RL for Options Hedging

Ashwin Siripurapu

Courant Institute of Mathematical Sciences, New York University
ars991@nyu.edu

1 Introduction

Reinforcement learning is a branch of machine learning concerned with finding the optimal sequence of actions to take over multiple periods of time when there may be uncertainty about the outcome of each action. In the past several years, reinforcement learning techniques have enjoyed considerable success in solving difficult problems in robotics, game–playing, and stochastic control.

The purpose of this project is to apply reinforcement learning techniques to the problem of hedging European and American options. We build primarily off of the work of [Du et al., 2020]; in particular, we seek to build a more general software system that can handle a broader range of trading and hedging problems. We seek also to replicate the aforementioned work in the case of European options, and achieve similar performance in the case of American options.

A reinforcement learning problem must be formulated in terms of an *agent* which interacts with an *environment* over the course of multiple time periods. In each time period, the agent receives an *observation* from the environment; it then performs an *action* in the environment, for which it receives a *reward* from the environment. Over time, the agent learns to perform actions that lead to high rewards.

The problem that we wish to solve is that of delta–hedging an options contract. Specifically, we desire to train an agent that trades the underlying stock of a particular options contract every day until the option expires (or is exercised early, in the case of an American option; see Section 6 for further discussion). How well the agent hedges is measured by the variance of the wealth of the agent's holdings: if the agent is always theoretically optimally hedged, then the agent's total wealth should remain constant (i.e., it should have zero variance).

The concrete formulation of our problem as a reinforcement learning problem is as follows:

- The *agent* is a deep neural network–based RL agent.
- The *environment* is a representation of the market in which the agent can trade.

- The *observations* received by the agent are the prices of the various assets in the market, and possibly additional market information (see Section 4.1 for more details).

- The *actions* taken by the agent are decisions to buy or sell various amounts of the stock underlying the contract we are trying to hedge (see Section 4.2 for more details).

- The *reward* experienced by the agent is a proxy for how well the agent has hedged; intuitively, if the agent takes actions which lead to high variance of wealth, it should be penalized with a low reward, and if the agent makes money it should be rewarded with a high reward (see Section 4.3 for more details).

The specific implementations of the action space, observation space, and reward space are covered in Section 4.

2 Software Abstractions and Libraries

2.1 OpenAI `gym` and `stable-baselines3`

> If I have seen further it is by standing on the shoulders of Giants.
>
> *Sir Isaac Newton*

This project would not have been possible without the existence of two key reinforcement learning libraries: OpenAI's `gym` library, and the `stable-baselines3` library.

The `gym` library provides a standard interface for reinforcement learning environments. Concretely, the library provides an abstract class `gym.Env` with methods that all valid reinforcement learning environments must implement. The most important of these methods are `gym.Env.step`, which takes as input an action from the agent and outputs the next observation and the agent's reward from taking that action; and `gym.Env.reset`, which resets the environment back to its initial state after a training episode is finished.

The `stable-baselines3` library provides a collection of reinforcement learning algorithms that are designed to work in tandem with environments conforming to the `gym.Env` interface. In particular, `stable-baselines3` provides off–the–shelf implementations of various deep reinforcement learning models such as PPO, DQN, and Pop-Art, and provides code to train these models on an arbitrary environment provided by the user, so long as the environment conforms

to the `gym.Env` interface.

In this project, we implement a modular software system around the `gym.Env` interface. We then plug our environment into the interface provided by `stable-baselines3`, and rely on that library to train our deep reinforcement learners. Of course, tuning the hyperparameters of the `stable-baselines3` models remains a significant challenge.

2.2 `golds`: Generic Options Library for Data Scientists

A founding goal of this project was to implement a well–architected software system so that future researchers could easily experiment with different RL techniques within the framework afforded by OpenAI `gym` and `stable-baselines3`.

To that end, we implement a modular system of extensible and reusable components named `golds`, which stands for "Generic Options Library for Data Scientists". [1] This system can be used to formulate a variety of trading problems as RL problems, and then train agents to solve those problems using the aforementioned libraries. Figure 1 illustrates the various components and how they interact. A brief explanation of the various components follows.

- `AmericanOptionEnv` is the main entry point into the system. This class is a subclass of OpenAI's `gym.Env`.
  This class is responsible for receiving *actions* from the agent once per time step. In our setting, the agent's actions represent *trades* that it wishes to make in the market. The environment performs those trades on behalf of the agent and receives updated prices (via the `PricingSource` [see below]), and also uses the agent's trades to compute a *reward* (via the `RewardFunction` [see below]).

  The new prices constitute an *observation* which is then returned to the agent, along with the reward.

- `PricingSource` is responsible for tracking the state of the market and passing price data back to the `AmericanOptionEnv`. To do this, it must have access to a `MarketDataSource` object, which gives it raw (unadjusted) market price data, as well as a `TransactionCostModel` object, which tells it how to adjust the raw market price data from the `MarketDataSource` in order to account for the agent's trading activity (e.g. if the agent buys shares, that will drive up the price of the stock due to market impact).

  The `PricingSource` needs to receive the agent's trades from the `AmericanOptionEnv`, because it needs to pass them along to the `TransactionCostModel` to compute market impact.

[1] It is also a whimsical reference to Gold's Gym, in continuation of the "gym" metaphor from OpenAI.

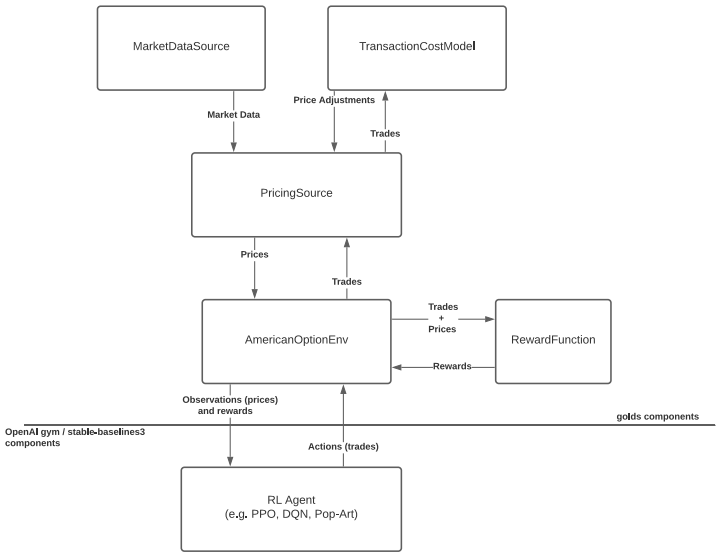


Fig. 1. Diagram of the components and data flows of `golds` and `gym/stable-baselines3`

– `MarketDataSource` is the source of all raw market data, i.e., market data that is *not* adjusted for the market impact of the agent's trading activity. A user can implement a subclass of this class in order to provide market data according to whatever mechanism fits the user's purposes: for example, simulating random asset prices according to a geometric Brownian motion, returning historical market data from a file or database, or even passing along live market data from a stock exchange!

– `TransactionCostModel` takes, as input, the trades carried out by the agent (which it gets from the `PricingSource`, which in turn gets the trades from `AmericanOptionEnv`). Internally, it computes the market impact caused by the agent, and returns to the `PricingSource` a *price adjustment* for each asset traded by the agent.

  The `PricingSource` combines this price adjustment with the raw prices from the `MarketDataSource` to get a market price, which it then returns to `AmericanOptionEnv` to pass on to the agent as an observation.

For the purposes of this project, we use a "dummy" transaction cost model which always returns zero price adjustment (i.e., the agent's trading never has any market impact).

– `RewardFunction` is the component that computes the *reward* that the agent receives at each time step. To do this, it takes as input the agent's trades and the market prices, and computes a reward based on that data, together with any internal state that it may maintain.

  For example, a reasonable reward function might reward an agent for making profitable trades while punishing an agent for losing money.

In addition to these fundamental components, there is also code for creating various assets (currencies, stocks, and options on stocks) and pricing those assets.

3 Data Sources

Deep learning techniques have developed a well–deserved reputation for requiring vast amounts of training data in order to function properly. Therefore, we considered the following mechanisms for acquiring the necessary amount of data.

3.1 Simulated Data

Following the work of [Du et al., 2020], we simulate stock market price data to train our RL agent. As mentioned in Section 2, there exists an extensible market data component (the `MarketDataSource` class) in the `golds` library which enables users to configure, in Python code, exactly how market data should be generated.

In our experiments, we implement a simple subclass of `MarketDataSource` to simulate prices from the starting day $t = 0$ through the final day $t = T$ at the end of the episode. Given parameters $r \geq 0$, $\sigma > 0$, and $\mu > 0$, indicating respectively the risk–free rate, annual stock return volatility, and annual stock return, and assuming an annualization convention of 252 trading days in the year, the simulation procedure is as follows:

1. The initial price of cash is set to 1. At any time $t > 0$, the price of cash is simply the price of cash at time $t - 1$ multiplied by $(1 + (r/252))$.
2. The initial stock price S_0 is drawn from some known distribution. To derive the stock price at any time $t > 0$, we apply a normally–distributed log return with mean $\mu/252$ and standard deviation $\sigma/\sqrt{252}$ to S_{t-1}, i.e.,

$$x_t \sim \mathcal{N}\left(\frac{\mu}{252}, \frac{\sigma}{\sqrt{252}}\right)$$
$$S_t \leftarrow S_{t-1}\left(1 + x_t\right)$$

This way, the stock price S_t follows a geometric Brownian motion.

3. At any given time $0 \leq t \leq T$, the price of a call or put option on the underlying stock S is given by the Black–Scholes formula or the Andersen model for European or American options, respectively. These formulae depend only on properties of the option (strike price and expiry date), t, r, and σ, and S_t, all of which are known at time t.

Because our portfolios only ever consist of one cash asset, a single stock S, and options on the stocks, this simulation technique is sufficient.

3.2 Historical Market Data

The idea of using historical market data for training the RL agent was broached early in the project. The advantages are obvious: training on real historical market data would be essential for any industrial application of this technique, and moreover the use of historical market data provides an excellent test of how robust RL is to some of the assumptions made by the simulation technique described above (in particular, constant average rate of return μ and constant volatility σ).

However, historical options data is difficult and expensive (in terms of money, computing power, and human effort) to acquire and process, particularly when intraday data is needed. By way of example, one of the problems encountered in handling historical options market data is that data vendors frequently provide pricing data for a given options contract only on days when the contract is sufficiently close to the money (i.e., only on days t when the strike price K of the contract is sufficiently close to S_t, or in other words, on days when the absolute value of the option's delta is sufficiently close to 0.5). If we were to use this data for training purposes, we would have to restrict ourselves to training only on episodes (that is, years) when our options contracts never went too far out of the money, which is a form of bias in the construction of the training set.

In addition to the difficulties of acquiring and processing the data, the total amount of historical options market data available is not sufficient to train deep reinforcement learners: as we soon discovered, it typically takes millions to tens of millions of training data points in order to successfully train an agent to hedge properly. Even under the extremely generous assumptions that a given stock had, on each day throughout the past decade, market prices for both put and call options at each of 100 different strike prices, we would still have only about 500,000 options prices on which to train a model to hedge this stock's options.

For these reasons, we did not use historical market data in this project.

Nevertheless, the use of historical data remains a pressing issue and warrants further inquiry. The present author knows of two plausible approaches to miti-

gate the effects of the relative paucity of historical options pricing data compared to simulated data:

Augmenting with GANs The work of [Goodfellow et al., 2014] was revolutionary in the field of deep learning for proposing a mechanism by which a pair of neural networks, working as "adversaries" to one another, could be used to estimate generative models. In particular, subsequent applied work focused on using the GAN (generative adversarial network) technique to generate image data and sequence data (e.g. text) that is in some intuitive sense similar to a given input dataset, but is distinct from the specific examples in that set. It is possible that a similar technique could work for augmenting historical market data with additional data points that are similar to, and yet not identical copies of, some base set of market data. In this way, the aforementioned problem of having too little historical data could be overcome.

Transfer Learning Following the work of [Yosinski et al., 2014], it may also be possible to train a RL agent to hedge options using simulated data, and then "fine–tune" the model on historical data. The idea here is that the initial training phase (on simulated data) would get the RL agent's deep neural network weights somewhere close to their optimal values, and then the fine–tuning phase serves to optimize the weights to deal with real–world data. This technique likely will only work if the simulated data is sufficiently close to historical market data; otherwise, the weights learned in the initial training phase will not be "close enough" for the relatively small quantity of historical market data to bring them to optimality.

4 Problem Formulation

As mentioned in Section 1, the reinforcement learning framework requires us to define three spaces which determine: (1) what input data the agent can act upon, (2) what actions the agent can take, and (3) what measure of its performance the agent receives from the environment. These are, respectively, the *observation space*, the *action space*, and the *reward space*.

4.1 Observation Space

For this project, following the work of [Du et al., 2020], we give the RL agent access to only the prices of the various assets (specifically, cash, the underlying stock, and the option on that stock) as observations. Per the requirements of the OpenAI `gym` environment framework, these prices are given to the agent as a vector which is then fed in to the agent's deep neural network.

In addition to the three prices, we can also consider augmenting the observation space to give the agent access to additional data, provided that it does not cause lookahead bias. In particular, the agent could receive a vector of its current

holdings of the various assets; any fixed parameters relevant to the various assets (e.g., the strike price and time until expiry of any options contracts); as well as the actual delta of the options contract that it is trying to hedge. Further work should focus on these extensions to the observation space.

4.2 Action Space

The actions available to the agent are buying and selling units of the underlying stock: for the purposes of this project, we assume that the agent cannot buy and sell cash directly, nor can it trade the option (as we assume our holdings of the options contract are fixed). Nonetheless, there is an important choice in the specification of the action space, namely whether to make it discrete or continuous.

On the one hand, the ultimate actions that take place in the market are in an important sense discrete: it is only possible to buy or sell integer numbers of shares.

On the other hand, continuous action spaces have the advantage that they encode the "smoothness" between actions that are close to each other in the space. Intuitively, buying 50 shares and buying 51 shares should have similar effects, whereas selling 100 shares should have vastly different effects. With a continuous action space, the agent gets this property for free; with a discrete action space, all actions are *a priori* equally "different", and any similarities have to be learned during training, as opposed to being automatically encoded in the linear algebra underlying the neural network.

American options present an additional consideration for how to encode the action space; this is discussed in Section 6.1

4.3 Reward Space

The reward space is the easiest of the three spaces to specify, because it is simply a single real number: as laid out in [Du et al., 2020], the agent's reward at time t is given by

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

where δw_t denotes the wealth increment from time $t-1$ to time t and κ is a parameter (set equal to 0.1 for this project).

The parameter κ governs how much we penalize the agent for having a high variance of wealth (which is bad, because being well–hedged implies that the agent's total wealth should be as close as possible to constant over time) relative to how much we reward the agent for earning high wealth in a given period (i.e., having high δw_t).

In addition, we can clip the reward, as suggested by [Mnih et al., 2015], so that the reward experienced by the agent never exceeds certain bounds. Clipping the reward is desirable because it limits the magnitudes of the error gradients (which are backpropagated through the network during training), and reduces the sensitivity of the agent to noise in the training data.

5 Hedging European Options

The first significant research goal of this project is to replicate the results of [Du et al., 2020] using the system that we have implemented. After non–trivial work on optimizing the hyperparameters of the PPO algorithm, approximately similar results were achieved, although there is still work to be done in terms of matching the exact performance of [Du et al., 2020].

5.1 Results

The following results were achieved using the choices and parameter settings shown in Table 1.

Setting	Value
Agent initial holdings	0 shares; \$10,000,000 cash; 100 call options ($K = 100$, $T = 1$)
r (risk–free rate)	0
σ (annual stock volatility)	0.07
μ (annual stock return)	0.2
Action space	Discrete: any integer in $[-100, 100]$
Observation space	Prices of stock, cash, and call option
Market impact (price adjustment)	None
γ (PPO discount factor)	0.8
λ (GAE lambda)	0.7
c_2 (PPO loss entropy bonus coefficient)	0.15
c_1 (PPO loss VF coefficient)	0.5
PPO maximum gradient norm	0.5
Reward clipping	Reward clipped to $(-50, 20)$

Table 1. Hyperparameter values and other settings for European options hedging

Some of the hyperparameters in the aforementioned table warrant further explanation. As stated in [Du et al., 2020], the loss function that is optimized by PPO (proximal policy optimization, the specific deep RL algorithm used for this portion of the project) is as follows:

$$L_t^{\mathrm{PPO}}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[L_t^{\mathrm{CLIP}}(\boldsymbol{\theta}) - c_1 L^{\mathrm{VF}}(\boldsymbol{\theta}) + c_2 S\left[\boldsymbol{\pi}_{\boldsymbol{\theta}}\right](s_t) \right],$$

where L^{VF} is the squared–error between the realized and predicted value functions, $S\left[\boldsymbol{\pi}_{\boldsymbol{\theta}}\right]$ denotes an entropy bonus, and $L_t^{\text{CLIP}}\left(\boldsymbol{\theta}\right)$ is a surrogate loss function. This surrogate loss function requires estimation of the advantage function at each time step during training, and the estimation procedure in turn relies on the λ parameter mentioned in Table 1.

The c_1 and c_2 parameters govern, respectively, the strength of the value function contribution to the overall loss, and the strength of the entropy bonus contribution. In particular, the higher c_2, the more the agent will "explore" rather than "exploit" during training.

Finally, note that the aforementioned loss function L_t^{PPO} is meant to be *maximized*. However, for consistency with established practice in the machine learning and optimization literature, the `stable-baselines3` library uses $-L_t^{\text{PPO}}$ as the loss function, which it then *minimizes*, as can be seen in the following figures.

Figure 2 shows the training loss (i.e., the value of $-L_t^{\text{PPO}}$) as a function of time during training. As can be seen, loss decreases steadily over time, although there are periodic spikes in the loss function. These spikes roughly correspond to times when the environment switched to using different pricing data to generate the observations sent to the agent. We switched to new market data every 15,000 episodes (in other words, the agent sees the same market data 15,000 times before it sees new market data in training). However, as training progresses, the agent becomes less overfitted to any particular set of market data observations—in other words, Figure 2 demonstrates that the agent successfully learns to generalize to new market data.

Figure 3 shows the value of the entropy bonus as as function of time during training. As can be seen, the entropy bonus starts negative (thereby reducing the value of the loss function $-L_t^{\text{PPO}}$ which the algorithm is trying to minimize). This corresponds to the agent initially being primed to "explore" the action space, rather than "exploit". As time goes on, however, the entropy bonus fluctuates towards and away from 0, indicating that at different stages of training, it learns to "explore" more or less. Again, the troughs of the training entropy bonus (corresponding to maximal desire to explore) correspond roughly to times when the training data was changed, once every 15,000 episodes.

The fact that the entropy bonus does not converge towards 0 over time indicates that the agent has perhaps not explored enough of the action space; in other words, it should be trained for longer in order to reach optimality.

Figure 4 shows the value of the value loss function L^{VF} over time during training. Recall that this is a squared–error loss function that penalizes differences between the realized and predicted value functions. As can be seen in the graph, the agent becomes significantly better at estimating the value function over time.

Finally, we evaluate the agent out of sample on market data that it did not see in training. Figure 5 shows the agent's stock holdings (orange) as compared to the optimal delta hedge (blue). Clearly the performance is not as good as that of the agent trained in [Du et al., 2020], though there are promising signs: the agent has (mostly) learned that its position should always be negative, which is indeed the case when hedging a long position in a European call option. Moreover, the agent appears to have learned the direction of the delta hedge, even as it overshoots the magnitude. Finally, as mentioned previously, the agent would likely benefit from additional training, as the entropy loss did not converge to 0 in Figure 3.

Fig. 2. Training loss vs. time (European options hedging)

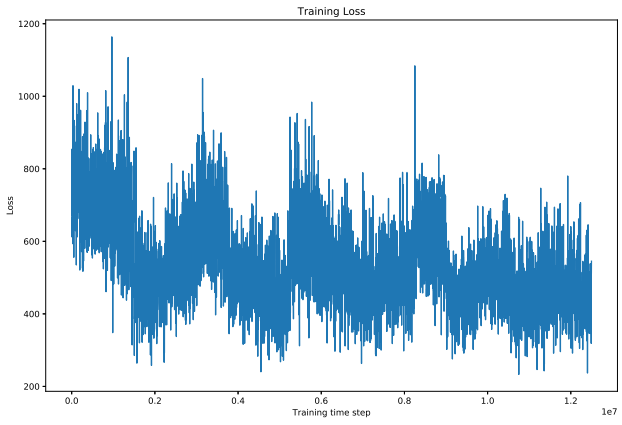


Fig. 3. Training entropy bonus vs. time (European options hedging)

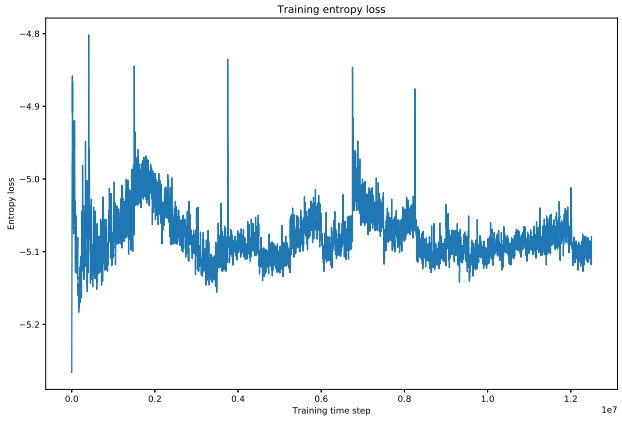


Fig. 4. Training value function loss vs. time (European options hedging)

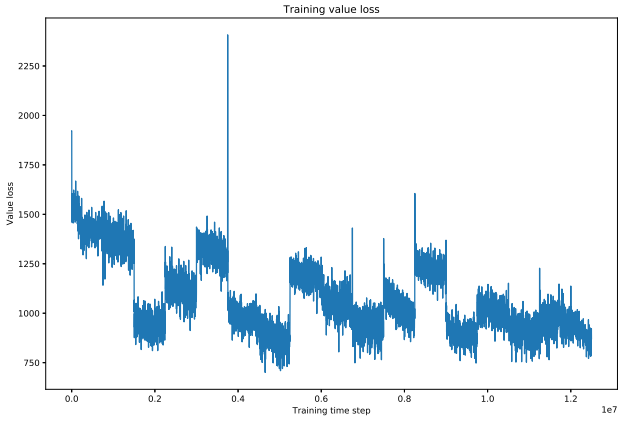
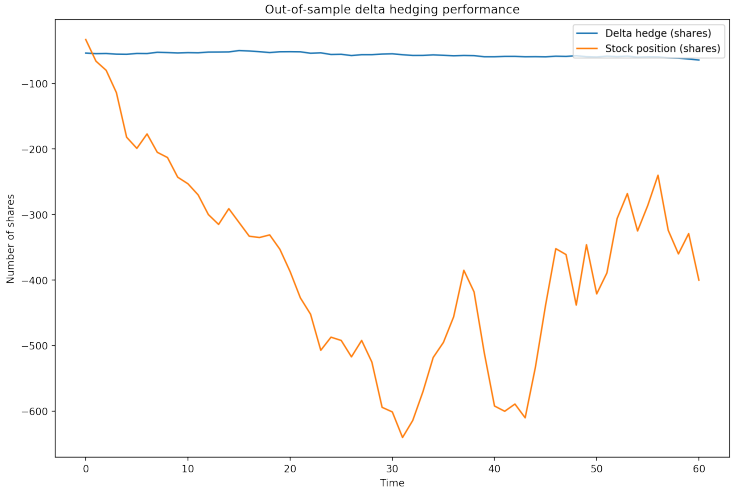


Fig. 5. Out–of–sample delta–hedging performance vs. time (European options hedging)



6 Hedging American Options

The second main goal of this project is to apply the techniques developed by [Du et al., 2020] to the problem of hedging American options. However, American options have certain features that make this problem more challenging than the European case.

6.1 Handling Early Exercise

The defining characteristic of an American option (as opposed to a European option) is the fact that it can be exercised at any time prior to expiry, or upon expiry itself. The fact that the holder of an American option has this additional action available at all times presents an interesting challenge in terms of setting up the reinforcement learning problem in the presence of an American option.

There are a few different ways in which we can encode the early exercise feature into our formulation of the reinforcement learning problem.

Mandatory Early Termination of Episode The OpenAI `gym` library allows for the environment to terminate a training episode at any point in time, based on the state of the environment or actions taken by the agent. In the example of a deep RL agent learning to play a video game, an episode might terminate in the event that the game enters a winning or losing state, irrespective of how many time steps have elapsed. In our context, episodes typically last for a fixed period of time (corresponding to a fixed number of trading days), but we can

always terminate the episode early.

Early termination of an episode provides one way to handle the early exercise feature of American options: in particular, if it ever becomes optimal to exercise the option (where optimality is determined by some model, e.g. the Andersen model or a backwards dynamic programming model), we can terminate the episode and give the agent its final reward based on whatever wealth was earned from exercising the option.

The advantage of this approach is that it is very simple to implement: the action space available to an agent is the same, whether it is hedging an American option or a European option. Indeed, without making any changes to the observation space, the agent has no explicit information about whether it is trading an American or a European option: it only sees prices!

One disadvantage of this approach is that the exercise boundary has to be determined exogenously: the agent never learns when to exercise the option early. Arguably, this is a good thing, as the job of the agent is simply to *hedge* the option, which should be a fixed component of its portfolio (albeit one whose value varies over time). Nevertheless, it would be interesting to see whether an agent can profitably learn to perform both actions (hedging and exercise) simultaneously.

Another, more subtle disadvantage of this approach is that the agent may never learn the concept of early exercise at all. The agent might simply learn that the length of an episode is unpredictable, and that it could end at any time; if this is the case, then the agent might incorrectly attempt to maximize short–term rewards (although to an extent this is mitigated by the value of its discount factor parameter, γ). Alternatively, it could be the case that the agent learns (incorrectly) that early exercise always happens around some fixed time step: if that is the case, then the agent could learn to behave differently near that time (e.g. dumping its holdings of the stock as the fixed time step approaches), irrespective of early exercise.

Mandatory Exercise without Termination A slightly different approach to handling early exercise is to force the option to be exercised (i.e., force the agent's holdings of the option to go to zero) as above, but *not* to terminate the episode immediately.

Instead, the agent could receive the proceeds of exercising the option (either as cash or stock) and then continue performing actions until the normal end of the episode.

Because the agent's holdings of the option will be 0 after exercise, the option can no longer act as a hedge against the agent's stock positions; that is, if the

agent behaves exactly as it did before, it will be penalized because the variance of its wealth will be much higher in the absence of the option.

As with the case of mandatory early termination of the episode, the agent in this scenario will not learn whether or not to exercise the option: that decision will be made exogenously. Moreover, it may again inadvertently learn that early exercise happens at a fixed time step (rather than when the stock price hits a certain exogenous boundary), which could lead it to behave improperly if early exercise occurs at a different time than the agent expects.

Agent–Controlled Exercise Finally, we can consider letting the agent learn when to exercise the option by itself. This involves augmenting the agent's action space to give it an additional action, namely whether or not to exercise the option at any given point in time, in addition to any trades of the stock that the agent may want to perform.

One complication of this approach is that the OpenAI `gym` library does not permit an action space that is the product of a continuous space and a discrete space. So, if the agent's space of possible trades has been implemented as a continuous action space, it is nontrivial to give it an additional (discrete) action to exercise.

A workaround for this is to define the agent's "exercise" action to be continuous, say in the range $[0, 1]$, if the trading action space is also continuous. Then, the environment rounds the agent's action value to the nearest integer and exercises the option if and only if the rounded value is 1. The disadvantage is that it may take much longer for the agent to learn how to use this supposedly–continuous action which in fact has a discrete boundary at 0.5 than it would if it had access to a truly discrete action.

6.2 Dealing with Market Impact

The final complication caused by American options is the fact that the market impact model shifts the price of the underlying stock, depending on the trading activity of the agent, which can interfere with the determination of the exercise boundary (in the event that we use mandatory exercise of the option, as opposed to agent–controlled exercise).

In particular, the agent could incorrectly learn to (e.g.) drive down the price of a stock by selling it in large quantities. If the price of the stock goes below some threshold price, mandatory exercise of an American put option will be triggered. Then, the agent could simply liquidate its holdings and achieve 0 variance of wealth for the remaining duration of the episode (assuming that the episode was not terminated upon exercise of the option).

6.3 Results

Table 2 shows the various choices that were made regarding hyperparameter settings and other aspects of the agent and environment.

Setting	Value
Agent initial holdings	-50 shares; \$10,000,000 cash; 100 put options ($K = 100$, $T = 1$)
r (risk–free rate)	0
σ (annual stock volatility)	0.07
μ (annual stock return)	0.2
Action space	Discrete: any integer in $[-100, 100]$
Observation space	Prices of stock, cash, and put option
Market impact (price adjustment)	None
γ (PPO discount factor)	0.827
λ (GAE lambda)	0.949
c_2 (PPO loss entropy bonus coefficient)	0.193
c_1 (PPO loss VF coefficient)	0.5
PPO maximum gradient norm	0.5
Reward clipping	Reward clipped to $(-50, 20)$
Early exercise mechanism	Mandatory early termination

Table 2. Hyperparameter values and other settings for American options hedging

Figure 6 shows the training loss of our American options hedging agent. The performance of this agent in training is impressive, especially compared to the performance shown previously in the corresponding visualization for the European options hedging problem. In particular, we see that the agent's loss converges to almost 0 about half way through the training process.

Figure 7 demonstrates the entropy bonus experienced by the American options hedging agent. The entropy bonus indicates the extent of the exploration—exploitation tradeoff; the characteristic shape of this curve (starting at a negative value, indicating a preference for exploration, before going up to 0 near the end of training) indicates that the agent has come close to some local optimum and no longer needs to explore the action space. Instead, it can focus on simply exploiting the actions that it has already learned will lead to high rewards.

Figure 8 shows that, in addition to having achieved low overall loss, the American options hedging agent has in fact achieved low value function loss as well. That is, the agent has successfully learned to predict the value function (i.e. the sequence of rewards that is likely to result from taking the action prescribed by an agent's policy in a given state).

The aforementioned three figures suggest that the American options hedging agent has been trained to an almost–optimal level. Unfortunately, the performance of the agent on an out–of–sample test set, as shown in Figure 9, shows that the actual performance of the agent on unseen data leaves much to be desired. Future work should focus on understanding why the agent apparently performed so well in training, while learning such a policy that clearly does not generalize to new data very well.

Fig. 6. Training loss vs. time (American options hedging)

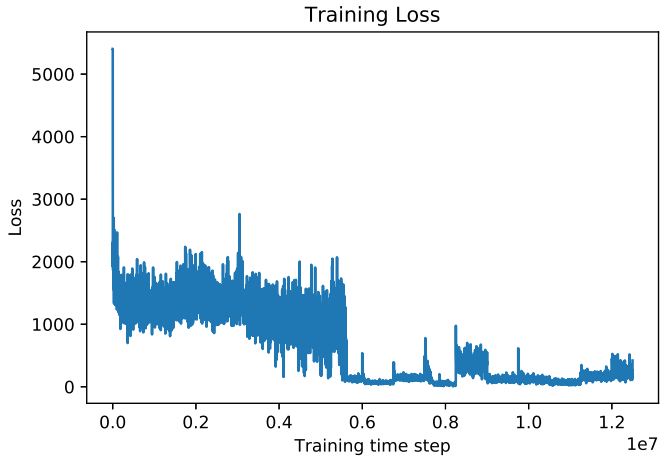


Fig. 7. Training entropy bonus vs. time (American options hedging)

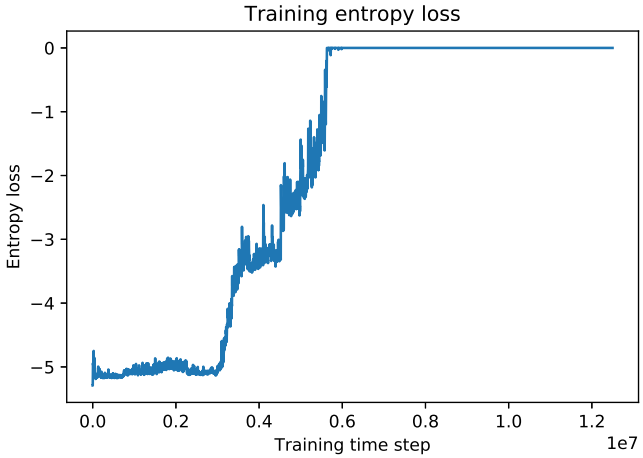


Fig. 8. Training value function loss vs. time (American options hedging)

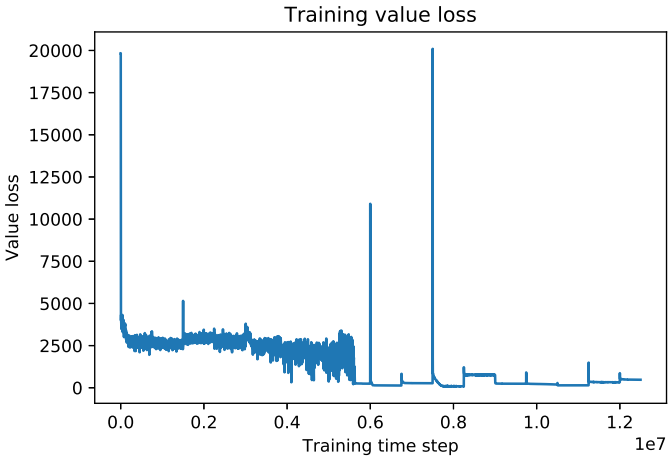
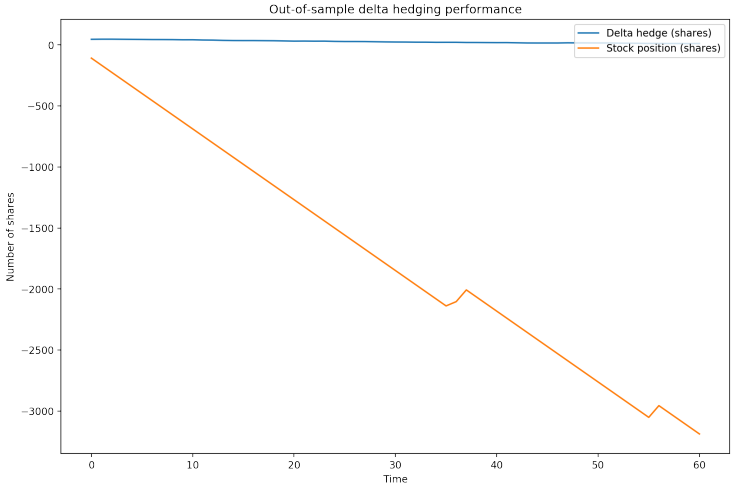


Fig. 9. Out–of–sample delta–hedging performance vs. time (American options hedging)



7 Conclusion

We conclude that the theoretical basis for RL successfully solving the problem of hedging European and American options has merit: the reinforcement learners that we trained and evaluated seem do a reasonably good job of minimizing their loss in training, and where that is not the case, the trend of the training loss over time suggests that additional training samples would alleviate the issue.

Nonetheless, it seems that there are significant gaps between the performance of these reinforcement learning agents and the optimal delta hedging strategy in out–of–sample trials. In general, the reinforcement learning agents tend to over–trade and take positions that are too large to constitute an optimal delta hedge. Plausibly, this could be corrected by the imposition of trading costs, which would serve to further penalize the agents for taking large positions beyond the penalty already imposed for being suboptimally hedged. Moreover, in the case of American options, the underperformance in out–of–sample test cases could be due to the agents not properly taking into consideration the effects of the early exercise feature of the contract, as discussed in Section 6.1.

Although the practical results of this study, in terms of out–of–sample P&L on simulated data, do not quite measure up to the expectations set by prior authors such as [Du et al., 2020], our results do demonstrate that RL is a promising technique in this area of applied finance.

Finally, this study did not address the all–important matter of real, historical market data. In order for our theoretical results to yield real–world profitability, future work must apply these reinforcement learning techniques to actual stock and options pricing data collected from the markets.

References

[Du et al., 2020] Du, J., Jin, M., Kolm, P. N., Ritter, G., Wang, Y., and Zhang, B. (2020). Deep reinforcement learning for option replication and hedging. *The Journal of Financial Data Science Fall 2020*, 2(4):44–57.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA. MIT Press.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.