Hedging using Deep Reinforcement Learning: Replication using Options

Shwetha Salimath, Sauraj Verma, Raghuwansh Raj

October 8, 2023

Introduction

Hedging is a trading strategy that aims to reduce, or hedge, the directional risk associated with price movements in the underlying asset. The approach uses options to offset the risk to either a single other option holding or an entire portfolio of holdings. The investor tries to reach a delta neutral state and not have a directional bias on the hedge. One of the drawbacks of delta hedging is the necessity of constantly watching and adjusting positions involved for the options involved in the hedging. It can also incur trading costs as delta hedges are added and removed as the underlying price changes.

Hedging of stocks is one of the fundamental concepts of finance. According to the Black-Scholes Model principle, in a complete and friction less market one can perfectly replicate the option by using a continuously rebalanced dynamic trading strategy in the stock and risk-less security. But in real world scenario, no such friction-less market exists with the main friction is the trading cost. In most cases, the hedge itself is not static but needs to be continuously readjusted. Nonetheless, both problems are related in the sense that one wishes to minimize the following two:

- The trading costs
- The deviation from the optimal hedge

Most of the hedging approaches use the idea of mean-variance optimization, that the goal is to maximize the mean returns from a given portfolio while keeping its variance at a minimum. Since hedging is done to reduce the losses of a security, it usually requires understanding the cointegrated interaction between two or more securities in order to understand the hedging dynamics between them. In our paper, we focus on using reinforcement learning to train an agent to understand the dynamics of the market and learn how to optimally trade off trading costs versus hedging variance for that security. The method implemented extends in a straightforward way to arbitrary portfolios of derivative securities. For example, envision a trader who has inherited a derivative security that he or she must hold to expiration because of some exogenous constraint. The trader has no directional view on the derivative or its underlier. With the method proposed in this article, the trader can essentially press a button to train an algorithm to hedge the position. The algorithm can then handle the hedging trades until expiration with no further human intervention.

Related work

Motivated by risk reduction, hedging a stock portfolio with index futures has been an active research topic since it was introduced. A hedger supposes that return of a hedged position (e.g., stock portfolio) can be closed to risk-free interest rate. In terms of optimal hedge ratio H_r, there are many methods used to estimate the ratio. i.e the traditional hedge (MVHR), least trimmed squares (LTS), and beta ratio of cross-hedge.The results suggest that MVHR and LTS methods are robust to estimate the ratio.

The MVHR was introduced to work around for the problem by taking the imperfect relationship of prices into account and determine the optimal ratio hr. Let Rs, Rf, and Rh are returns of spot position (e.g., open portfolio), futures positions (e.g., index futures for hedging), and the hedged portfolio with futures, respectively, then we get

$$Rh = Rs + hRf \tag{1}$$

$$Var(Rh) = Var(Rs) + h2Var(Rf) + 2hCov(Rs, Rf) \tag{2}$$

The optimal ratio h (or hr) to minimize the Var(Rh) is:

$$hr = Cov(Rs, Rf)/Var(Rf) \tag{3}$$

Stating that traditional methods to estimate optimal hedge ratio are misspecified, error correction model (ECM) was proposed to estimate optimal hedge ratio and forecast out of sample for evaluation as in Ghosh (1993). Firstly, it carries out cointegration test. Secondly, it use OLS regression to estimate error correction model. The model incorporates relationship of the long-run equilibrium as well as the short-run dynamics. The result shows that optimal hedge ratio is significantly improved with adjusted

R2 from ECM which is higher than traditional methods. Also, by comparing root-mean-squared error (RMSE), out-of-sample forecasts from the ECM are found to be better than other methods. Reinforcement learning was proposed to train trading systems to make profit and to adjust risk Moody. The result shows that reinforcement learning can avoid large losses when market crashed. Basis risk hedging strategy was developed using reinforcement learning as in Watts. Without assets modeling requirements, state–action–reward–state–action (SARSA)-based algorithm was applied to find an optimal trading policy to hedge a non-traded asset. Q-learning is proposed to extend Black–Scholes–Merton (BSM) model for option pricing.

1 Reinforcement Learning

RL has been developed largely independently from classical utility theory in finance. It provides a way to train artificial agents that learn through positive rein- forcement to interact with an environment, with the goal of optimizing a reward over time. The learning agent does this through simple trial and error by receiving feedback on the amount of reward that a particular action yields. In contrast to supervised learning, an RL agent is not trained on labeled examples to optimize its actions. In addition, RL is not trying to find a hidden structure in unlabeled data and hence is different from unsupervised learning.

RL allows us to solve complex problems that cannot be directly solved using numerical methods or under any functional approxmiation. Because RL utilizes the idea of a Markov Decision Process (MDP), the agent that follows the MDP learns to interact with an environment by means of the sequences of actions they take, with the goal of optimizing a cumulative reward over time. At each time step t, the agent observes the current state of the environment $s_t \in S$ and chooses an action $a_t \in A$. This choice influences both the transition to the next state, s_{t+1}, and the reward, R_{t+1}, the agent receives. The agent's goal is to choose actions to maximize the expected cumulative reward:

$$\mathbb{E}[G_t] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ..] \tag{4}$$

Where constant $\gamma \in [0, 1]$ is referred to as the discount factor. The sum of the cumulative rewards can either be finite or infinite. A policy π is a strategy for determining an action a_t, conditional on the current state s_t. Policies can be deterministic or stochastic. In the deterministic case, π

maps $S \longrightarrow A$; in the stochastic case, π maps a state $s \in S$ to a probability distribution $\pi(a|s)$ on A.

In our paper, we focus on using Q-learning based on a deep neural network approach, also known as Deep Q-Network (DQN).

1.1 Q-Learning and Deep Q-Learning

Q-learning is one of the temporal difference methods for computing the value of a given state when we do not have complete information about the MDP. The action-value function $Q^\pi : S \times A \longrightarrow \mathbb{R}$, also known as the Q-function, expresses the value of starting in state s, taking action a, and following policy π thereafter:

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{5}$$

The state-value function is defined as the action-value function in which the first action also comes from the policy π. The goal of Q-learning is to learn Q^*, The optimal action-value function satisfies the Bellman equation:

$$Q^*(s, a) = \mathbb{E}[R + \gamma max Q^*(s', a')|s, a] \tag{6}$$

The basic idea of Q-learning is to turn the Bellman equation into the update $Q_{i+1} = \mathbb{E}[R+\gamma max Q^*(s', a')|s, a]$ and iterate the function until convergence is approached. Once one has determined the optimal action-value function, the optimal policy can be computed as:

$$\pi^*(s) = argmax Q^*(s, a) \tag{7}$$

With Q-learning we encounter two problems,

- The amount of memory required to save and update that table would increase as the number of states increases.
- The amount of time required to explore each state to create the required Q-table would be unrealistic, hence making the value iteration intractable.

Deep Q-learning

We overcome the problems of Q-learning by using a deep learning model (DQN) to approximate the Q-values. In our case the neural network used consists of linear layers as we are dealing with just 1D problem. In DQN, the action-value function is approximated with a DNN, $Q(s, a; \theta) \longrightarrow Q^*(s, a)$, in which θ represents the network parameters. The DNN is then trained by minimizing the loss between the predicted Q-value from the DQN and the actual value of the DQN

In DQN we need to feed the state to the DQN, which would return the Q-values of all possible actions in the state. Select one action according to ϵ-greedy approach, which gives maximum Q-value. Thus, we move from state s to next state s' to receive the reward. We store the transition in our replay buffer as $\{s, a, r, s'\}$. We then sample random batches of replay buffer to calculate the loss. The loss is the mean squared error between the target Q and predicted Q:

$$\mathcal{L}_i(\theta_i) = (Q_i - \hat{Q}_i)^2 \tag{8}$$

The loss update happens on the basis of the target network and the policy network. In our computations, the DQN starts to train after 2000 episodes and then uses the replay buffer to update it's parameters over the remaining training episodes.

Environment

We have a main trading environment class, consisting of many functions, to give us the state, action and reward. We have a function to initialise parameters and to generate the stimulation of data when calling for the environment. Then we have a function to reset, to take next step and to calculate the reward.

We look at the simplest possible example: a European call option with strike price K and expiry T on a non-dividend-paying stock. We take the strike and maturity as fixed, exogenously given constants. For simplicity, we assume the risk-free rate is zero. The agent we train will learn to hedge this specific option with this strike and maturity. It is not being trained to hedge any option with any possible strike/maturity.

The agent comes into the current period with a fixed option position of L contracts. We assume for simplicity that this option position will stay the same until the option either is exercised or expires; we are training an agent to be an optimal hedger of a given contract, not an agent that can decide not to hold the contract at all. Each period, the agent observes a new state and then can decide on an action. Available actions always include trading shares of the underlying, with bounds dictated by the economics of the problem. For example, with L contracts, each for 100 shares, one would not want to trade more than $100 \times L$ shares. If the option is American, then there is an additional action, which is to exercise the option and hence buy or sell shares at the strike price K.

State Space

Our state variable consists of P_t price at time t, τ being the time to maturity, which is computed as $\tau = T - t$ and n_t being the number of shares held by the agent at the time after action has been taken.

$$S : {\mathbb{R}_+}^2 \times \mathbb{Z} = \{(P_t, \tau, n_t) | S > 0, \tau > 0, n \in \mathbb{Z}\} \tag{9}$$

Action Space

After observing the state, the agent takes an action a_t by choosing the integer amount of the stocks to trade from the action space:

$$A = \{-100L, ..., 100L\} \tag{10}$$

In the above equation L is number of option contracts and 100 is number of shares in each contract. We have a discrete set of action space which lets us know how many shares should the agent hedge to maximize his profits. The action space was kept discrete to make the computations tractable.

Reward Function

In order to define the reward function, we would need to observe the price in order to determine the reward the agent should get for a certain risk aversion factor. If the log price process is a random walk, then the wealth can be decomposed as:

$$\delta w_t = q_t - c_t \tag{11}$$

Where q_t is the random walk term and c_t is the total trading cost paid in period t. With the problem of maximizing the expected wealth accumulated by the agent, the reward can be defined as

$$R_t = \delta w_t - \frac{\kappa}{2}(\delta w_t)^2 \tag{12}$$

Where κ is the risk-aversion factor associated with the portfolio, which is also user assigned when initialising the environment. By plugging each one-period reward into the cumulative reward, we obtain an approximation of the mean–variance objective. Thus, training reinforcement learners with this kind of reward function amounts to training the expected-utility maximizers.

In the context of option hedging, it amounts to training automatic hedgers that are prepared to optimize the trade-off of costs versus variance from being out of hedge.

Rebalancing and Trading Costs

Our goal for the agent is to minimization of variance and trading cost while maximizing the profits in an option hedge. For n shares, the trading cost is,

$$cost(n) = C \times TickSize \times (|n| + 0.01n^2) \tag{13}$$

C is the intensity of the bid-offer spread and TickSize is used to compute the cost relative to the midpoint of the bid-offer. Both of these can be set by the user when initialising the environment. We have taken them as C = 1 and TickSize = 0.1.

Simulations and Training of the RL Agent

We require lots of data to train an RL agent, and as we do not usually have that amount of data in real life, we have created a simulation environment to create data. The simulator consists of three main parts:

- Simulate a Geometric Brownian Motion with pre-specified parameters of $\mu = 0.05$ and drift factor as 0.01, and a daily lognormal volatility, $\sigma = 0.01$. We generate a total of 3,000 simulated time series, with each simulated time series having 50 observations , in a total giving 150,000 price observations for the agent to learn from.
- We define the strike price $K = 100$, the initial stock price $S_0 = 100$, the time to expiry $T = 50$, and the risk aversion factor $\kappa = 0.1$
- We simulate an environment where the agent needs to learn to hedge it's position in a long call option.

We use the Black-Scholes equation to compute both the call and put option price and delta. Once we have our values and the simulations done, we define the environment and train our agent for the entirety of the simulation. The training takes a total of 4 hours on GPU, and then we assess the performance of our agent on out-of-sample data, a total of 30,000 observations.

Results and Conclusion

After we train our agent, we assess it's performance on out-of-sample data. The RL agent is at a disadvantage, initially. Recall that it does not know any of the following pertinent pieces of information:

- The strike price K
- The fact that the stock price process is a GBM
- The volatility of the price process
- The Black-Scholes model formula
- The payoff function $(S - K)^+$ at maturity
- Any relevant information about Option Greeks

It must infer the relevant information from these variables, insofar as it affects the value function, by interacting with a simulated environment.

Figure 1 shows the out-of-sample performance of our trained RL agent on one of the 30,000 out-of-sample simulations. We see that the agent learns to hedge the stocks using a long call option whenever the trading costs tend to go up, and the agent will immediately drop the hedging strategy when the costs lower and the total profit on the stocks is not dominated by the trading costs. The essence of this hedging strategy is that under the variance-minimization criterion, the agent chooses to reduce the losses it can make by shifting it's portfolio towards the option in order to take advantage of the Δ-gain it gives, directly covering up the losses made from the stock transaction costs and making the hedge an optimal strategy.

We also need to remember that the multiplier C is set here at 1 to represent some degree of market friction, which is the reason why the agent tends to do an asymmetric hedge against the stock, as for a small drop in the stock PnL, the agent will aggressively hedge it using a long call option. Further on, we explore the performance of our agent over one trading year of 252 observations to see how it performs. Figure 2 shows the simulated GBM with a strong positive trend, showing that the underlying stock is on a long position, and Figure 3 shows how the agent uses the price change in the underlying to dynamically hedge the transaction costs using it's long-call strategy.

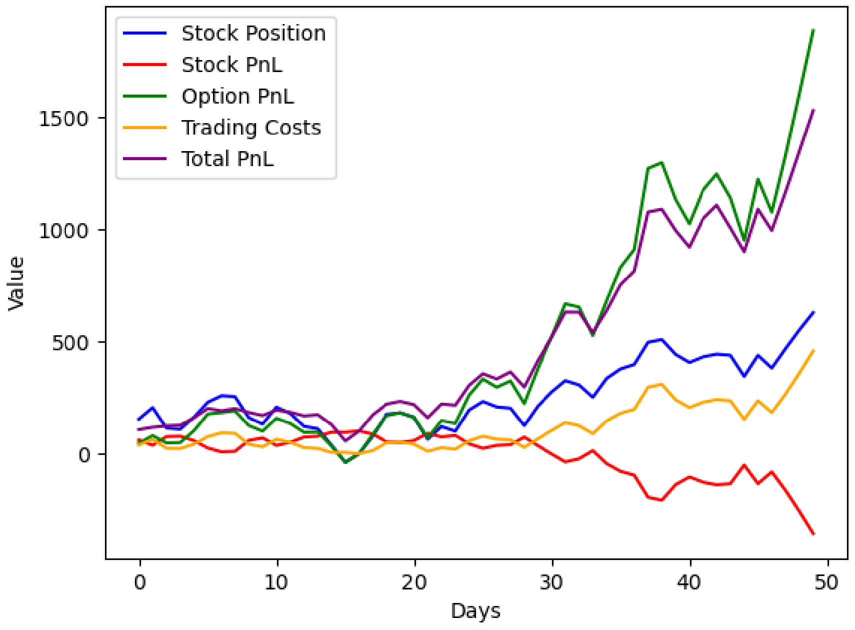

Figure 1: The trained RL agent's hedging on an out-of-sample series

Further, we analyze the impact of the parameters on the profitability in our hedging strategy. Figure 4,5 and 6 show the distribution of the PnL for the agent, with Figure 4 showing the total PnL with respect to change in κ, Figure 5 showing the total PnL with respect to C, and Figure 6 showing the option PnL with respect to κ.

Off the bat, we observe that with a higher risk-aversion, the agent tends to hedge even more aggressively than before as the density of the PnL becomes more leptokurtic over time. This is also indicative of the fact that higher risk-aversion factors arise from an increased chance of encountering fat-tailed events, which causes the agent to behave in a more aggressive manner to mitigate any further losses. Next, we see the impact of the market friction multiplier and notice that with a higher friction, the agent's PnL range widens by a large margin, with a multiplier of 5 showing a high degree of friction, stating that under extreme friction settings, the agent's ability to make profits is lower due to a large number of other agents competing in the market for profits and alpha.

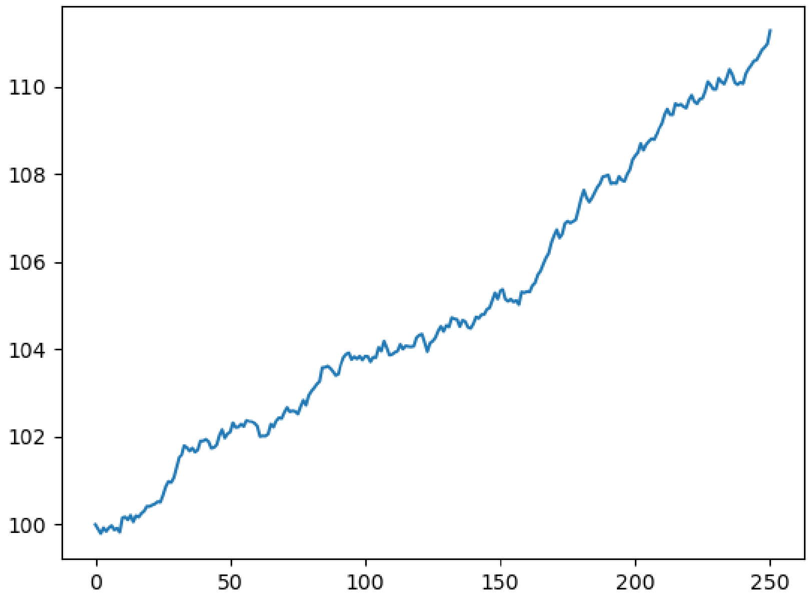

Figure 2: A GBM simulation of 252 days

Lastly, we observe the agent's option PnL and notice that the PnL distribution is symmetric at 0, with the frequency of profits going higher as the risk-aversion factor goes up. This can be related to the agent's actions that change on the basis of the reward function; a higher risk-aversion factor penalizes the agent even more, which forces the agent to take actions of buying a larger share of positions in the stock in order to hedge it's losses further and reduce the degree of risk it can perceive.

Conclusion

The main contribution of this article is to show that with RL one can train a machine learning algorithm to hedge an option under realistic conditions. Somewhat remarkably, it accomplishes this without the user providing any relevant pieces of information. A key strength of the RL approach is that it does not make any assumptions about the form of trading cost. RL learns the minimum variance hedge subject to whatever transaction cost function

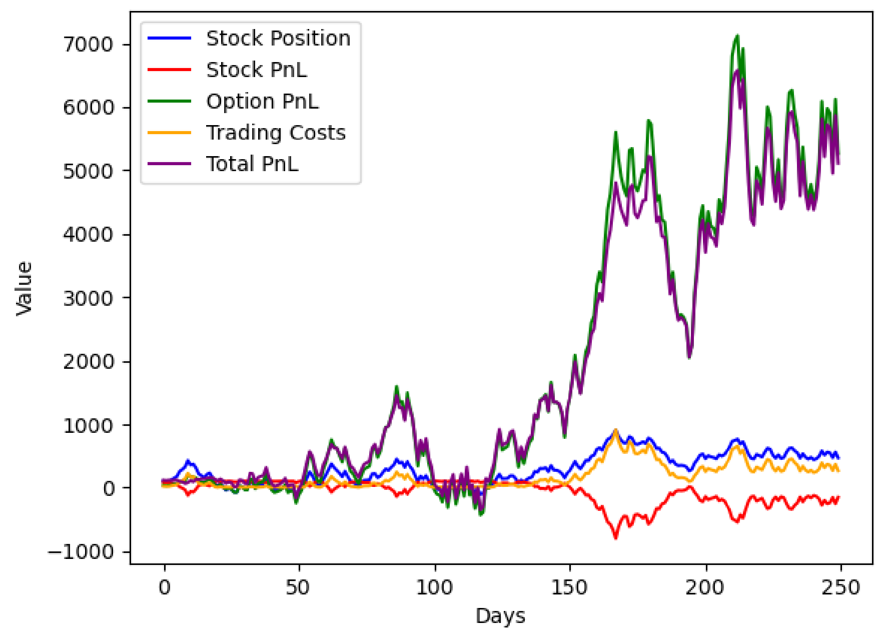

Figure 3: The trained RL agent's hedging on an out-of-sample series over the 252 days

one provides. All it needs is a good simulator in which transaction costs and options prices are simulated accurately. This has the interesting implication that any option that can be priced can also be hedged, whether or not the pricing is done by explicitly constructing a replicating portfolio—whether or not a replicating portfolio even exists among the class of tradable assets.

The approach of dynamic hedging done here optimally learns to trade off variance and cost using whatever assets it is given as instruments to use in a hedging portfolio. In other words, the agent is able to find the minimum-variance dynamic hedging approach, and this holds importance because markets are not complete and some of the assets required for perfect replication might not be present. We further wish to study the problem of dynamic hedging by adding in even more state variables, such as the option delta and the range of strike prices we can use.

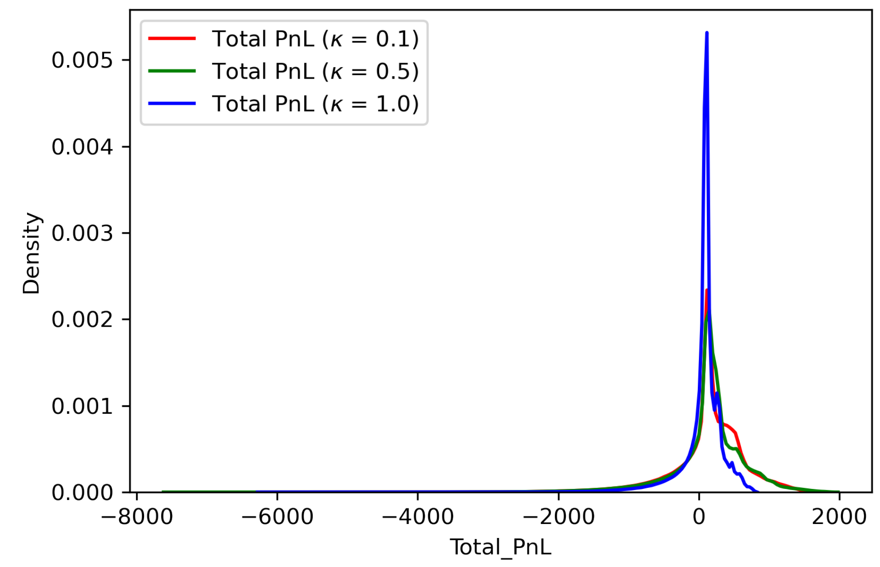

Figure 4: The agent's total PnL across varying κ

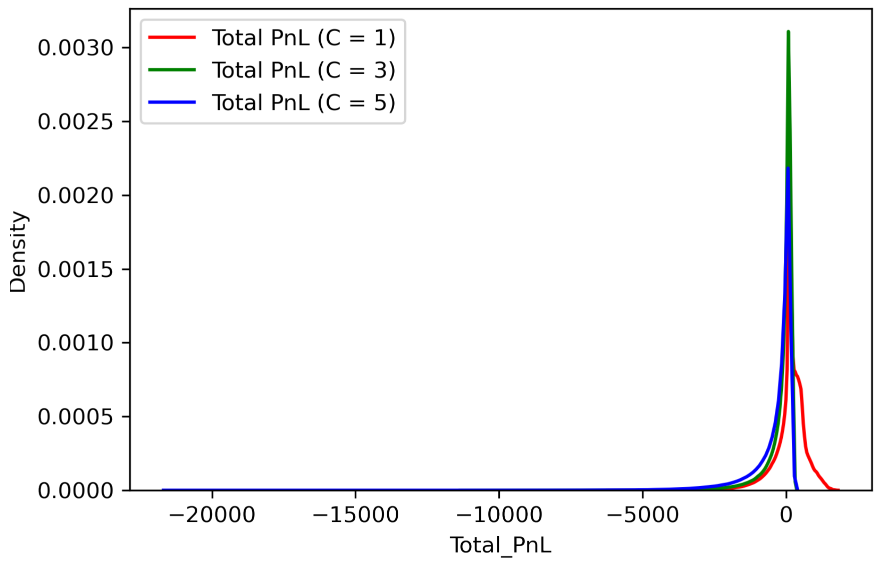

Figure 5: The agent's total PnL across varying C

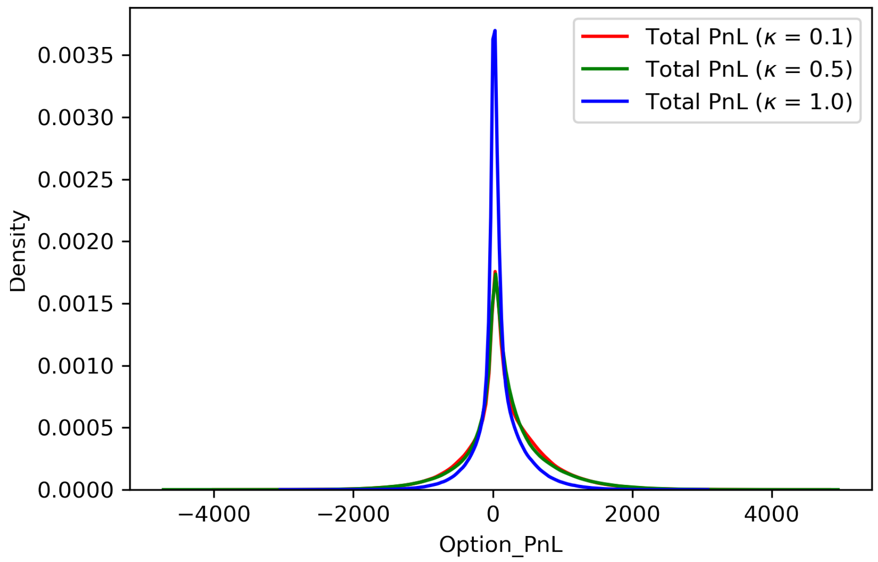

Figure 6: The agent's Option PnL across varying κ