view and does not represent the views of MUFG Securities EMEA plc ("MUSE"). This paper is not advice. Certain information contained in this presentation has been obtained or derived from third party sources and such information is believed to be correct and reliable but has not been independently verified. Furthermore the information may not be current due to, among other things, changes in the financial markets or economic environment. No obligation is accepted to update any such information contained in this presentation. MUSE shall not be liable in any manner whatsoever for any consequences or loss (including but not limited to any direct, indirect or consequential loss, loss of profits and damages) arising from any reliance on or usage of this presentation and accepts no legal responsibility to any party who directly or indirectly receives this material.

for hedging. This is despite the fact that deep RL, has proven to be a very powerful tool in solving action-involved problems with delayed rewards Lillicrap et al. (2015) which is also a description of hedging problems. Buehler et al. (2019) use a repeated neural network approach, i.e. a different NN for each hedging decision, so one NN for each day of the 30-day maturity. Halperin (2017) uses a Q-Learner method in the Black-Scholes-Merton framework with payoff at the horizon (i.e., time T), for a general option on an equity. For hedging the author used the underlying and the risk free asset in a discrete state and discrete time setup, 24 steps for one year maturity. Similar to Buehler et al. (2019), a separate set of basis functions was fitted at each timestep in their "*Fitted Q Iteration*" for the optimal action and action-value functions. Kolm and Ritter (2019) and Du et al. (2020) both employ similar approaches where the objective function is the quadratic utility on the final wealth as the summation of wealth at all times. Cao et al. (2021) and Giurca and Borovkova (2021) using DDPG method expand the range of objective functions by considering the total future hedging cost during the life of the instrument, that is an idea that has been proposed by Tamar et al. (2016).

All these papers either consider agents that learn hedging in each timestep, eventually considering multiple agents that make the hedging strategy depending on the number of timesteps as a hyper parameter; or, consider penalty functions that rely on more information than only the final pay-off. While the first approach can significantly increase the over-fitting problem and is computationally costly, the second approach seems to be unrealistic.

In this paper, to study these issues we focus our attention on delta hedging as one of the simplest possible hedging strategies that can serve as a benchmark problem. More specifically, this paper uses Deep Deterministic Policy Gradient (DDPG), introduced in Lillicrap et al. (2015), to train a textitsingle agent to hedge a European call option in a complete market (Black-Scholes). This is in contrary to the earlier research like Buehler et al. (2019) or Halperin (2017). In addition, unlike Kolm and Ritter (2019) and Du et al. (2020), Cao et al. (2021) and Giurca and Borovkova (2021), we consider a penalty function that only makes an assessment of the hedging strategy at the *final trading step*. We observe that a DDPG method alone is very unlikely to find a proper hedging strategy even by increasing the number of the training episodes or hyper-parameter tuning. Therefore, we introduced a general diagnosis method which helped us to better understand the ways to overcome this limitation. From the diagnosis we could see the benefit of using ensembling methods and after employing some standard methods, i.e. average and median, we observed that the accuracy and the stability of the hedging strategies improved even with few training episodes ($<$ 10,000) and few models ensembled ($<$ 50). In the following table we show how our paper can be distinguished from the existing literature.

The paper is designed as follows: after a short introduction to delta hedging in a Black-Scholes model and RL in Sections 2 and 3 respectively, then in Section 4 we train the DDPG model to learn delta hedging. In Section 5 we report the results and see that the DDPG fits poorly. In Section 6 we introduce a diagnosis method where we find out and observe how the ensembling can improve the results. Section 7 concludes.

Buehler et al. (2019) Halperin (2017)	multiple agents	final step pay-off
Kolm and Ritter (2019) Du et al. (2020) Cao et al. (2021) Giurca and Borovkova (2021)	single agent	multi-step pay-offs
ZAK	single agent	final step pay-off

Table 1: The literature on RL and hedging.

2 Black-Scholes model and hedging

Let us consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, consisting of a state space, a sigma algebra on the state space, and a probability measure on the sigma algebra. As usual a random variable X is a measurable function defined on Ω, whose cumulative distribution function is denoted by F_X and its probability density function (if it exists) by f_X. The expectation is denoted by E and for an integrable random variable X is denoted by $E(X)$. In this paper for simplicity we denote the conditional probability $E[X|A]$, for a measurable set A by $E_A[X]$. Finally $Var(X)$ and $std(X)$ denote the variance and the standard deviations of X and are defined as $Var(X) = E\left[(X - E(X))^2\right]$ and $std(X) = \sqrt{Var(X)}$.

2.1 Black-Scholes Model

In this paper we follow standard text book discussions on Black-Scholes model and delta hedging (see e.g., Björk (2009)). Consider the Black-Scholes model that consists of a risk free asset with interest rate r and an asset whose value is modeled by a Geometric Brownian Motion S_t given by:

$$dS_t = \mu S_t dt + \sigma S_t dW_t, \tag{1}$$

where $\{W_t\}_{0 \le t < \infty}$ is a standard Brownian motion, $\sigma > 0$ is the volatility and μ is the drift. For the underlying asset (or stock), S_t indicates the stock price at time t; dS_t is the change in a small time interval dt at time t, dS_t/S_t means the change per unit value of the stock, which is the return of the stock at time t.

In order to find the price of the derivatives by expectation we need to find the so-called risk neutral probability measure $\mathbb{Q}$ under which the discounted asset is a martingale:

$$E^{\mathbb{Q}}[e^{-rt} S_t | \mathcal{F}_s] = e^{-rs} S_s, \forall s < t,$$

where $\mathcal{F}_s$ is the sigma algebra generated by $\{W_\tau\}_{0 \le \tau \le s}$ i.e. all information until time s. For the Black-Scholes model this is equivalent to $\mu = r$. So below we take $\mu = r$. We can write S_T as

$$S_T = S_t \exp\left\{ \left(r - \frac{1}{2}\sigma^2 \right)(T - t) + \sigma(W_T - W_t) \right\},$$

With the Black-Scholes Model, we are able to simulate stock dynamics to any horizon.

2.2 Call option pricing

Here we consider a European call option on the stock price S_t with strike K, risk-free interest rate r, volatility σ and expiry T.

We know that the disjoint increments of the Brownian motion are independent and $W_T - W_t$ has a normal distribution of $N(0, T-t)$. This implies that $(W_T - W_t)/\sqrt{T-t} =: Z \sim N(0,1)$ and

$$S_T = S_t \text{exp}\left\{\left(r - \frac{1}{2}\sigma^2\right)(T-t) + \sigma Z\sqrt{T-t}\right\},$$

where S_t and Z are independent. A call option with strike price $K > 0$ and expiry date has the following pay-off:

$$C = \max(S_T - K, 0).$$

The call option price is the expected pay-off under the risk neutral probability measure and for the Black-Scholes model this is

$$C_t = S_t \Phi(d_+^t) - Ke^{-r(T-t)}\Phi(d_-^t) \tag{2}$$

where,Φ is the CDF of the normal distribution $N(0,1)$ and

$$d_\pm^t := \left[\log(S_t/K) + \left(r \pm \frac{1}{2}\sigma^2\right)(T-t)\right] \Big/ \sigma\sqrt{(T-t)}.$$

2.3 Delta hedging and replication

For a general derivative with pay-off $X(S_T)$, $\partial X/\partial S$, is referred to as the delta, denoted by Δ, which is the ratio between the value of an option contract and the corresponding movement of the underlying stock's price. For example, for a call option contract, if Δ is equal to 0.5, if the underlying stock price increases by 1 unite per share, the call option's value will increase by 0.5 unite per share. In the Black-Scholes model the delta of a European call option C is:

$$\Delta = \frac{\partial C}{\partial S} = \frac{\partial}{\partial S}\left[S\Phi(d_+) - Ke^{-rT}\Phi(d_-)\right] = \Phi(d_+).$$

Hence $\Delta \in (0,1)$.

The Black-Scholes model is a complete model. This means that all derivatives can be uniquely replicated by a dynamic rebalancing of a portfolio of the underlying and the risk free asset. For a call option C the replicating portfolio is:

$$(a_t, b_t) = (\Phi(d_+^t), -Ke^{-rT}\Phi(d_-^t)),$$

where (a_t, b_t) is the number of shares (S) and the risk free position (bank account) respectively.

2.4 Simulated data

Using the Black-Scholes model with a fixed volatility, we are able to generate as many future stock price simulations as we want. Here we set up the starting price S_0 of the stock at 100, interest rate $r = 0.03$ and volatility $\sigma = 0.2$. We divide a trading year into 100 intervals, or hedging steps. As increasing the number of intervals would increase the difficulty of the learning process (the bigger the number of trading, the harder for the model to 'track back'), so limited by computing power, 100 intervals is a suitable example value.

Recall the formula of the Black-Scholes model for the stock price:

$$S_t = S_0 \text{exp} \left\{ \left(r - \frac{1}{2}\sigma^2 \right) t + \sigma W_t \right\},$$

With $\Delta t = 0.01$ we generate 100 samples $B_i \sim N(0, \Delta t)$ which follow the standard normal distribution, then set $W_t = W_{0.01i} = \text{sum}(B_i) \cdot \sqrt{\Delta t}$. Hence, all the stock price simulations follow the Black-Scholes model and have the same interest rate and volatility. Figure 1 shows an example of ten random simulations.

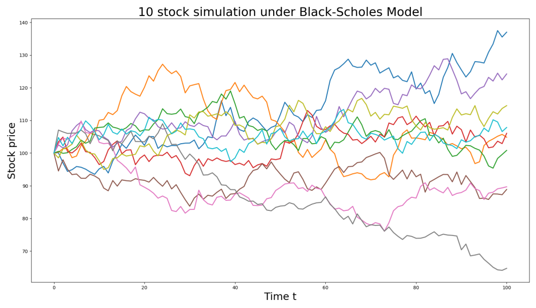


Figure 1: 10 randomly generated stock price paths.

In this paper we consider at the money call option with strike equal to $K = S_0 = 100$.

3 Deep Reinforcement Learning

Here we discuss RL and the specific methods that we employ in this paper. We follow the discussions in Sutton and Sutton and Barto (2018) and Goodfellow et al. (2016).

3.1 The brief review of reinforcement learning

The idea of RL find its root in solving a dynamic programming problem. So let us begin by a brief review of the dynamic programming. First let us introduce the following notations:

$s \in \mathcal{S}$, States.

$a \in \mathcal{A}$, Actions.

$r : \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, Rewards.

s_t, a_t, State,action,and reward at time t and $r_t = r(a_t, s_t)$.

N, Number of episodes.

γ, Discount factor.

α, The learning step.

G_t, Return; or discounted future reward; $G_t = \sum_{k=0}^{N-n} \gamma^k r_{t+k}$.

$P(s'|s, a)$, Transition probability.

$\pi(a|s)$, Stochastic policy.

$\pi_\theta(s)$, is a policy parameterized by θ.

$\pi(s)$, Deterministic policy.

μ, is the stationary state distribution.

μ_θ, is the parameterized stationary state distribution.

$V_w(s) = \mathbb{E}[G_t|s]$, is a value function parameterized by$w$.

$V^\pi(s), = \mathbb{E}_{a\sim\pi}[G_t|s]$.

$Q_w(s, a) = \mathbb{E}[G_t|s, a]$, Action value function parameterized by w.

$Q^\pi(s, a) = \mathbb{E}_{a\sim\pi}[G_t|s, a]$.

$J_\theta = \sum_{s\in\mathcal{S}} \mu_\theta(s)V^{\pi_\theta}(s)$, Reward function.

Consider an agent in state $s \in \mathcal{S}$ in time zero that wants to find a policy π that maximizes its lifetime reward:

$$V(s) = \max_\pi E_s \left(\sum_{t=0}^{\infty} \gamma^t r\left(\pi\left(a_t|s_t\right), s_t\right) \right).$$

Usually a solution for this problem is defined recursively where the agents tries to find a policy function $\pi : \mathcal{A}|\mathcal{S} \to \mathcal{A}$, and the optimal solution is given as:

$$V^\pi(s) = E_s \left(\sum_{t=0}^{\infty} \gamma^t r\left(\pi\left(a_t|s_t\right), s_t\right) \right).$$

There are many possible generalizations; for instance, one can consider a stochastic reward and also a stochastic policy. In that case we can specify if the expectation is conditional to which random variable. While the main aim of the agent is to maximize the lifetime reward, their target is to find a policy function that can make this possible. In that respect, the agent can either find the value function or the policy function, as finding either can result in finding the other one. There are very few dynamic programming problems with a closed form solution. Usually, a solution to a dynamic programming problem is approximated by using a numerical method. For instance, the value iteration method can approximate the value function by starting from an initial value function and approximate the value function within the following iteration:

$$V_{n+1}(s) = \max_a \left\{ r(a,s) + E_s\left(V_n(s')\right)\right\}.$$

Now consider a situation where we do not consider any model and want to study a model free dynamic programming problem. This essentially means we do not consider a specific transition probability. RL is an approach to solve these type of non-parametric/model-free problems. More precisely, in that case a RL setup tries to find a policy or a value function by making observation from the environment. In this way one can observe the reward and the transition by interacting with the environment.

A typical example is the video games, where an agent would receive information about the reward from the different actions they take; e.g., in a car driving game one would realize if the car crashes another car or hit a wall by watching the game screen. Like a human, the agent learns to achieve successful strategies that lead to the greatest long-term rewards. This paradigm of learning by trial-and-error, solely from rewards, is known as RL. Also like a human, the agent construct and learn their knowledge directly from raw inputs, such as market prices and its fluctuation. This knowledge can be summarised using deep neural networks.

RL is an approach to find the value or the policy function without considering a specific transition model or payoff. There are three main approaches to RL:

- Value based: In this method we approximate the value function J through updating a deep neural network V_W, where W are the weights of the neural network. Most value-based methods focus on updating the so-called Q-function $Q_w(.)$ which approximates J and is effectively V_W. This is done by a (deep) neural network for $Q_w(.)$ rather than J_W. More precisely the method is based on the following update for the next state s':

  $$w \leftarrow w - \alpha \left(Q_w(s,a) - r(s,a) - \gamma \max_{a'} Q_w(s',a') \right).$$

  This is known as the deep Q-learning (DQN) algorithm.

- Policy based: In this method, we approximate the policy function π by a deep neural network network π_θ, where θ are the weights of the network. The major idea in this method is based on the stochastic gradient policy theorem which states:

  $$\nabla J_\theta \propto \sum_{s\in\mathcal{S}} \mu_\theta(s) \sum_a \nabla \pi_\theta(a|s) Q^{\pi_\theta}(a,s).$$

  As a result, one can derive the following updates for the policy function parameter θ in step n:

  $$\theta \leftarrow \theta + \alpha\gamma^n G_n \nabla \log \pi_\theta(a_n|s_n).$$

- Combined value and policy based, Actor-Critic: In this method we update the value network J_w and policy π_θ at the same time. In any update the actor will move according to the policy function, and next, the critic will be using the value function to critic the actor move. The critic is essentially based on a so-called baseline with which the critic compares the value of the value after the actor's move. The baseline in most algorithms is an update of the value function, which implies not only the policy but also the value function need to be updated. An algorithm can be developed where the updates are done in step n according to the following:

$$\begin{aligned} w &\leftarrow w + \alpha_1 \gamma^n (G_n - V_w(s_n)) \nabla V_w(s_n), \text{ Critic}, \\ \theta &\leftarrow \theta + \alpha_2 \gamma^n (G_n - V_w(s_n)) \nabla \pi_\theta(s_n), \text{ Actor}. \end{aligned}$$

We employ the actor-critic method where the policy is assumed deterministic. In our hedging application, the state is Markov so we can make the decision only based on the current state. This is the idea of the deterministic gradient policy theorem by Silver et al. (2014) which states a version of the chain rule:

$$\nabla J_\theta \propto E_{s \sim \mu_\theta} [\nabla_\theta \pi(s) \nabla Q^{\pi_\theta}(s, a)|_{a=\pi_\theta(s)}].$$

In the update we need to use the gradient of both the policy and the value function which means we need two neural networks: actor and critic. This is called the Deterministic Policy Gradient algorithm (DPG).

3.2 Deep Deterministic Policy Gradient (DDPG)

DDPG is one of the most popular reinforcement algorithms introduced in Lillicrap, et al. (2015) that can combine the DPG and the DQN. Prior to DDPG attempts to combine DPG and DQN were unstable. DDPG introduced exploration (caching), i.e. it did not update on every step: this was sufficient to create stability. However as we will see, this is not enough to solve the hedging problem we consider.

The policy function $\pi(.|s)$ is usually modeled as a probability distribution over actions $\mathcal{A}$ given the current state, thus it is stochastic. In the deterministic policy gradient (DPG) the policy is given as $a = \pi(s)$. We can consider the deterministic policy as a special case of the stochastic one, when the probability distribution contains only one extreme non-zero value over one action.

Comparing with the deterministic policy, the stochastic policy requires more samples as it integrates the data over the whole state and action space. On the other hand, as transitional Q-learning has its limitation, where all the Q value will stored in a table, where will save all Q value for every possible actions for each state. However most of the problem now have huge possible states and actions available, storing every Q value becomes impossible. So we use Neural Network to replace the Q value table to decides and gives Q value to actions. However, unless there is sufficient information it is difficult to create a good deterministic policy. So, in a deterministic policy algorithm, an exploration step must be added. DDPG actually considers an exploration step that is given as follows:

$$a = \pi(s) + \mathcal{R},$$

where $\mathcal{R}$ is a random variable that can be given by a random process. Here we use an Ornstein–Uhlenbeck process.

In DDPG the advantages of actor-critic and Deep Q-learning are both present. On the Policy Gradient part, an action prediction network will output an action; the action target network helps value-based network to update State value. Where the state target network use the action from Action target network to update state value where state prediction network use action from action prediction network.

4 Implementing DDPG for a call option in the BS model

The purpose of this project is to assess deep RL to re-balance the hedging portfolio by trading the stock market. We will use DDPG, where we need to specify the model components including the states and the actions and then we will give the first results.

The code consists of two parts: the training agent with neural networks, and the environment. The training agent we used was the DDPG agent from Keras-rl, a deep RL project base on Keras. We first give details on the working environment then the setup of the DDPG agent and neural network.

4.1 Environment

We want our model to hedge based on the current information at every step, as in delta hedging. From the delta hedging formula we know that Δ depends on the price of the stock S_t and the time to maturity $T - t$, given that option parameter (strike) is fixed. We need to decide how to re-balance between the bank account and stock account. So, the inputs at time t are:

$$S_t, (T - t), X_t, Y_t,$$

which indicates the current stock price, the time left until expiry, the quantity of money we hold in the bank account, and the quantity of money we hold as stock, respectively. Note that, in delta hedging, borrowing from the bank is allowed, and the stock can be shorted. The hedging portfolio must be self-financing so net value cannot be created or destroyed by changing the position. The position net value can only change from interest on the bank account or movement of the stock price. Thus the actual state we need to consider is

$$S_t, (T - t), Z_t,$$

where $Z_t = X_t + Y_t$. The action is how much stock to hold.

We also need a reward function, so each time it finishes an episode, it will receive a "score" based on its performance during this episode. An episode is a hedge-to-maturity starting from time zero and ending at the option maturity.

Now, we know what is required for building an environment where our model could learn and play the hedging game by:

- Updating states at each time point for the model to observe and make an action.
- A reward function to tell the model how successful it is.

The whole environment works like this: at time t, the model observes the current state $s_t = (S_t, T - t, X_t, Y_t)$ and inputs this state to the agent; the agent will then decide to make an action a_t to decide how much money we should put into stock market to buy the underlying stock. Because the portfolio is self financing at each time point, which means the total amount of money we hold remains the same, the money in bank becomes $a'_t = X_t + Y_t - a_t$. Then after a time interval pass, the current time changes from t to $t+1$. We will observe a new state at time $t+1$, $(S_{t+1}, T-(t+1), X_{t+1}, Y_{t+1})$. Because of the stock price changes through time and we gain/pay interest r to the bank depending on our bank account. The new state X_{t+1} and Y_{t+1} are related as follows:

$$X_{t+1} = (1 + r)a'_t$$

$$Y_{t+1} = a_t \cdot \frac{S_{t+1}}{S_t}$$

4.2 Agent setting

We have introduced the DDPG algorithms in the previous section, which contained two parts, actor and critic. The actor neural network will receive the inputs (states) from the environment and output an action, and the critic neural network will input the result of the action and output a score for the action.

Actor hidden layers: $64 \times 64 \times 64 \times 64 := 64^4$

- fully connected
- activation: relu
- output: sigmoid

Critic hidden layers: $64 \times 64 \times 64 \times 64 := 64^4$

- fully connected
- activation: relu
- output: linear

We aim compress the input state in the range of -1 to 1, where activation functions work more efficiently. However, at the same time, we aim to maintain the relationship between different inputs. For example, the stock price S_t and X_t share the same units (money), so we shall use the same scale to compress them. So the compression method we used for stock price, amount in bank and stock is divided by the maximum stock price S_{max} of all the data we use for training and testing. We still use the same scale to compress the input, otherwise the neural networks' weights would be hard to compare and test for further use.

It is important to mention that unlike Buehler et al. (2019) and Halperin (2017) in this paper we do not fit different NN at each timestep, but consider a single agent for the whole process.

4.3 Reward function and the final pay-off

A typical challenge in machine learning and deep RL is how to find a good reward function. Sometimes even a small change can give a totally different result. We want our reward function to determine the difference between the amount of money we need to pay $(S_T - K)_+$ and the total amount of money $X_T + Y_T$ we hold at the expiry time T. The smaller the difference, the better the result, which means that even if we hold much more than what we need to pay at the expiry time, the result is still not good because we are hedging not investing. Therefore, our reward function is:

$$r(s_t, a_t) = \begin{cases} -|(S_T - K)_+ - (X_T + Y_T)| & t = T \\ 0 & o.w. \end{cases}.$$

This means that unless we perfect hedge the option, we will always get a negative reward, and the bigger the reward, the better the result.

As one can see this penalty function is clearly based only on the final pay-offs. For instance, Kolm and Ritter (2019) and Du et al. (2020) used an objective like the following

$$\max_{\text{strategies}} \sum_{t=0}^{T} \left(E\left[\delta w_t\right] - \frac{\kappa}{2} Var\left[\delta w_t\right] \right),$$

for a positive number κ where w_t denoted the agent's wealth at time t. On the other hand, Cao et al. (2021) and Giurca and Borovkova (2021) use the cost instead. More precisely, their objective is to minimize $Y(0)$ where

$$Y(t) = E\left(C_t\right) + c\sqrt{E\left(C_t^2\right) - E\left(C_t\right)^2}.$$

Here c is a positive number and C_t is the hedging cost from time t and onward.

These objective function are to some extent unrealistic in practice as they account for more information than only the final pay-off.

In Figure 2 we have a schematic view of the algorithm:

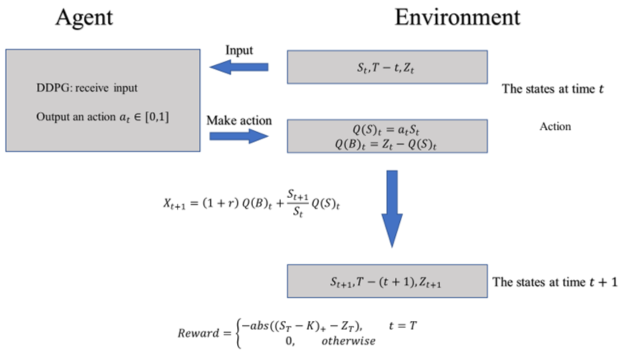

Agent
Environment
Input
DDPG: receive input
Output an action $a_t \in [0,1]$
Make action
$S_t, T - t, Z_t$
The states at time t
$Q(S)_t = a_t S_t$
$Q(B)_t = Z_t - Q(S)_t$
Action
$X_{t+1} = (1+r)\,Q(B)_t + \frac{S_{t+1}}{S_t} Q(S)_t$
$S_{t+1}, T - (t+1), Z_{t+1}$
The states at time $t+1$
$Reward = \begin{cases} -abs((S_T - K)_+ - Z_T), & t = T \\ 0, & otherwise \end{cases}$

Figure 2: DDPG for hedging.

5 Stability and result v.s. Delta hedging

In the complete market implied from the Black-Scholes model, delta hedging gives perfect replication of derivatives. In the following discussions, in this section and Section 6, we make comparison between the different hedging strategies we introduce and the delta hedging. As the delta hedging is a function of the asset value and the time to maturity we also have chosen to present the hedging strategies we study as a function of asset value and time to maturity. That is why for the asset value S_t we consider the action of the agent at the state $s_t = (S_t, T-t, Z_t^{\Delta})$, where Z_t^{Δ} is the delta hedging portfolio value associated with the asset value S_t.

5.1 First observation: training and hyperparameter tuning

We trained the model with 12,000, 20,000, 30,000 and 45,000 episodes respectively and used the saved weights to test on 500 out of sample data to check the result (see Figure 3).

As well as varying the training length, we tested different NN configurations

- 16^3
- 64^3
- 64^4, no improvement with additional layers, so this is used in all results presented below
- 64^5

We can see that, after 20,000 episodes, the result stops to improve and the outcome is poor in two ways:

- The average error is high.
- The stability of the model is not satisfactory, i.e. results vary considerably between different out of sample simulation runs.

In our model, the worst performance shows an error of over -12, and most results are in the range of -4 to 0. This indicates that our model could not prove a stable outcome. In some situations, it could perform well but many times would perform very badly.

We shall now diagnose why the model performs badly and use this to improve performance.

5.2 Errors of the fitted models

In the next step we measured the error of the algorithm for 100 fitted models. More precisely, we have considered 100 DDPG models each trained for 10,000 episodes. We measured the error in two different ways. If we denote the stock value of the delta hedging for the stock value S_t by $\Delta_{BS}(S_t)$ and the stock value of the hedging proposed by the DDPG method by $\Delta_{DDPG}(S_t)$ then we introduce the error and the absolute error for the stock value S_t as follows:

$$\begin{aligned} e(S_t) &= \Delta_{BS}(S_t) - \Delta_{DDPG}(S_t), \\ e^a(S_t) &= \left|\Delta_{BS}(S_t) - \Delta_{DDPG}(S_t)\right|. \end{aligned}$$

If there is no confusion about time we can drop the time subscript. Note that in a hedging strategy we better to consider two errors: the (real) error and the absolute error. While the absolute error measures the hedging strategy accuracy in the real word, we do not only care about the accuracy, but we also care about the fact that the model can make a profit. So, the error function with a signed value is more meaningful than the absolute error where profit and loss are weighted the same.

Since the error can happen during the time to expiry we can introduce the error for each time spot t as follows

$$\begin{aligned} e_t &= E\left(\Delta_{BS}(S_t) - \Delta_{DDPG}(S_t)\right), \\ e_t^a &= E\left|\Delta_{BS}(S_t) - \Delta_{DDPG}(S_t)\right|. \end{aligned}$$

Given that $S_t/S_0 \sim \log \text{Normal}(rt, \sigma^2 t)$ the errors can be given as follows:

$$\begin{aligned} e_t &= \int_0^\infty \left(\Delta_{BS}(S) - \Delta_{DDPG}(S)\right) f_t(S) dS, \\ e_t^a &= \int_0^\infty \left|\Delta_{BS}(S) - \Delta_{DDPG}(S)\right| f_t(S) dS, \end{aligned}$$

where $f_t(x) = \frac{1}{x\sigma\sqrt{2\pi t}} e^{-\frac{(\log(x)-rt)^2}{2\sigma^2 t}}$.

Now we can integrate the whole errors over time and introduce the total error as follows:

$$e_t^{Total} = \Delta t \sum_{t=1}^{T} E\left(\Delta_{BS}(S_t) - \Delta_{DDPG}(S_t)\right), \tag{3}$$

$$e_t^{a,Total} = \Delta t \sum_{t=1}^{T} E\left|\Delta_{BS}(S_t) - \Delta_{DDPG}(S_t)\right|. \tag{4}$$

Finally, since we have done the training to fit 100 models we find the values for the total error for each fitted model and report the distribution, mean and its variance.

As it has been mentioned earlier we have divided the time interval $[0, 1]$ to 100 equal time sub-interval, so $\Delta t = 0.01$. We also have trained 100 models and first present the histograms of both errors in the following figures:

The mean and variance of the errors are also reported in the following table:

As one can see the errors average and variance are high. This is due to the fact that the pay-offs are only known in the final step. Actually, the DDPG method has not been able to learn the delta hedging strategy by playing blind for 99 steps in each episode. This shows we need to further investigate this problem by introducing a diagnosis method.

	e_t^{Total}	$e_t^{a,Total}$
Mean	2.85	26.98
Variance	102.92	31.68

Table 2: Mean and variance of e_t^{Total} and $e_t^{a,Total}$ for 100 trained DDPG models.

6 Diagnosis and improvement

To better understand how a model can be improved we need a diagnosis method. For the diagnosis, we chose to look at the stock and risk free assets for a fixed time to maturity t. According to the closed form solution that we have for the delta hedging we can find the values of the stock of value S_t in the portfolio by $S_t\Phi(d_+^t)$. For instance, in the following figure of the hedging for the fixed $T - t = 0.25$, time to maturity.

We can generate the same figures for the fitted model. This gives us a basis for comparison between the hedging portfolio of the fitted model and the delta hedging. In the figure 6, we can see a green line which is the correct performance action by delta hedging and which is our objective. Except the performance of 12,000 episodes, all others did well when stock price is low and high. They only did badly in the middle. No model fit is satisfactory. Even while the "symptom" is clear, we still do not have a direct answer why our model does not converge. It is not due to training length: training with 60,000 episodes is worse.

So we need to consider further modeling change.

In the figure 7, we can see the result for 100 fitted models for the time to maturity $t = 25$. As it is already mentioned each learning is based on 10,000 episodes.

As one can perhaps guess from the figure, using an ensemble method such as average or median can improve performance.

6.1 Average and median DDPG hedging

In Figure 6 the models are trained on the same data set, just with different numbers of training episodes. In order to use an ensemble method, we should train models on the same number of episodes but with different data sets. So let us assume we train M models and the resulting DDPG hedging strategies are given by

$$\Delta^l_{DDPG}(S_t), l = 1, 2, ..., M.$$

Then the average DDPG and median DDPG are defined as follows:

$$\Delta_{Average-DDPG}(S_t) = \frac{1}{M}\sum_{l=1}^{M}\Delta^l_{DDPG}(S_t), \tag{5}$$

$$\Delta_{Median-DDPG}(S_t) = median_{l=1,...,M}(\Delta^l_{DDPG}(S_t)), \tag{6}$$

So, without changing other things, we train several models with 10,000 episodes each, then save their weights. Then we can check the performance of the fitted models by taking average and median of 1, 4, 12, 20, 60 and 100 trained models.

6.2 Hedging cross-sections and surfaces

First, let us look at average and median ensembling using hedging cross-sections at times to maturity of: 25/100; 50/100; and 75/100 from the training in Figures 12, 13 and 14.

We can see that using an average ensemble improves the performance but that there are still regions of difference even using an average of 100 fitted models.

To better understand the hedging performance we can look at the hedging surfaces rather than the cross-sections. A hedging surface simply includes all the hedging shares for all times to maturity in a surface, where on the horizontal axis we have time and the stock price and on the vertical axis we have the in-stock value of a hedging strategy. Let us first look at the hedging surface of analytic delta hedging in Figure 8.

In Figures 15 and 16 we show the hedging surface for both the average and the median DDPG. Interestingly the surfaces for a single fitted model look better than the average and median for up to 20 models. However, this is an indication of the fact that the training results are not very stable. On the other hand, as the numbers increase the surfaces show more robust convergence to the ideal delta hedging surface.

In order to better compare the ensemble surfaces with ideal delta hedging we look at the difference between the hedging surfaces of the average and median DDPG hedging method and the delta hedging surface. In Figures 17 and 18. The two set of figures again can reaffirm the improvement in the stability and also the accuracy from increasing the number of the fitted models in the average and median DDPG hedging.

Finally we also want to look at the values of the errors and the standard deviation as measures of accuracy and stability. We use the total errors defined in (3) and (4), however we also define the standard deviation similar to the total errors as follows:

$$(std)_t^{Total} = \Delta t \sum_{t=1}^{T} std\left(\Delta_{BS}(S_t) - \Delta(S_t)\right), \tag{7}$$

$$(std)_t^{a,Total} = \Delta t \sum_{t=1}^{T} std\left|\Delta_{BS}(S_t) - \Delta(S_t)\right|, \tag{8}$$

where Δ is either $\Delta_{Average-DDPG}$ or $\Delta_{Medina-DDPG}$. In Tables 3, 4 and Figures 9, 10 we report the total errors and standard deviations for both the average and median DDPG methods.

M	Absolute error, Average-DDPG	Error, Average DDPG	Absolute error, Median-DDPG	Error, Median DDPG
1	28.24	-14.53	28.24	-14.53
4	20.120	1.3062	19.7638	-3.7157
12	13.7071	2.1002	13.9925	-1.6396
20	12.7192	2.1369	12.0564	-1.6708
60	11.303	1.5088	10.4679	-2.6965
100	11.5468	2.9346	10.6523	-0.9433

Table 3: Errors for different number of models, M, ensembled

n	Absolute STD, Average-DDPG	STD, Average DDPG	Absolute STD, Median-DDPG	STD , Median DDPG
1	39.39	20.42	39.39	20.42
4	28.134	1.9975	27.6024	5.0187
12	19.1159	2.9578	19.6667	2.2686
20	17.7077	2.9595	16.939	2.3563
60	15.7391	2.0626	14.7307	3.816
100	16.0628	4.0746	14.9604	1.3565

Table 4: Standard deviation of errors for different number of models, n, ensembled

From Table 3 and Table 4 of errors and the standard deviations respectively, we see that the ensembling methods can improve the accuracy and the stability of the model.

A graphical view of Table 3 and Table 4 is given in Figures 9 and figure 10. We see that the most improvement in error is complete by emsembling 20 models and there is only marginal improvement thereafter.

Although it is not evident from the error metrics, e.g. Table 3 or Figure 9, the hedging cross-sections and surface differences are generally much closer to the exact analytic solution for the median than for the average: compare Figure 17 to Figure 18. This may indicate better performance for more exotic payoffs.

Some bias remains and would need additional improvement to remove.

6.3 Hedging errors and an important observation

Finally, we report the errors of each method in Table 5. As one can see the median DDPG method closely performs like the true delta hedging method. Furthermore, while the average DDPG is not performing as good as the median DDPG but the errors are at an acceptable level comparing with the error generated by delta hedging.

	Delta hedging	Average DDPG	Median DDPG
Mean error	1.4876	2.0157	1.8927
std error	1.1038	1.4765	1.3942

Table 5: Errors of the three method for 100 step.

Now let us take a closer look at the delta hedging and the alternative ensemble DDPG hedging, from a practical perspective.

The delta hedging method when applied to the simulated data produces error. We have observed that this error to our surprise does not decline as fast as we expected. As you can see in Table 6 and in Figure 11 the mean error and the standard deviation reduce at a logarithmic rate with respect to the number of the hedging steps, which is too slow. This observation demonstrates that delta hedging is poor in practice even under perfect conditions. As a result if we have another hedging strategy which is almost as accurate as delta hedging, but also flexible to the type of the underling process, it will be more useful in practice. This is what the current paper will ultimately suggest as the errors of the ensembled DDPG methods are almost the same as the delta hedging.

	$N = 100$	$N = 1,000$	$N = 10,000$
Mean error	1.4876	0.4708	0.15
std error	1.1038	0.3567	0.1136

Table 6: Mean and std of the delta hedging method for 100, 1,000 and 10,000 steps.

This poor performance motivates us to look for another method that while being more flexible can perform at least as good as the delta hedging. We have examined the errors for delta hedging, versus the ensemble hedging, recalling results in Table 5, and saw that the numbers are very close to one another. This is really important since as alternatives ensemble DDPG methods are only path dependent and can learn the hedging strategies if enough scenarios or paths are fed to the machine.

7 Conclusions

This paper focused on the popular and simple delta hedging strategy to study some issues related to RL hedging that were neglected in the literature. The paper assessed a deep RL method, DDPG, in learning delta hedging in a complete market based on the Black-Scholes Model as a benchmark test.

More specifically, the paper identified and addressed two issues, first that if there could be just a single agent to learn hedging and second if the results can only rely on the final pay-offs. It was demonstrated that RL alone was unable to converge to the exact analytic hedging performance whatever the number of training episodes and for a range of different NN architectures.

However, using basic diagnostics suggested ensembling. Using the average and median ensembling with DDPG greatly improved the performance in terms of accuracy and stability. The median gave closer results to the exact analytic hedging surface but gave equivalent results in terms of error metrics. This suggests better performance on more complex payoffs or underlying processes. Some bias remains and would need additional improvement to remove.

Building a working model is important, but more importantly we are exploring the advantages and limitations of deep RL in financial/hedging and real world applications. One great advantage of using the RL in the way we have used is that it is a model-free method. This essentially means that the learning is only path data dependent and we can use more sophisticated simulated paths, like models with rough volatility, and learn appropriate hedging. Challenges remain in fitting a deep RL model. For deep RL, the longer the delay of getting rewards using a single pair of neural networks (actor and critic), the harder it is for the model to track back. This required use of ensemble methods. More extensive exploration may resolve the remaining issues and emphasises the need for customization with deep RL for financial applications.

References

Björk, T. *Arbitrage Theory in Continuous Time*, 3rd ed.; Oxford University Press: Oxford, UK, **2009**

Buehler, H., Gonon, L., Teichmann, J. and Wood, B., Deep hedging. *Quantitative Finance* , 1271-1291.Published online, **2019**

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

Du, J., M. Jin, P. N. Kolm, G. Ritter, Y. Wang and B. Zhang *Deep Reinforcement Learning for Option Replication and Hedging*, Journal of Financial Data Science, Fall 2020, DOI: https://doi.org/10.3905/jfds.2020.1.045, **2020**

Giurca, A. and Borovkova, S. *Delta Hedging of Derivatives using Deep Reinforcement Learning*, Available at SSRN: https://ssrn.com/abstract=3847272

Goodfellow, I., Y. Bengio, and A. Courville *Deep learning*, 3rd ed.; MIT press: Cambridge, USA, **2016**

Halperin, I., Qlbs: Q-learner in the black-scholes (-merton) worlds.". *The Journal of Derivatives* **2020**

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

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* **2015**

Ruf, J. and Wang, W., Neural Networks for Option Pricing and Hedging: A Literature Review. *Journal of Computational Finance* **2020**

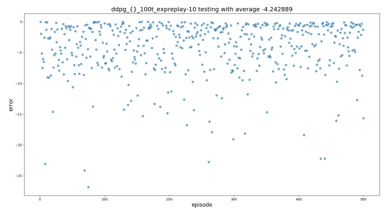
Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. and Riedmiller, M., Deterministic policy gradient algorithms. *In Proceedings of the 31st International Conference on International Conference on Machine Learning* Volume 32 (ICML'14). JMLR.org, I–387–I–395 **2014**

Sutton, R. S. and Barto,A. G. *Reinforcement Learning: An Introductions*, 2nd ed. The MIT Press, Cambridge, USA, **2018**

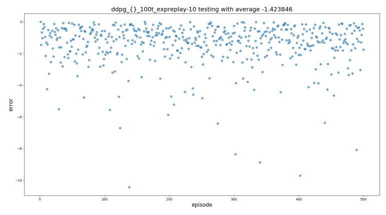
Tamar, A, D. D. Castro, and S. Mannor *Learning the variance of the reward-to-go*, Journal of Machine Learning, 17: 1-36, **2016**

8 appendix

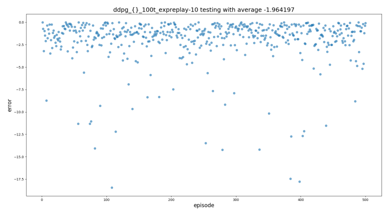
12,000 episodes, average error: -4.24.



20,000 episodes, average error: -1.42.



30,000 episodes, average error: -1.96.



45,000 episodes, average error: -1.70.

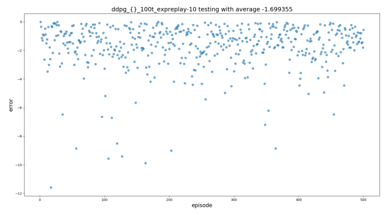


Figure 3: Performance of DDPG for different training lengths

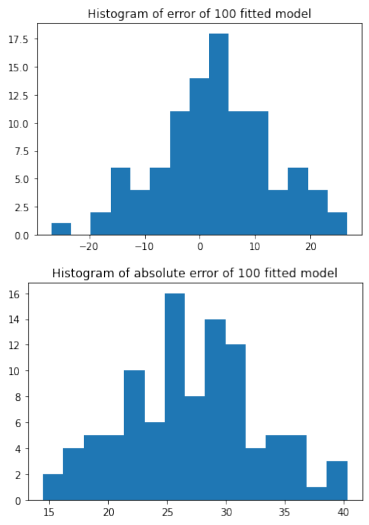


Figure 4: Histogram of the errors for 100 models each trained on 10,000 episodes.

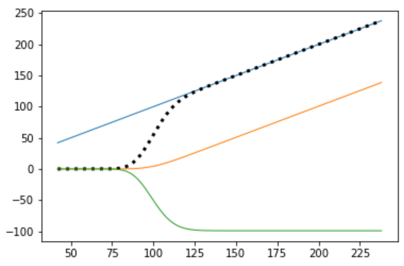


Figure 5: The value of the stock and the risk free for 25 steps time to maturity.

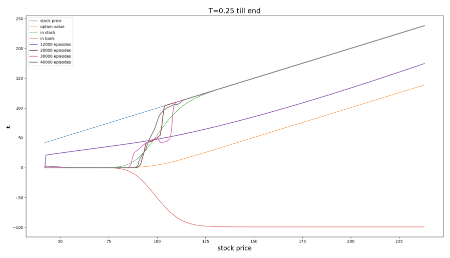


Figure 6: Lines showing model performance after 12,000, 20,000, 30,000, 45,000 episodes of training. There is little convergence

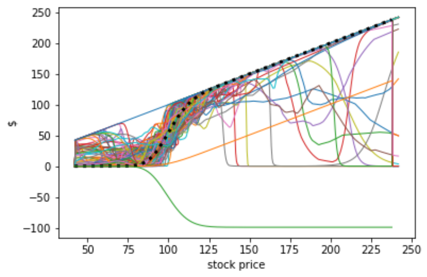


Figure 7: DDPG hedging cross-sections for 100 fitted models at time to maturity 25/100.

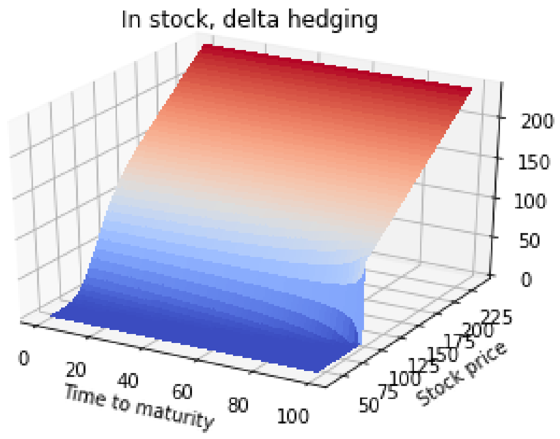


Figure 8: Ideal delta hedging surface from analytic expression.

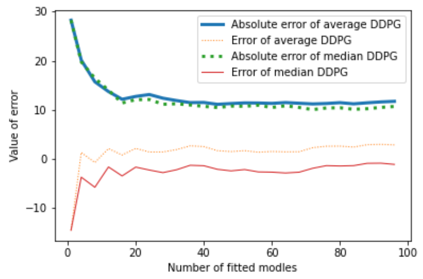


Figure 9: The errors of increasing numbers of ensembled fitted models.

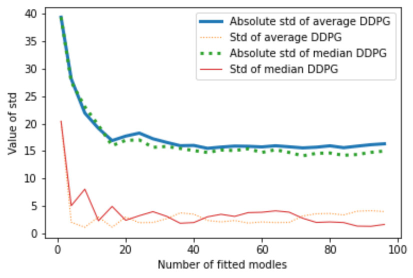


Figure 10: The standard deviations errors of increasing numbers of ensembled models.

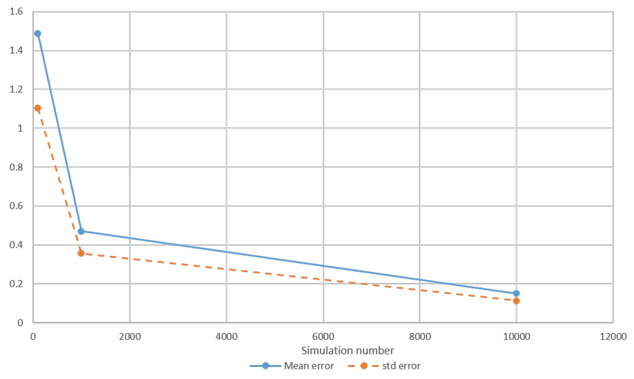


Figure 11: Mean error and std error of delta hedging for different timesteps.

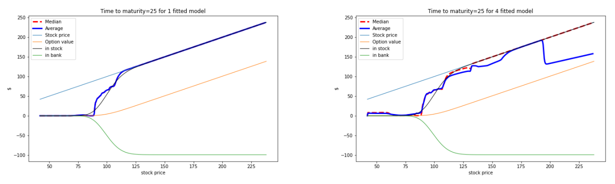


Figure 12: Hedging cross-section for average (blue) and median (red) DDPG at time to maturity 25/100

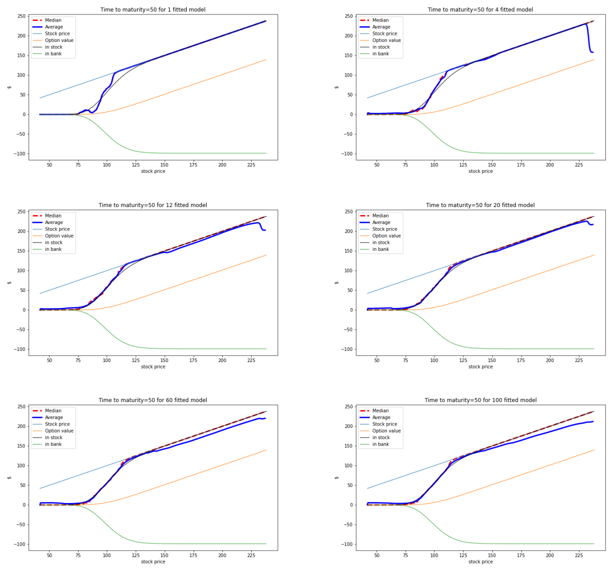


Figure 13: Hedging cross-section for average (blue) and median (red) DDPG at time to maturity 50/100

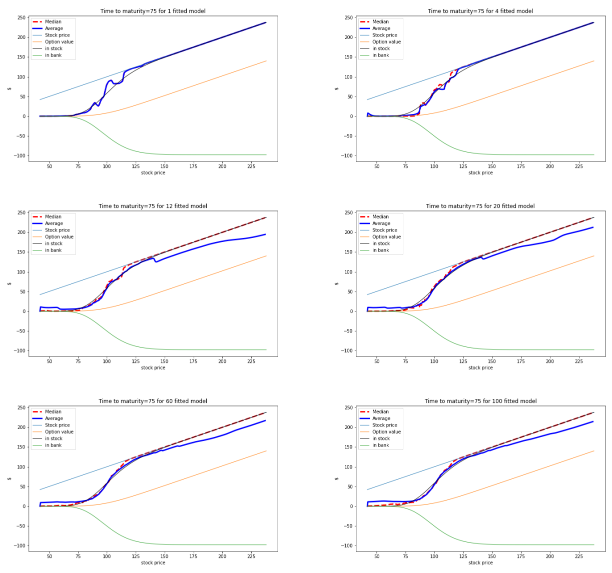


Figure 14: Hedging cross-section for average (blue) and median (red) DDPG at time to maturity 75/100

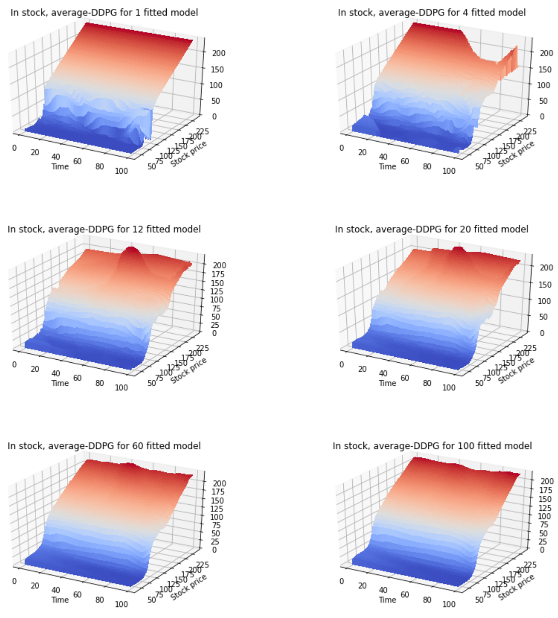


Figure 15: Hedging surface for average DDPG

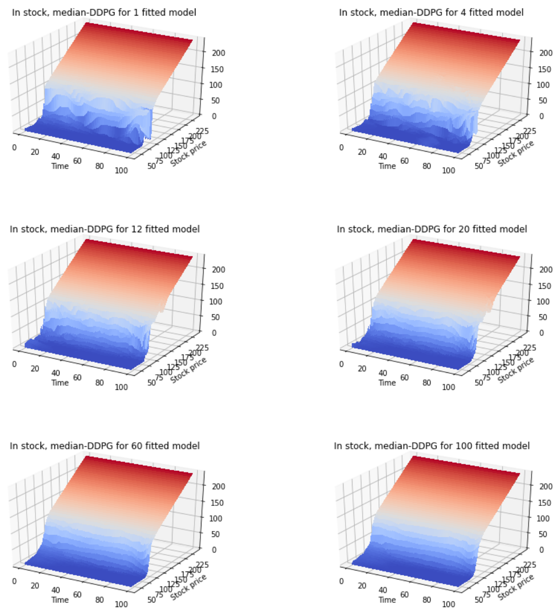


Figure 16: Hedging surface for median DDPG

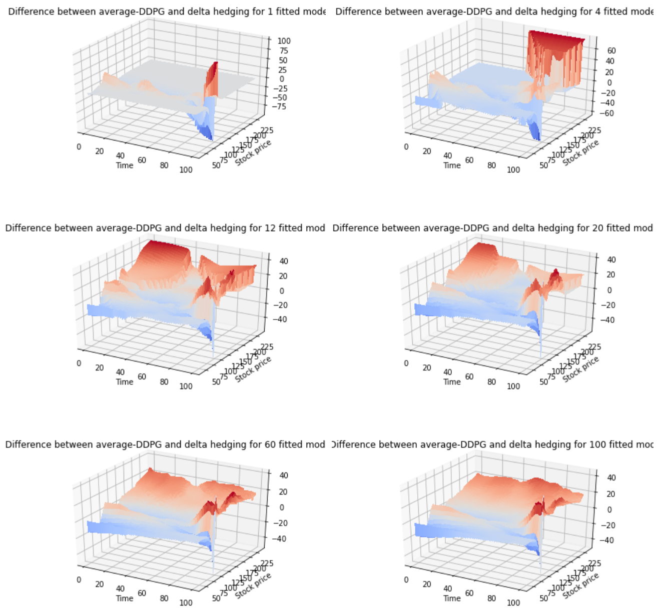


Figure 17: Differences in hedging surfaces for average DDPG hedging

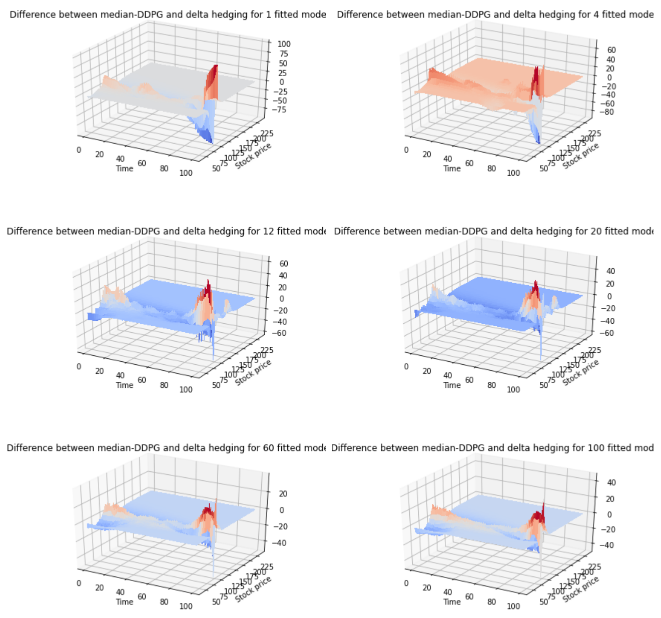


Figure 18: Differences in hedging surfaces for median DDPG hedging