Dynamic Replication and Hedging: *A Reinforcement Learning Approach*

PETTER N. KOLM AND GORDON RITTER

PETTER N. KOLM
is a clinical professor and director of the Mathematics in Finance Master's Program at NYU's Courant Institute of Mathematical Sciences in New York, NY.
petter.kolm@nyu.edu

GORDON RITTER
is adjunct professor at NYU's Courant Institute of Mathematical Sciences, NYU's Tandon School of Engineering, and the Department of Mathematics of Baruch College, and is a Professor of Practice in Rutgers University's Department of Statistics in New York, NY.
wgr2@nyu.edu

The problem of replicating and hedging an option position is fundamental in finance. Since the publication of the seminal work of Black and Scholes (1973) and Merton (1973) on option pricing and dynamic hedging (jointly referred to as BSM), a substantial number of articles have addressed the problem of optimal replication and hedging. The core idea of BSM is that in a complete and frictionless market there is a continuously rebalanced dynamic trading strategy in the stock and riskless security that perfectly replicates the option.

However, in practice, continuous trading of arbitrarily small amounts of stock is infinitely costly. Instead, the portfolio replicating the option is adjusted at discrete times to minimize trading costs. Consequently, perfect replication is impossible, and an optimal hedging strategy will depend on the desired trade-off between replication error and trading costs. In other words, the hedging strategy chosen by an agent depends on the agent's risk aversion.

Although a number of articles have considered discrete time hedging or transaction costs alone, Leland (1985) was first to address discrete hedging under transaction costs. His work was followed by others.[1] The majority of these studies treat proportionate transaction costs. More recently, several studies have considered option pricing and hedging subject to both permanent and temporary market impact in the spirit of Almgren and Chriss (1999), including Rogers and Singh (2010); Almgren and Li (2016); Bank, Soner, and Vob (2017); and Saito and Takahashi (2017).

In this article, we show how to build a system that can learn how to optimally hedge an option (or other derivative security) in a fully realistic setting. Our method applies to the real-world engineering problem faced daily by trading and risk management desks at investment banks. In such situations, continuous-time theory is only a guide. Portfolio rebalance decisions must be made in discrete time and in markets with frictions, in which liquidity is not guaranteed and the market impact of the hedge could be substantial if not managed carefully. Almgren and Chriss (1999) showed that executing a large trade in a single stock is a multiperiod planning problem that can be solved by mean–variance optimization. The option hedging problem is similar but more complex. 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 (1) all forms of cost and (2) the deviation from the optimal hedge.

---

[1] See, for example, Figlewski (1989), Boyle and Vorst (1992), Henrotte (1993), Grannan and Swindle (1996), Toft (1996), Whalley and Wilmott (1997), and Martellini (2000).

This article contributes to the literature in several ways. First, our method is quite general. In particular, given any derivative security that we know how to price (even if that pricing is done by Monte Carlo), our method will quickly produce an autonomous agent who knows how to optimally trade off trading costs versus hedging variance for that security. The relative importance of cost versus variance is determined by the agent's risk-aversion parameter.

Second, our method is based on reinforcement learning (RL). Although RL is well known in its own right, to the best of our knowledge this form of machine learning technique has previously not been applied to discrete replication and hedging subject to nonlinear transaction costs. It is worthwhile to note that with the flexibility of the technique presented in this article, it is a straightforward process to extend the model with additional features and constraints such as round-lotting and position-level constraints. Although Halperin (2017) applied RL to options, the methods therein appear very specific to the BSM model, whereas our method allows the user to plug in any option pricing and simulation library and then train the system with no further modifications. Note also that Halperin (2017) did not consider transaction costs. Our article is also related to work by Buehler et al. (2018), who evaluated neural network–based hedging under convex risk measures subject to proportional transaction costs.

Third, our method is based on a continuous state space, and the training neither uses finite-state-space methods nor does it use or require a (necessarily arbitrary) selection of basis functions (as semigradient methods from Sutton and Barto (2018) would). Rather, we introduce a training method that has not been applied to derivatives hedging problems previously. Our training method relies on applying nonlinear regression techniques to the *sarsa targets* (Equation 6) derived from the Bellman equation.

Methods that require finite state spaces fail for larger problems, due to the curse of dimensionality. The state vector must contain all variables that are relevant to making a decision. For example, suppose there are k such variables, and each variable is allowed to have 10 possible values. The resulting state space has 10^k elements. Of course, this leads to insurmountable problems, such as (1) the fact that the training process can never visit most of the states; (2) there is no guarantee that the

value function will be continuous, let alone smooth; (3) a vector containing all such states cannot fit in computer memory; and (4) one must estimate millions of independent parameters from relatively fewer data points. By using a continuous state space, we avoid the curse of dimensionality and are thereby able to apply our method to higher-dimensional problems.

Fourth, the method 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.

REINFORCEMENT LEARNING

RL[2] has been developed largely independently from classical utility theory in finance. It provides a way to train artificial agents that learn through positive reinforcement 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.

Mathematically speaking, RL is a way to solve multiperiod optimal control problems. The agent's policy typically consists of explicitly maximizing the action-value function for the current state. This value function is an approximation of the true value function of the multiperiod optimal control problem. *Training* refers to the process of improving on the approximation of the value functions as more training examples are made available.

Following the notation of Sutton and Barto (2018), the sequence of rewards received after time step

---

[2]See Sutton and Barto (2018) and Kaelbling, Littman, and Moore (1996) for an introduction to RL.

t is denoted R_{t+1}, R_{t+2}, R_{t+3}, …. The agent's goal is to maximize the expected cumulative reward, denoted by

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \qquad (1)$$

The agent then searches for policies that maximize $\mathbb{E}[G_t]$. The sum in Equation 1 can be either finite or infinite. The constant $\gamma \in [0, 1]$ is known as the discount rate. If rewards are bounded, then $\gamma < 1$ ensures convergence of the infinite sum.

A policy, denoted π, is a way of choosing an action a_t, conditional on the current state s_t. A policy is allowed to be stochastic. For example, choosing a random action is also a policy.

There are principally two kinds of value functions; at optimality, one is a maximization of the other. The action-value function expresses the value of starting in state s, taking an arbitrary action a, and then following policy π thereafter

$$q_\pi(s, a) := \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \qquad (2)$$

where $\mathbb{E}_\pi$ denotes the expectation under the assumption that policy π is followed. The state-value function is the action-value function, where the first action also comes from the policy π

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = q_\pi(s, \pi(s))$$

Action-value functions are, for most practical purposes, more useful than state-value functions because any action-value function immediately gives rise to a natural policy: If $\hat{q}$ is any action-value function, the $\hat{q}$-greedy policy is to choose the action a, in state s, that maximizes $\hat{q}(s, a)$.

Policy π is defined to be at least as good as π' if $v_\pi(s) \geq v_{\pi'}(s)$ for all states s. An optimal policy is defined to be one that is at least as good as any other policy. There need not be a unique optimal policy, but all optimal policies share the same optimal state-value function $v_*(s) = \sup_\pi v_\pi(s)$ and optimal action-value function $q_*(s, a) = \sup_\pi q_\pi(s, a)$. Note also that v_* is the supremum over a of q_*. In particular, $v_*(s)$ is the expected gain (under any optimal policy), given that one started from state s. Colloquially, one might then refer to $v_*(s)$ as the value of being in state s.

The search for an optimal policy reduces to the search for the optimal action-value function q_* because

the q_*-greedy policy is optimal. The typical way of searching for q_* is to produce a sequence of iterates that approximates q_* with increasing accuracy. Methods for producing those iterates are based on the Bellman equations, which we now recall.

Let $p(s', r \mid s, a)$ denote the probability that the process transitions to state s' and the agent receives reward r, conditional on the event that the process was previously in state s, and in that state, the agent choses action a. The optimal state-value function and action-value function satisfy the Bellman equation

$$v_*(s) = \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_*(s')] \qquad (3)$$

$$q_*(s, a) = \sum_{s', r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \qquad (4)$$

where the sum over s', r, denotes a sum over all states s', and all rewards r.

The intuition for Equation 3 is that the value of being in state s equals the average, over all possible next states s', of the value of being in s' plus the reward associated with making the transition $s \rightarrow s'$. The intuitive interpretation for Equation 4 is very similar; indeed $\max_{a'} q_*(s', a') = v_*(s')$, so the bracketed quantities are the same in both equations.

The state-value function $v_*(s)$ has a natural interpretation in derivatives pricing theory. Specifically, in continuous time and frictionless markets, the optimal value function of the dynamic replicating strategy is obviously equal to the no–arbitrage price of the option. This is the value function that solves the Hamilton–Jacobi–Bellman partial differential equation, as shown by Merton and Samuelson (1992). Thus, it is natural that RL, in which value functions organize the search for optimal policies, should apply to pricing and, by extension, hedging of derivatives.

TRAINING VIA SIMULATION AND BATCH LEARNING

Although the state of the art is still evolving, the vast majority of the most successful applications of RL in recent years use a simulation of the environment to generate training data (as opposed to, say, training on historical data).

In a famous example from Mnih et al. (2013, 2015), a deep RL system learned to play video games on a superhuman level. According to the authors, the network was not provided with any game-specific information or hand-designed features and was not privy to the internal state of the emulator. It simply learned from nothing but the video input, the reward and terminal signals, and the set of possible actions.

In another famous example, Silver et al. (2017) created the best Go player in the world "based solely on RL, without human data, guidance, or domain knowledge beyond game rules." The associated system, termed AlphaGo Zero "is trained solely by self-play RL, starting from random play, without any supervision or use of human data."

In these cases (and many simpler ones—see Sutton and Barto (2018) for examples), the agents are trained in a simulated environment, as opposed to being trained on historical data. This has an advantage: Millions of training examples can be generated, limited only by computer hardware capabilities. The examples in the present article follow the same pattern: The system is trained by interacting with a simulator.

We now provide more details about how the training procedure works. We start with an estimate $\hat{q}$ of the optimal action-value function. This estimate is often initialized to be the zero function and is refined as the algorithm continues.

All RL systems must balance exploration and exploitation in the training process. They must sometimes take random actions to explore new areas of state space and action space—this is exploration. However, ultimately they must use their experience to concentrate the search around strategies that are likely to be optimal and refine the estimate of the value function on those areas of state space. We follow standard practice, which is to force exploration during training by using an ϵ-greedy policy relative to $\hat{q}$

$$\pi_{\epsilon\text{-greedy}}(s) = \begin{cases} \tilde{a} & u < \epsilon \\ \text{argmax}_a \hat{q}(s,a) & u \geq \epsilon \end{cases} \quad (5)$$

where ϵ is a real number between 0 and 1, u is a uniformly distributed random variable on $(0, 1)$, and $\tilde{a}$ is sampled uniformly from the action space. As is standard in RL and necessary to ensure convergence, we decrease the value of ϵ as training progresses.

Let s_t be the state at the t-th step in the simulation, and let $a_t = \pi_{\epsilon\text{-greedy}}(s)$ be the associated ϵ-greedy action. Let

$$X_t := (s_t, a_t)$$

be the resulting state-action pair. The update target Y_t is defined to be any valid approximation of $q_\pi(s_t, a_t)$. In this article we use the one-step sarsa target, which approximates $q_\pi(s_t, a_t)$ as follows

$$Y_t = r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}) \quad (6)$$

Intuitively, Equation 6 resembles part of the Bellman equation

$$q_*(s,a) = \sum_{s',r} p(s',r \mid s,a)[r + \gamma \max_{a'} q_*(s',a')] \quad (7)$$

Indeed, if $a_{t+1} = \text{argmax}_{a'} q_*(s',a')$, then Equation 6 would be a sample of the random variable in brackets in Equation 7. Thus, Equation 6 may be viewed as an approximation of $q_\pi(s_t, a_t)$.

We shall define a batch to be a collection of pairs of the form (X_t, Y_t) where $X_t := (s_t, a_t)$ is a state-action pair, and Y_t is the corresponding update target (Equation 6). A batch is typically obtained by running the simulator for the required number of time steps and choosing the actions via some policy π that is being evaluated.

Suppose we are going to run B different batches, indexed by $b = 1, \ldots, B$. We assume there is a nonlinear regression learner available that can learn a function of the form $Y = \hat{q}^{(b)}(X)$ using all of the samples in the batch. Suitable nonlinear regression learners are a topic of frequent study in the statistical learning literature (see Friedman, Hastie, and Tibshirani (2001) for an overview). They include random forests, Gaussian process regression, support vector regression, and artificial neural networks.

The fitted model $\hat{q}^{(b)}$ will then be used to improve the model current $\hat{q}$ by model averaging. We then generate batch $b + 1$, using the updated/improved $\hat{q}$ to calculate the Y_t, and repeat until we have B batches and $\hat{q}$ has been updated B times. Alternating between generation of batches and fitting models continues until some convergence criterion is reached. The simulations in this article used $B = 5$ batches each consisting of 750,000 (X, Y) pairs.

AUTOMATIC HEDGING IN THEORY

We define automatic hedging to be the practice of using trained RL agents to handle the hedging of certain derivative positions. The agent has a long option position that cannot be traded. The agent is only allowed to trade any other nonoption positions that would be used for replication. In a world with no trading frictions and where continuous trading is possible, there may be a dynamic replicating portfolio that hedges the option position perfectly, meaning that the overall portfolio (option minus replication) has zero variance. In our setting in this article, we will consider frictions and where only discrete trading is possible. Here the goal becomes minimization of variance and cost.

We will derive the precise form of the reward function, assuming our agent has a quadratic utility.[3] In particular, the agent's optimal portfolio is given by the solution to a mean–variance optimization problem with risk-aversion κ

$$\max\left(\mathbb{E}[w_T] - \frac{\kappa}{2}\ \mathbb{V}[w_T] \right) \tag{8}$$

where the final wealth w_T is the sum of individual wealth increments δw_t

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

and so $\mathbb{E}[w_T] = w_0 + \sum_t \mathbb{E}[\delta w_t]$. The variance term involves cross-covariances of the form $cov(\delta w_t, \delta w_s)$ for $s \neq t$, but if we are willing to assume independence of wealth increments across time, that is

$$cov(\delta w_t, \delta w_s) = 0 \text{ for } s \neq t$$

then $\mathbb{V}[w_T] = \sum_t \mathbb{V}[\delta w_t]$.[4]

In complete markets, options are redundant instruments. They can be exactly replicated (with zero variance) by a continuous-time dynamic trading strategy that trades infinitely often in infinitesimal increments. In the real world, the profit and loss (P&L) variance of

---

[3] See Ritter (2017) for a discussion of how the mean–variance assumption fits in within a general utility framework.

[4] The independence assumption will be violated in a number of interesting examples, such as assets with long-lived transient market impact.

an option minus its offsetting replicating portfolio is not zero. In the spirit of Almgren and Chriss (2001), our hedging agent would like to solve a simplified version of Equation 8, namely

$$\min_{\text{strategies}} \sum_{t=0}^{T}\left(\mathbb{E}[-\delta w_t] + \frac{\kappa}{2}\ \mathbb{V}[\delta w_t] \right) \tag{9}$$

where the minimum is computed across all permissible trading strategies. What is different in our work as compared to that of Almgren and Chriss (2001) is that a machine will learn the optimal strategy by simulating a financial market and applying RL to the simulation results.

If the log price process is a random walk, then wealth increments can be decomposed as

$$\delta w_t = q_t - c_t$$

where q_t is random walk term, and c_t is the total trading cost paid in period t (including commissions, bid–offer spread cost, market impact cost, and other sources of slippage). In the random walk case, the expected wealth increment is therefore just -1 times the expected cost

$$\mathbb{E}[-\delta w_t] = \mathbb{E}[c_t]$$

In other words, in this case the problem (Equation 9) becomes a trade-off of cost versus variance. The agent can hedge more frequently to reduce the variance of the hedged position, but at increased trading costs.

As shown by Ritter (2017), with an appropriate choice of the reward function, the problem of maximizing $\mathbb{E}[u(w_T)]$ can be recast as a RL problem. The reward in each period corresponding to Equation 9 is approximately

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

By plugging each one-period reward into the cumulative reward (Equation 1), we obtain an approximation of the mean–variance objective. Thus, training reinforcement learners with this kind of reward function amounts to training 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.

In the next section, we shall show that automatic hedging is indeed possible using RL training methods.

AUTOMATIC HEDGING IN PRACTICE

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.[5]

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 $100 \cdot L$ 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.

In any successful application of RL, the state must contain all of the information that is relevant for making the optimal decision. Information that is not relevant to the task at hand, or which can be derived directly from other variables of the state, does not need to be included. For European options, the state must minimally contain the current price S_t of the underlying and the time $\tau := T - t > 0$ still remaining to expiry, as well as our current position of n shares. The state is thus naturally an element of

$$\mathcal{S} := \mathbb{R}_+^2 \times \mathbb{Z} = \{(S, \tau, n) \mid S > 0, \tau > 0, n \in \mathbb{Z}\}$$

If the option is American, then it may be optimal to exercise early just before an ex-dividend date. In this situation, the state must be augmented with one additional variable: the size of the anticipated dividend in period $t + 1$.

The state does not need to contain the option Greeks because they are (nonlinear) functions of the variables the agent has access to via the state. We expect agents, given enough simulations, to learn such nonlinear functions on their own as needed. This has the advantage of not requiring any special, model-specific calculations that may not extend beyond BSM models.

Practitioners often compute the delta of an option position, for hedging purposes, using the BSM formula:

$$\Delta = \frac{\partial C}{\partial S} = N(d_1),$$

$$d_1 = \frac{\ln \frac{S_t}{K} + \frac{\tau \sigma^2}{2}}{\sigma \sqrt{\tau}},$$

$$\tau := T - t > 0 \qquad (11)$$

but with σ replaced by the implied volatility. This is referred to as *practitioner delta* by Hull and White [2017]. Note that parameters such as K and σ^2 are not provided to the agent, although they are used in constructing the simulation under which the agent is trained.

The agent will learn the properties of the stochastic world it inhabits by means of a large number of simulations of such world, as described. Nonlinear functions such as Δ as given by Equation 11, insofar as they affect the optimal strategy, will become part of the agent's learned action-value function (Equation 2).

We simulate a BSM world but modified to reflect the realities of trading: discrete time and space. We consider a stock whose price process is a geometric Brownian motion (GBM) with initial price S_0 and daily lognormal volatility of σ/day. We consider an initially at-the-money European call option (struck at $K = S_0$) with T days to maturity. We discretize time with D periods per day; hence each episode has $T \cdot D$ total periods. We require trades (hence also holdings) to be integer numbers of shares. We assume that our agent's job is to hedge one contract of this option. In the following specific examples, the parameters are $\sigma = 0.01$, $S_0 = 100$, $T = 10$, and $D = 5$. In addition, we set the risk aversion $\kappa = 0.1$.

We first consider a frictionless world without trading costs and answer the question of whether it is possible for a machine to learn what we teach students

---

[5] However, we note that this is possible on an extended state space.

E X H I B I T 1
Out-of-Sample Simulation of a Trained RL Agent
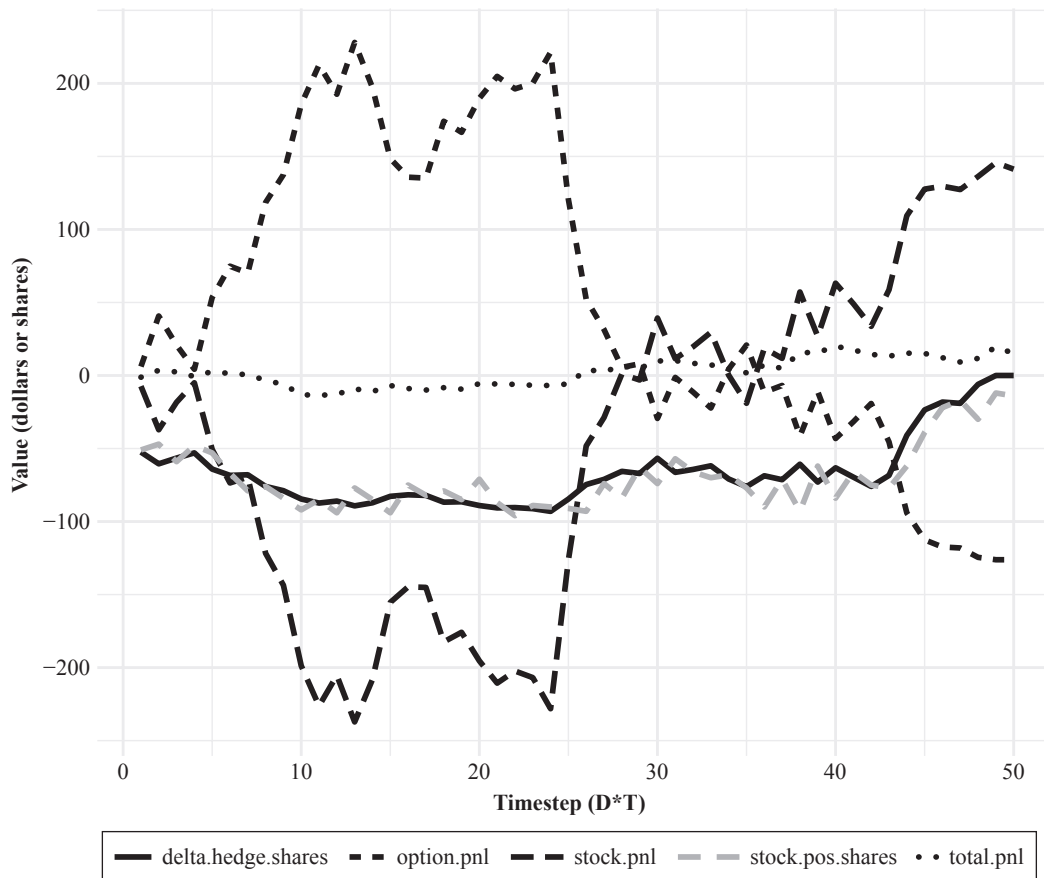
Notes: We depict cumulative stock, option, and total P&L; RL agent's position in shares (stock.pos.shares); and −100·Δ (delta.hedge.shares). Observe that (1) cumulative stock and options P&L roughly cancel one another to give the (relatively low variance) total P&L, and (2) the RL agent's position tracks the delta position even though it was not provided with it.

in their first semester of business school: formation of the dynamic replicating portfolio strategy. Unlike our students, the machine can only learn by observing and interacting with simulations.

The RL agent is at a disadvantage, initially. Recall that it does not know any of the following pertinent pieces of information: (1) the strike price K, (2) the fact that the stock price process is a GBM, (3) the volatility of the price process, (4) the BSM formula, (5) the payoff function $(S - K)_+$ at maturity, and (6) any of the Greeks. It must infer the relevant information from these variables, insofar as it affects the value function, by

interacting with a simulated environment.[6] Each out-of-sample simulation of the GBM is different, but we show a typical example of the trained agent's performance in Exhibit 1.

---

[6] One could try to help the algorithm by providing the BSM delta as part of the state variable, hence allowing the reinforcement learner to use that directly, but we deliberately chose not to include any of the option Greeks as state variables. Giving the system access to the option Greeks is sure to improve its performance because the function being learned is closer to linear. We chose not to do this to make the problem as hard as possible and to see if RL is up to the challenge. However, in a real-world production scenario, we recommend making the problem as easy as possible by including certain option Greeks in the state variable, unless they are prohibitively hard to calculate.

Exhibit 2
Out-of-Sample Simulation of a Baseline RL Agent That Uses Policy Delta or π_{DH}, Defined in Equation 12
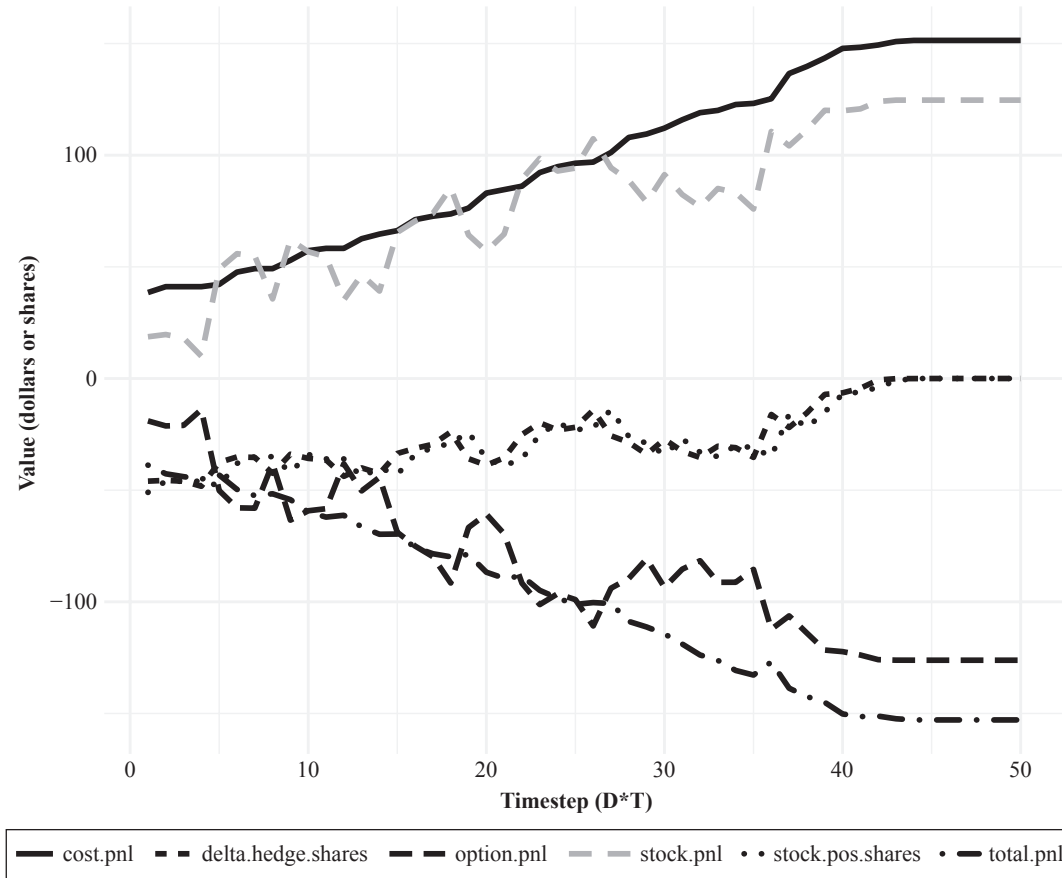
Notes: We show cumulative stock P&L and option P&L, which roughly cancel one another to give the (relatively low variance) total P&L. We show the position, in shares, of the agent (stock.pos.shares). The agent trades so that the position in the next period will be the quantity $-100\cdot\Delta$ rounded to shares.

Because the examples of Exhibit 1 were generated in a frictionless simulation, why is the total P&L not exactly zero? The answer is discretization error. Time is discretized (to five periods per day), so continuous hedging is not possible. Moreover, the simulation requires trading an integer number of shares, which introduces further discretization error.

Any complex model should be tested against a simpler model as a baseline. To justify its additional complexity, the more complex model should be able to do something that the simpler model cannot. Along these lines, let us define a simple policy, π_{DH}, as a baseline for the more complex policy learned by RL methods.

As in Equation 11, let $\Delta(p_t, \tau)$ denote the delta as computed from the price p_t at time t and the time-to-expiry $\tau = T - t$. The full state variable is then $s_t = (p_t, \tau, n_t)$,

where n_t denotes the agent's current holding, in shares, at time t. Our simple baseline policy must output an action, which is just a number of shares to trade, given this state vector. Define

$$\pi_{DH}(s_t) = \pi_{DH}(p_t, \tau, n_t) := -100 \cdot \text{round}(\Delta(p_t, \tau)) - n_t \quad (12)$$

where the round function returns the closest integer to the argument.

The policy π_{DH}, without rounding, is optimal in a hypothetical trading-cost-free world, where the number of time steps goes to infinity and where one can trade fractional numbers of shares. There is, however, no reason to expect that π_{DH} would solve the utility-maximization problem (Equation 9) in a simulation with

E X H I B I T   3
Out-of-Sample Simulation of Our Trained RL Agent
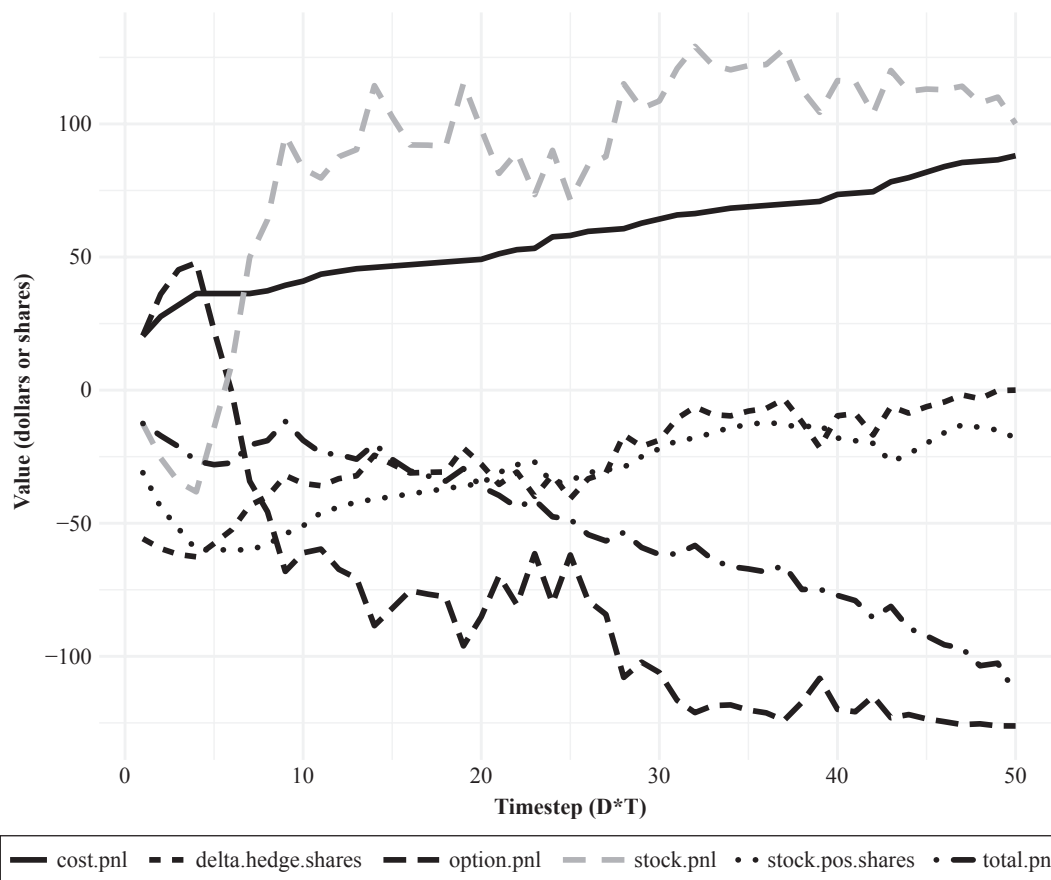
Note: The curve representing the agent's position (stock.pos.shares) controls trading costs and is hence much smoother than the value of −100·Δ (called delta.hedge.shares), which naturally fluctuates along with the GBM process.

nontrivial trading costs or, for that matter, in the real world (where we know trading costs are nontrivial).

For a trade size of *n* shares we define

$$\text{cost}(n) = \text{multiplier} \times \text{TickSize} \times (|n| + 0.01n^2) \quad (13)$$

where we take TickSize = 0.1. With multiplier = 1, the term TickSize × |*n*| represents a cost, relative to the midpoint, of crossing a bid–offer spread that is two ticks wide. The quadratic term in Equation 13 is a simplistic model for market impact. Exhibit 1 has multiplier = 0.

A key strength of the RL approach is that it does not make any assumptions about the form of the cost function (Equation 13). It will learn to optimize expected utility under whatever cost function is provided. In Exhibit 1, we had taken multiplier = 0 in

the function cost(*n*), representing no frictions. We now take multiplier = 5, representing a high level of friction. Our intuition is that in high-trading-cost environments (which would always be the case if the position being hedged were large relative to the typical volume in the market), the simple policy π_{DH} trades too much. One could perhaps save a great deal of cost in exchange for a slight increase in variance.

Given the mean–variance utility function in Equation 9, we expect RL to learn the trade-off between variance and cost. In other words, we expect it to realize lower cost than π_{DH}, possibly coming at the expense of higher variance, when averaged across a sufficiently large number of out-of-sample simulations (i.e., simulations that were not used during the training phase in any way).

E X H I B I T 4

Kernel Density Estimates for Total Cost (left panel) and Volatility of Total P&L (right panel) from N = 10,000 Out-of-Sample Simulations
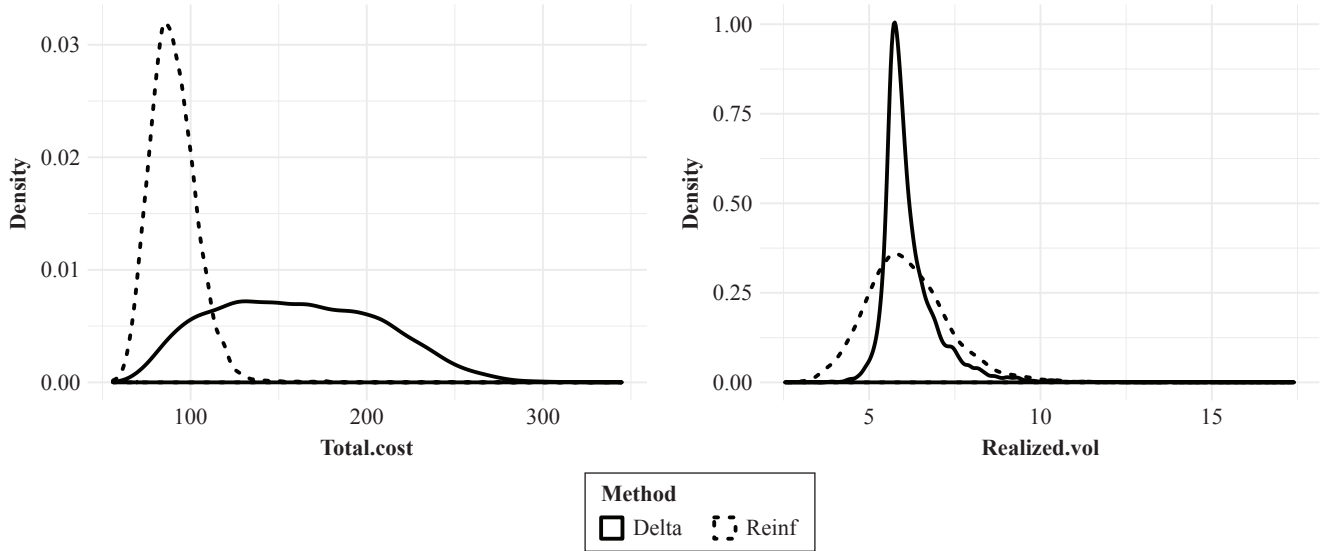
Notes: *Policy delta is* π_{DH}*, while policy reinf is the greedy policy of an action-value function trained by RL. The reinf policy achieves much lower cost (t-statistic = −143.22) with no significant difference in volatility of total P&L.*

We trained the agent using five batches with 15,000 episodes per batch, each episode having $D \cdot T = 50$ time steps, as before. This means that each call to the nonlinear regression learner involves 750,000 (X_t, Y_t) pairs. The training procedure took one hour on a single CPU. After training, we ran $N = 10,000$ out-of-sample simulations. Using the out-of-sample simulations, we ran a horse race between the baseline agent that uses just delta-hedging and ignores cost and the RL-trained agent that trades cost for realized volatility.

Exhibit 2 shows one representative out-of-sample path of the baseline agent. We see that the baseline agent is overtrading and paying too much cost. Exhibit 3 shows the RL agent on the same path. We see that, while maintaining a hedge, the agent is trading in a cost-conscious way. The curves in Exhibit 2, representing the agent's position (stock.pos.shares), are much smoother than the value of −100·Δ (called delta.hedge.shares in Exhibit 2), which naturally fluctuates along with the GBM process.

Exhibit 3 consists of only one representative run from an out-of-sample set of $N = 10,000$ paths. To summarize the results from all runs, we computed the total cost and standard deviation of total P&L of each path. Exhibit 4 shows kernel density estimates (basically,

smoothed histograms) of total costs and volatility of total P&L of all paths. In each case, we performed a Welch two-sample *t*-test to determine whether the difference in means was significant. The difference in average cost is highly statistically significant, with a *t*-statistic of −143.22. The difference in vols, on the other hand, was not statistically significant at the 99% level.

One can also gauge the efficacy of an automatic hedging model by how often the total P&L (including the hedge and all costs) is significantly less than zero. For both policies (delta and reinf), we computed the *t*-statistic of total P&L for each of our out-of-sample simulation runs and constructed kernel density estimates (see Exhibit 5). The reinf method is seen to outperform: Its *t*-statistic is much more often close to zero and insignificant.

CONCLUSIONS

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 of the following pertinent pieces of information:

EXHIBIT 5
Kernel Density Estimates of the t-Statistic of Total P&L for Each of Our Out-of-Sample Simulation Runs and for Both Policies Represented Previously (delta and reinf)
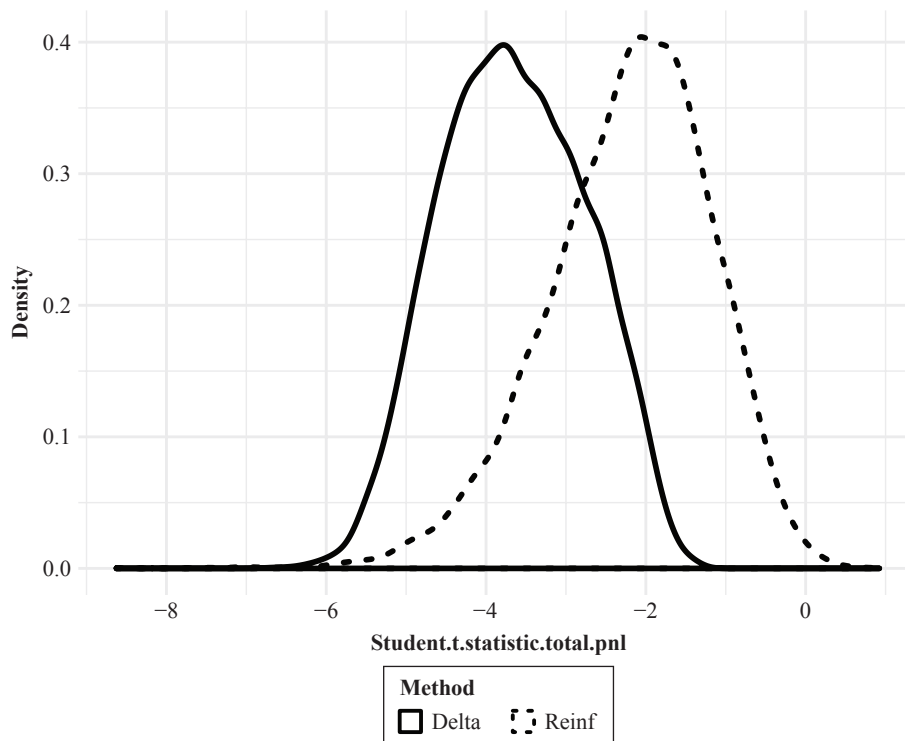
Note: The reinf method is seen to outperform in the sense that the t-statistic is much more often close to zero and insignificant.

(1) the strike price K, (2) the stock price process, (3) the volatility of the price process, (4) the BSM formula, (5) the payoff function $(S - K)_+$ at maturity, and (6) any of the Greeks. This is the financial derivatives analogue of the examples of Mnih et al. (2013) and Mnih et al. (2015), wherein computers learned to play games without knowing the rules.

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 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.

Our approach does not depend on the existence of perfect dynamic replication. It will learn to optimally

trade off variance and cost using whatever assets it is given as potential candidates for inclusion in a hedging portfolio. In other words, it will find the minimum-variance dynamic hedging strategy, whether or not the minimum variance is actually zero (as it typically is in derivatives pricing, where one needs perfect replication to derive a no-arbitrage price). This is important because, in many realistic cases, markets are not complete and hence some of the assets required for perfect replication may not exist.

Another advantage of this approach is that it can deal automatically with position-level constraints. It is part of the structure of any RL problem that, for each possible state s of the environment, the agent has a (potentially state-dependent) list of possible actions. In the examples given, the list of possible actions was taken to be buying or selling up to 100 shares in integer numbers of shares. We note that other trade or position constraints could be incorporated in a straightforward

way, simply by modifying the state-dependent list of available actions.

In this article, we leave open several avenues for further research. One obvious point of interest would be to train agents like ours on more sophisticated hardware and hence to take advantage of many more simulations and finer discretization of time. Silver et al. (2017) described various Go players that were trained on clusters with up to 176 GPUs and/or 48 TPUs, with training times ranging from 3 to 40 days. For reference, all of the examples in this article were trained on a single CPU, and the longest training time allowed was one hour.

Transaction costs are not static. The intraday term structure of trading volume has a well-known smile shape (documented by Chan, Christie, and Schultz 1995), with a nontrivial fraction of US equity trading volume occurring in the close and closing auction. Our RL system should handle this sort of complication very well. For instance, the simulator could be augmented with a nuanced cost function that depends on the time of day and add a discrete time-of-day indicator to the state vector.

Another interesting line of research would be to investigate optimal hedging strategies for portfolios of options in the presence of trading costs. Obviously, for low-gamma portfolios, delta-hedging would not be needed so frequently, thus naturally reducing the trading costs for that kind of portfolio. In general, the most cost-effective way to reduce variance is likely to use other options rather than a replicating portfolio of the underlier.

REFERENCES

Almgren, R., and N. Chriss. 1999. "Value under Liquidation." *Risk* 12 (12): 61–63.

———. 2001. "Optimal Execution of Portfolio Transactions." *Journal of Risk* 3: 5–40.

Almgren, R., and T. M. Li. 2016. "Option Hedging with Smooth Market Impact." *Market Microstructure and Liquidity* 2 (1): 1650002.

Bank, P., H. M. Soner, and M. Vob. 2017. "Hedging with Temporary Price Impact." *Mathematics and Financial Economics* 11 (2): 215–239.

Black, F., and M. Scholes. 1973. "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy* 81 (3): 637–654.

Boyle, P. P., and T. Vorst. 1992. "Option Replication in Discrete Time with Transaction Costs." *The Journal of Finance* 47 (1): 271–293.

Buehler, H., L. Gonon, J. Teichmann, and B. Wood. 2018. "Deep Hedging." *arXiv* 1802.03042.

Chan, K. C., W. G. Christie, and P. H. Schultz. 1995. "Market Structure and the Intraday Pattern of Bid–Ask Spreads for NASDAQ Securities." *The Journal of Business* 68 (1): 35–60.

Figlewski, S. 1989. "Options Arbitrage in Imperfect Markets." *The Journal of Finance* 44 (5): 1289–1311.

Friedman, J., T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning.* Berlin: Springer, 2001.

Grannan, E. R., and G. H. Swindle. 1996. "Minimizing Transaction Costs of Option Hedging Strategies." *Mathematical Finance* 6 (4): 341–364.

Halperin, I. 2017. "QLBS: Q-Learner in the Black–Scholes (–Merton) Worlds." *arXiv* 1712.04609.

Henrotte, P. "Transaction Costs and Duplication Strategies." Graduate School of Business, Stanford University, 1993.

Hull, J., and A. White. 2017. "Optimal Delta Hedging for Options." *Journal of Banking & Finance* 82: 180–190.

Kaelbling, L. P., M. L. Littman, and A. W. Moore. 1996. "Reinforcement Learning: A Survey." *Journal of Artificial Intelligence Research* 4: 237–285.

Leland, H. E. 1985. "Option Pricing and Replication with Transactions Costs." *The Journal of Finance* 40 (5): 1283–1301.

Martellini, L. 2000. "Efficient Option Replication in the Presence of Transactions Costs." *Review of Derivatives Research* 4 (2): 107–131.

Merton, R. C. 1973. "Theory of Rational Option Pricing." *The Bell Journal of Economics and Management Science* 4 (1): 141–183.

Merton, R. C., and P. A. Samuelson. *Continuous-Time Finance.* Boston: Blackwell Boston, 1992.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. "Playing Atari with Deep Reinforcement Learning." *arXiv* 1312.5602.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. 2015. "Human-Level Control through Deep Reinforcement Learning." *Nature* 518 (7540): 529.

Ritter, G. 2017. "Machine Learning for Trading." *Risk* 30 (10): 84–89.

Rogers, L. C. G., and S. Singh. 2010. "The Cost of Illiquidity and Its Effects on Hedging." *Mathematical Finance* 20 (4): 597–615.

Saito, T., and A. Takahashi. 2017. "Derivatives Pricing with Market Impact and Limit Order Book." *Automatica* 86: 154–165.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. 2017. "Mastering the Game of Go without Human Knowledge." *Nature* 550 (7676): 354–359.

Sutton, R. S., and A. G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge, MA: MIT Press, 2018 (in progress).

Toft, K. B. 1996. "On the Mean–Variance Tradeoff in Option Replication with Transactions Costs." *Journal of Financial and Quantitative Analysis* 31 (2): 233–263.

Whalley, A. E., and P. Wilmott. 1997. "An Asymptotic Analysis of an Optimal Hedging Model for Option Pricing with Transaction Costs." *Mathematical Finance* 7 (3): 307–324.

To order reprints of this article, please contact David Rowe at d.rowe@pageantmedia.com or 646-891-2157.