Machine Learning for Optimal Execution, Hedging, and Arbitrage

Gordon Ritter

Spring 2019

New York University	Courant Institute of Mathematics and Tandon School
Rutgers University	Department of Statistics and FSRM program
Baruch College	Department of Mathematics and MFE program
Ritter Alpha, LP	President, Chief Investment Officer

https://cims.nyu.edu/~ritter/
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3281235

This presentation is confidential and intended solely for attendees of The Courant Institute and Bloomberg conferences, and such persons' professional advisors. This presentation may not be reproduced or distributed. This presentation is for informational purposes only and does not constitute an offer to sell, or a solicitation of an offer to buy, any security or instrument in or to participate in any trading strategy with the presenter or in any fund or account advised by the presenter (each, a "fund"). If such offer is made, it will only be made by means of a confidential offering memorandum, which would contain material information (including certain risks of investment and conflicts of interest) not discussed in this presentation and which would supersede and which would qualify in its entirety the information discussed in this presentation. Any decision to invest in a fund should be made after reviewing the relevant confidential offering memorandum, conducting such investigations as the investor deems necessary and consulting the investor's own legal, accounting and tax advisors in order to make an independent determination of the suitability and consequences of an investment. No representation or warranty, express or implied, as to the accuracy or completeness of the information discussed in this presentation is made. It is believed to be reasonably accurate and current for the purposes of the illustrations provided but has not been independently verified. There can be no assurance that a fund's objectives will be achieved or that a fund will avoid substantial or complete losses. An investment in a fund is speculative and involves substantial risk of loss. An investment should only be considered by persons who can afford a loss of their entire investment. Past performance is not necessarily indicative of future results. Any statements herein regarding the viability or likelihood of success of trading techniques or approaches, future events or other similar statements constitute only subjective views, are based upon expectations or beliefs, should not be relied on, are subject to change due to a variety of factors and involve inherent risks and uncertainties, both general and specific, many of which cannot be predicted or quantified. Actual results could differ materially from those discussed, contemplated by or underlying the statements made during this presentation. In light of these risk and uncertainties, there can be no assurance that these statements are or will prove to be accurate or complete in any way.

The key idea of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies.

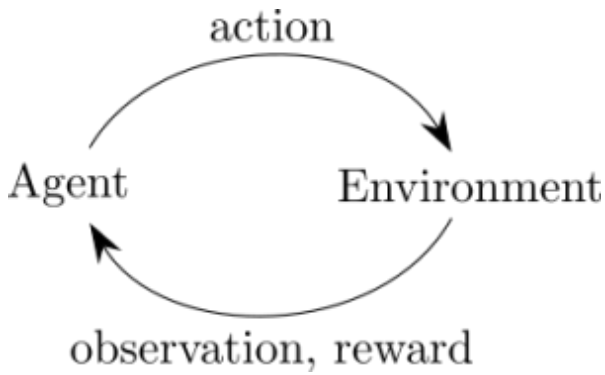
— Sutton and Barto (1998)

- The foundational treatise on value functions was written by Bellman (1957), at a time when the phrase "machine learning" was not in common usage.
- Nonetheless, modern machine learning owes its existence, in part, to Richard Bellman.
- The best modern treatise on the subject is Sutton and Barto (2018).

- A *value function* is a mathematical expectation in a certain probability space.
- The underlying probability measure is the one associated to a system which is very familiar to classically-trained statisticians: a Markov process.
- When the Markov process describes the state of a system, it is sometimes called a *state-space model*.
- When, on top of a Markov process you have the possibility of choosing a *decision* (or action) from a menu of available possibilities (the "action space"), with some reward metric that tells you how good your choices were, then it is called a *Markov decision process (MDP)*.

General Framework for Decision Making

- An interacting system: agent interacts with environment.
- The "environment" is the part of the system outside of the agent's *direct* control.
- At each time step t, the agent observes the current state of the environment $s_t \in \mathcal{S}$, and chooses an action $a_t \in \mathcal{A}$.
- This choice influences both the transition to the next state, as well as the reward the agent receives.



- The underlying distribution of a Markov Decision process can be written

$$p(s', r \mid s, a)$$

  for the joint probability of transitioning to state s' and receiving reward r, conditional on the previous state being s and the agent taking action a when in state s.
- This distribution is typically not known to the agent, but its existence gives mathematical meaning to notions such as "expected reward."

- A *policy* π is, roughly, an algorithm for choosing the next action, based on the state you are in.
- More formally, a policy is a mapping from states to probability distributions over the action space.
- If the agent is following policy π, then in state s, the agent will choose action a with probability $\pi(a \mid s)$.

- The *goal* G_t is the cumulative discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{1.1}$$

  where $\gamma \in (0, 1)$ is necessary for the infinite sum to be finite.
- Reinforcement learning is the search for policies which maximize the goal function, in expectation:

$$\max \mathbb{E}[G_t]$$

- There is an analogous version of the theory which applies to maximizing average reward, or reward per unit time.

- Assuming we start in state s, take action a, and then follow some fixed policy π from then on, what is out expected cumulative reward over time?
- The answer is the *action-value function*:

$$q_\pi(s, a) := \mathbb{E}[G_t] \text{ starting from } s, \text{ taking } a, \text{ then following } \pi$$

- The *state-value function* for policy π is defined to be

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

  where $\mathbb{E}_\pi$ denotes the expectation under the assumption that policy π is followed.
- The action value function is a more general concept than the state-value function, due to the obvious relation

$$v_\pi(s) = \sum_a \pi(a \,|\, s) q_\pi(s, a)$$

- The state-value function defines a partial ordering on policies.
- 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 which 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

$$v_*(s) = \max_\pi v_\pi(s)$$

and optimal action-value function

$$q_*(s, a) = \max_\pi q_\pi(s, a).$$

- Note that $v_*(s) = \max_a q_*(s, a)$, so the action-value function is more general than the state-value function. If we are willing to do some computation, we can also recover q_* from v_*:

$$q_*(s, a) = \mathbb{E}[r_{t+1} + \gamma v_*(s_{t+1}) \,|\, s_t = s, a_t = a]$$

- If we knew the q-function corresponding to the optimal policy, say q_*, we would know the optimal policy itself:

$$\text{choose } a \in \mathcal{A} \text{ to maximize } q_*(s_t, a)$$

  This is called following the *greedy policy*.
- Hence we can reduce the problem to finding q_*, or producing a sequence of iterates that converges to q_*.
- Anything that produces a sequence of functions converging to q_* using some form of "experience" or exposure to sequential (time-series) data, whether it be real or simulated data, is called a *reinforcement learning algorithm*.

- The optimal state-value function and action-value function satisfy the Bellman equations

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma\, v_*(s')] \tag{1.2}$$

$$q_*(s, a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{1.3}$$

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

- Quoting Sutton and Barto: "Many reinforcement learning methods can be clearly understood as approximately solving the Bellman optimality equation, using actual experienced transitions in place of knowledge of the expected transitions."
- The basic idea of several reinforcement learning algorithms is to associate the value on the right-hand side of the Bellman equation, i.e.

$$Y = r + \gamma \max_{a'} q_*(s', a')$$

  with the state-action pair $X = (s, a)$ that generated it. We will use this (X, Y) notation frequently in the sequel.
- The only problem is that we don't know $q_*(s', a')$ so we cannot actually calculate the Y-value or "update target" in the above equation.
- Perhaps we could use our current best guess of the function q_* to estimate the update target, or Y-value.

- The term dynamic programming (DP) refers to a collection of algorithms that can be used to explicitly find solutions of the Bellman equation and hence compute optimal policies given a perfect model of the environment as a Markov decision process (MDP).
- This relies on at least three assumptions that are rarely true in practice: (1) we accurately know the dynamics of the environment; (2) we have enough computational resources to complete the computation of the solution; and (3) the Markov property.
- For the kinds of tasks in which we are interested, one is generally not able to implement this solution exactly because various combinations of these assumptions are violated.
- For example, although the first and third assumptions present no problems for the game of backgammon, the second is a major impediment. Because the game has about 10^{20} states, it would take thousands of years to solve the Bellman equation for Backgammon.

- Imagine a scenario where we simulate the underlying Markov process, or perhaps even rely on the "simulation" that is known as the "real world." (The training process does not know or care whether it is being trained on real data or simulated data.)
- Performing the computations above, sequentially in our "simulation" leads to a sequence of (X, Y) pairs where, recall

$$X_t = (s_t, a_t)$$

is the state-action pair at the t-th step in the simulation, and

$$Y_t = r_t + \gamma \max_{a'} \hat{q}_t(s', a')$$

with $\hat{q}_t$ denoting the best approximation of q_* as it existed at the t-th time step.

- Then, recall the Bellman equation:

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

  which we can rewrite as $q^*(X) = \mathbb{E}[Y \mid X]$. The problem then shifts to learning the association between Y and X – a supervised learning problem.
- Reinforcement learning methods which focus on learning the association $Y = f(X) +$ noise with (X, Y) as above are generally referred to, for brevity, as "function approximation" by Sutton and Barto (2018). This function is typically very nonlinear; more on this later.

- It turns out that most of the problems financial engineers think about can be written in terms of value functions.
- Suppose you have a contingent claim, and suppose there is a dynamic replicating portfolio which gives the same payout as the contingent claim in every state of the world. Suppose policy π is to trade according to the dynamic replicating portfolio strategy. Then the no-arbitrage price of the option is

$$v_\pi(s_0)$$

  where s_0 is the state of the world today.
- Note that this applies to any state-contingent claim, and is not limited to Black–Scholes models or lognormal models.
- This is also true in the presence of transaction costs. No-arbitrage assumptions are often stated in a transaction-cost-free world, but reinforcement learning algorithms can produce dynamic hedging strategies which are optimal in a world with frictions. More on this later.

- Consider almost any optimal order execution problem (e.g. Almgren–Chriss). Almgren and Chriss say to execute an order of size X while optimizing the mean-variance form of expected utility. The value function is the expected integrated revenue and variance

$$v_\pi(s) = \mathbb{E}[\int_0^T (x_t r_t - \frac{\lambda}{2} x_t^2 \sigma_t^2 - f(\dot{x}_t))dt \,|\, x_0 = s]$$

  where $f(\dot{x}_t)$ is some function of the time-derivative $\dot{x}_t := dx_t/dt$ which approximates market impact.
- Many buy-side quant traders are also optimizing forms of mean-variance with trading costs, and are thus solving a problem conceptually similar to that of optimal order execution problems, but with the added complexity of needing return predictions.

- There is an obvious connection with games and with game theory. The player is the "agent" and then the "environment" consists of the other players or the simulated environment of the game. For many games including Go and some video games, the best player in the world is a reinforcement learning trained agent.

- In 1947, John von Neumann and Oskar Morgenstern proved that if a decision-maker
  (a) is faced with risky (probabilistic) outcomes of different choices, and
  (b) has preferences satisfying four axioms of "rational behavior"

  then that decision-maker will behave as if they are maximizing the expected value of some function, u, called the utility function, defined over the potential outcomes.
- When applied to trading of financial assets, the outcomes are typically various levels of wealth w_T at some future time T, and rational agents maximize

$$\mathbb{E}[u(w_T)]$$

  (not $\mathbb{E}[w_T]$, since this leads to paradoxes, and ignoring risk is ill-advised).

- In finance, an *optimal trading strategy* is usually defined as a strategy that optimizes expected utility of final wealth, where final wealth is the sum of a number of wealth increments over shorter time periods:

$$\text{maximize: } \mathbb{E}[u(w_T)] = \mathbb{E}[u(w_0 + \sum_{t=1}^{T} \delta w_t)] \tag{1.4}$$

  where $\delta w_t := w_t - w_{t-1}$.
- The utility function quantifies how the decision-maker values different outcomes, relative to one another. It is typically concave, increasing, and differentiable.

- Under certain assumptions on the return distribution, maximizing $\mathbb{E}[u(w_T)]$ is equivalent to maximizing a mean-variance form of that problem for any reasonable utility function.
- The main assumption on the return distribution is that its isoprobability contours exhibit elliptical symmetry, but note that this allows fat-tailed distributions.
- In other words, the validity of mean-variance optimization as a solution to an investor's expected-utility-maximization problem does not require us to assume normally-distributed returns.

- Under these assumptions, there exists some constant $\kappa > 0$, which depends on initial wealth w_0 and the investor's utility function, such that maximizing $\mathbb{E}[u(w_T)]$ is equivalent to:

$$\text{maximize}: \quad \left\{ \mathbb{E}[w_T] - \frac{\kappa}{2} \mathbb{V}[w_T] \right\} \tag{1.5}$$

- Suppose we could invent some definition of "reward" R_t so that

$$\mathbb{E}[w_T] - \frac{\kappa}{2} \mathbb{V}[w_T] \approx \sum_{t=1}^{T} R_t \tag{1.6}$$

  Then (1.5) looks like a "cumulative reward over time" problem.
- Among the various algorithms provided by reinforcement learning, a number of them seek to maximize $\mathbb{E}[G_t]$ where

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

  which by (1.6) would then maximize expected utility as long as $\gamma = 1 - \epsilon$ with ϵ small.

- Consider the reward function

$$R_t := \delta w_t - \frac{\kappa}{2}(\delta w_t - \hat{\mu})^2 \tag{1.7}$$

where $\hat{\mu}$ is an estimate of a parameter representing the mean wealth increment over one period, $\mu := \mathbb{E}[\delta w_t]$.

- Then

$$\frac{1}{T}\sum_{t=1}^{T} R_t = \underbrace{\frac{1}{T}\sum_{t=1}^{T} \delta w_t}_{\to \mathbb{E}[\delta w_t]} - \frac{\kappa}{2}\underbrace{\frac{1}{T}\sum_{t=1}^{T}(\delta w_t - \hat{\mu})^2}_{\to \mathbb{V}[\delta w_t]}$$

and for large T, the two terms on the right hand side approach the sample mean and the sample variance, respectively.

- Thus with this one special choice of the reward function (1.7), if the agent learns to maximize cumulative reward, it should also approximately maximize the mean-variance form of utility.

- If we're using the reward function $R_t := \delta w_t - \frac{\kappa}{2}(\delta w_t - \hat{\mu})^2$ then we have a preference for average reward rather than discounted reward as the goal G_t.
- According to Sutton and Barto (2018), In the average-reward setting, the quality of a policy π is defined as the average rate of reward while following that policy, which we denote as follows

$$\begin{aligned} r(\pi) &= \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi] \\ &= \lim_{t\to\infty} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi] \\ &= \sum_s \mu_\pi(s) \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) r \end{aligned}$$

  where μ_π is the steady-state distribution, which (in this setting) is assumed to exist for any π and to be independent of S_0.
- This assumption about the MDP is known as *ergodicity*.

- One can then order policies according to $r(\pi)$.
- In particular, we consider all policies that attain the maximal value of $r(\pi)$ to be optimal.
- Sutton and Barto (2018) actually say that in the context of function approximation, one should always use the average-reward framework.

- In the average-reward setting, goals are defined in terms of differences between rewards and the average reward:

$$G_t := R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + \dots$$

  The corresponding value functions are known as *differential value functions*. They are defined in the same way and we will use the same notation for them.
- Differential value functions also have Bellman equations, just slightly different from those we have seen earlier. We simply remove all γs and replace all rewards by the difference between the reward and the true average reward. See Sutton and Barto (2018) for details.

- The state variable s_t is a data structure which, simply put, must contain everything the agent needs to make a trading decision, and nothing else.
- Candidate state variables include:
  1. the current position or holding in the asset
  2. the values of any signals which are believed to be predictive
  3. the current state of the market, including current price and any relevant microstructure / limit-order book details
  4. If options are involved, additional variables such as time to expiry.

- In trading problems, the most obvious choice for an action is the number of shares to trade.
- If the agent's interaction with the market microstructure is important then there will typically be more choices to make, and hence a larger action space.
- For example, the agent could decide which execution algorithm to use, whether to cross the spread or be passive, target participation rate, etc.
- If one of the assets is an option, there may be additional actions available, such as early exercise.

- It should now be clear from our discussion above that the state vector will typically be an object which is "inconveniently" high dimensional.
- For example, with 10 predictive signals, 5 variables describing the current state of the limit order book, and a few other things such as expiration time and current holding, we get a 20-dimensional state vector.
- One should forget about enumerating a list of states in real-world trading problems.

- Recall our earlier discussion of how these models can be trained.
- Simulation leads to a sequence of (X, Y) pairs where, recall

$$X_t = (s_t, a_t), \quad Y_t = r_t + \gamma \max_{a'} \hat{q}_t(s', a')$$

  with $\hat{q}_t$ denoting the best approximation of q_* as it existed at the t-th time step.
- Function approximation methods focus on learning the unknown function f where $Y = f(X) + \text{noise}$. In other words, regression.
- This is an old problem in statistics, and can be approached by many methods, such as Artificial Neural Networks, basis functions, etc.
- Most regression methods are not frightened by the idea that X can be a 20-dimensional vector!

The Ornstein-Uhlenbeck Process

- For this example, assume that there exists a tradable security with a strictly positive price process $p_t > 0$. (This "security" could itself be a portfolio of other securities, such as an ETF or a hedged relative-value trade.)
- Further suppose that there is some "equilibrium price" p_e such that $x_t = \log(p_t/p_e)$ has dynamics

$$dx_t = -\lambda x_t + \sigma\, \xi_t \tag{2.1}$$

  where $\xi_t \sim N(0,1)$ and ξ_t, ξ_s are independent when $t \neq s$.
- This means that p_t tends to revert to its long-run equilibrium level p_e with mean-reversion rate λ.
- These assumptions imply something similar to an arbitrage! Positions taken in the appropriate direction while very far from equilibrium have very small probability of loss and extremely asymmetric loss-gain profiles.

- We do not allow the agent, initially, to know anything about the dynamics.
- Hence, the agent does not know λ, σ, or even that some dynamics of the form (2.1) are valid.

- The agent also does not know the trading cost.
- We charge a spread cost of one tick size for any trade.
- If the bid-offer spread were equal to two ticks, then this fixed cost would correspond to the slippage incurred by an aggressive fill which crosses the spread to execute.
- If the spread is only one tick, then our choice is overly conservative.
- Hence

$$\text{SpreadCost}(\delta n) = \text{TickSize} \cdot |\delta n| \tag{2.2}$$

- We also assume that there is permanent price impact which has a linear functional form: each round lot traded is assumed to move the price one tick, hence leading to a dollar cost $|\delta n_t| \times$ TickSize/LotSize per share traded, for a total dollar cost for all shares

$$\text{ImpactCost}(\delta n) = (\delta n)^2 \times \text{TickSize/LotSize}. \tag{2.3}$$

- The total cost is taken to be

$$\text{cost}(\delta n) = \text{multiplier} \times (\text{SpreadCost}(\delta n) + \text{ImpactCost}(\delta n)) \tag{2.4}$$

  This functional form matches Almgren–Chriss, and some other accounts in the literature.
- The multiplier allows us to quickly and easily test the reinforcement learning agent in regimes of more or less liquidity.

- The state of the environment

$$s_t = (p_t, n_{t-1})$$

will contain the security prices p_t, and the agent's position, in shares, coming into the period: n_{t-1}.

The learned value function looks as follows.

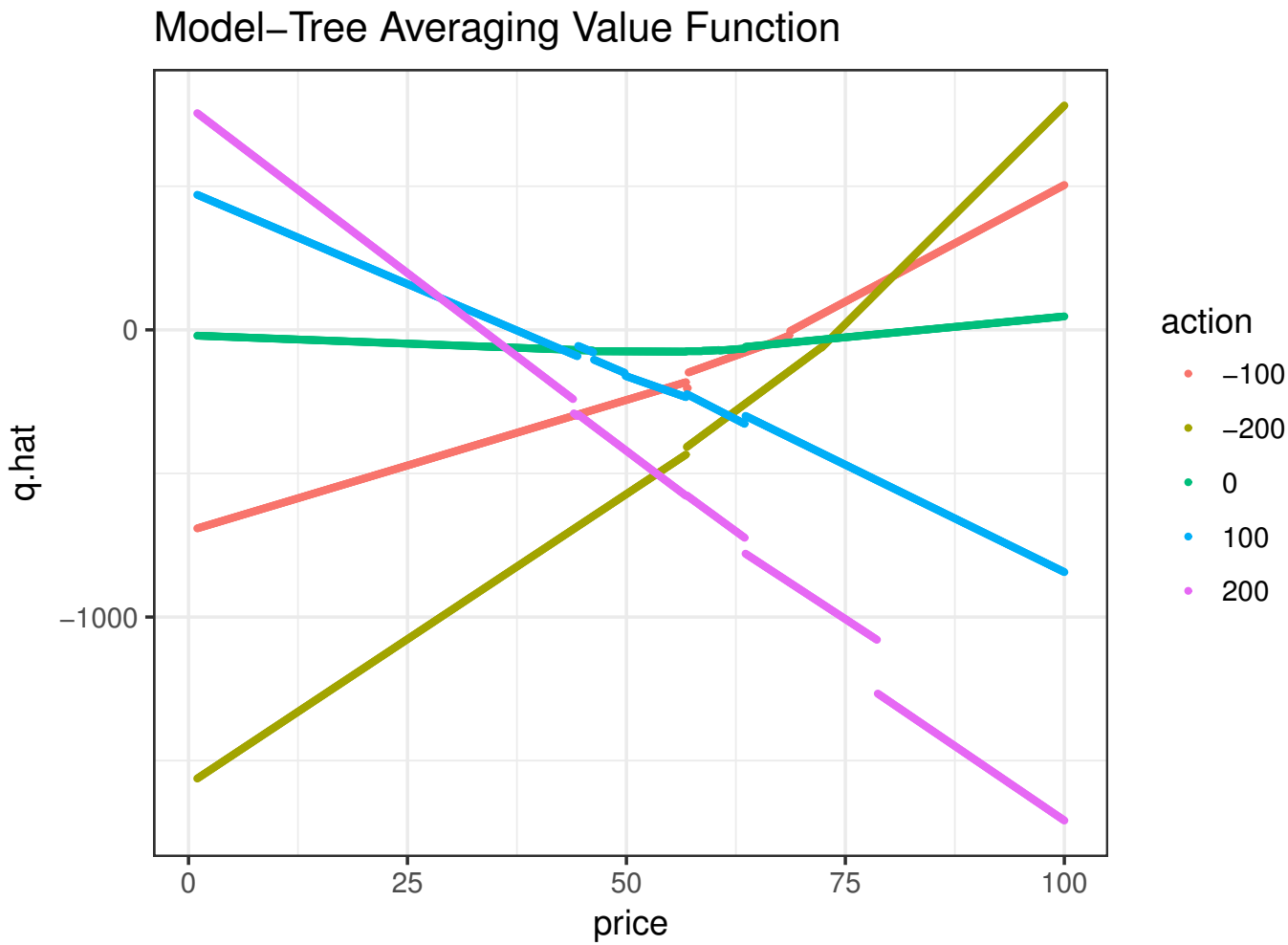


Figure: Value function $p \to \hat{q}((0, p), a)$ for various actions a, where $\hat{q}$ is estimated by using continuous state-space methods. For details, see the preprint on my Courant website.

- The relevant decision at each price level (assuming zero initial position) is the maximum of the various piecewise-linear functions shown in the figure.
- There is a no-trade region in the center, where the green line is the maximum.
- There are then small regions on either side of the no-trade zone where a trade $n = \pm 100$ is optimal, while the maximum trade of ± 200 is being chosen for all points sufficiently far from equilibrium.

- We then evaluate the system on 5,000 new samples of the stochastic process.

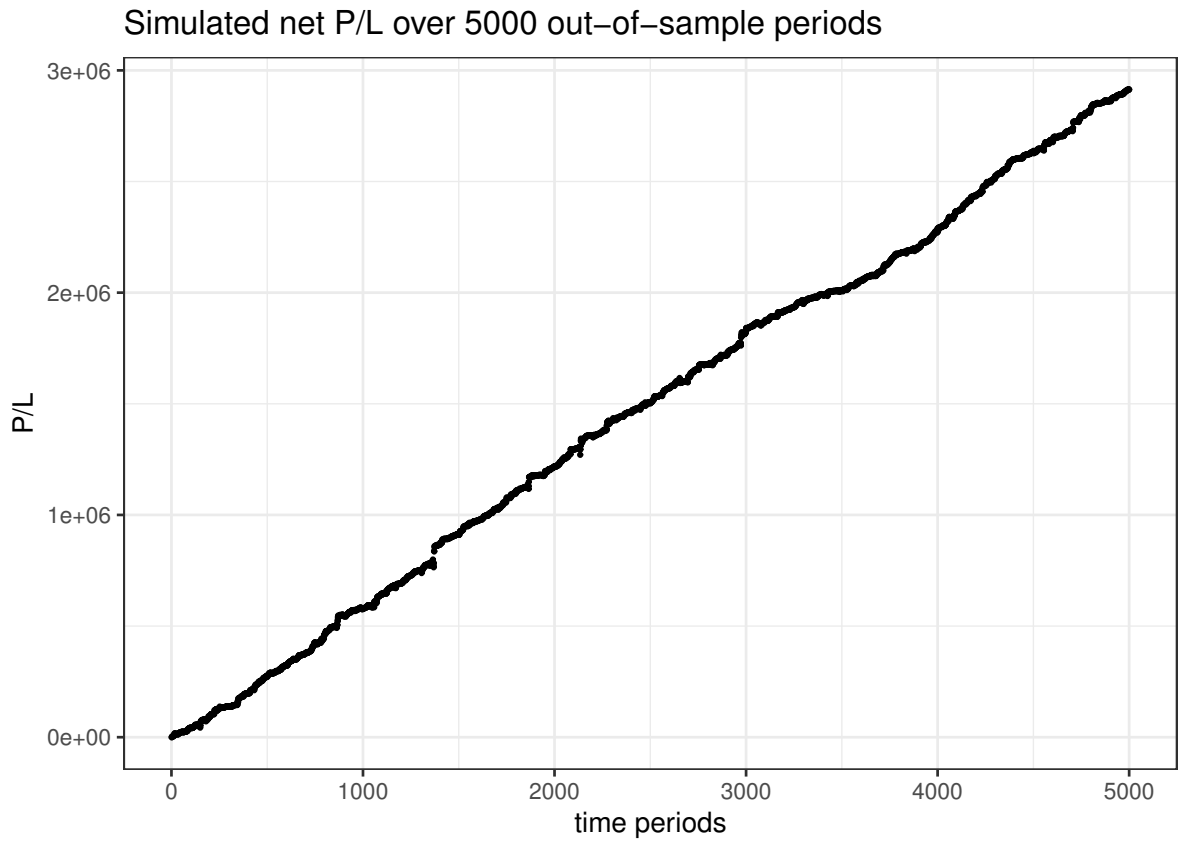


Figure: Cumulative simulated out-of-sample P/L of trained model.

- The initial results are encouraging. We constructed a system wherein we know there is an arbitrage, analogous to a game where it's actually possible to win.
- The machine then learns to play this game and learns a reasonable strategy. It finds the arbitrage!
- This opens up the possibility of viewing classical no-arbitrage theorems in a new way: Classically, there is no arbitrage in a system of stochastic price processes if there exists a risk-neutral measure. In this framework, an arbitrage is defined as a high-Sharpe trading strategy.
- There is no high-Sharpe trading strategy (and hence no arbitrage) if and only if the optimally-trained reinforcement learning agent cannot achieve a high Sharpe ratio.

Optimal Hedging for Derivatives

- This part is joint work with Petter Kolm, and was published in the inaugural issue of the Journal of Financial Data Science.
- Let's explore applying the above to another problem of interest to traders: hedging an option position.
  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.

- 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

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

- 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 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.

- 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 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 volatilty of the price process,
  4. the BSM formula,
  5. the payoff function $(S - K)_+$ at maturity,
  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.
- The results are depicted in Figure 3.1.

Figure: Out-of-sample simulation of a trained agent. We depict cumulative stock, option, and total P&L; RL agent's position in shares (stock.pos.shares), and $-100 \cdot \Delta$ (delta.hedge.shares). Observe that (a) cumulative stock and options P&L roughly cancel one another to give the (relatively low variance) total P&L, and (b) the RL agent's position tracks the delta position even though they were not provided with it.

A second example for Figure 3.1.

- A key strength of the RL approach is that it does not make any assumptions about the form of the cost function (2.4).
- It will learn to optimize expected utility, under whatever cost function you provide.
- As we need a baseline, we define π_{DH} to be the policy which always trades to hedge delta to zero according to the Black–Scholes model, rounded to the nearest integer number of shares.

$$\pi_{DH}(s_t) = \pi_{DH}(p_t, \tau, n_t) := -100 \cdot \text{round}(\Delta(p_t, \tau)) - n_t \quad (3.1)$$

- Previously we had taken multiplier $= 0$ in the function cost(n) representing no frictions.
- We now take multipler $= 5$, 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 a very large position relative to the typical volume in the market), then 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.

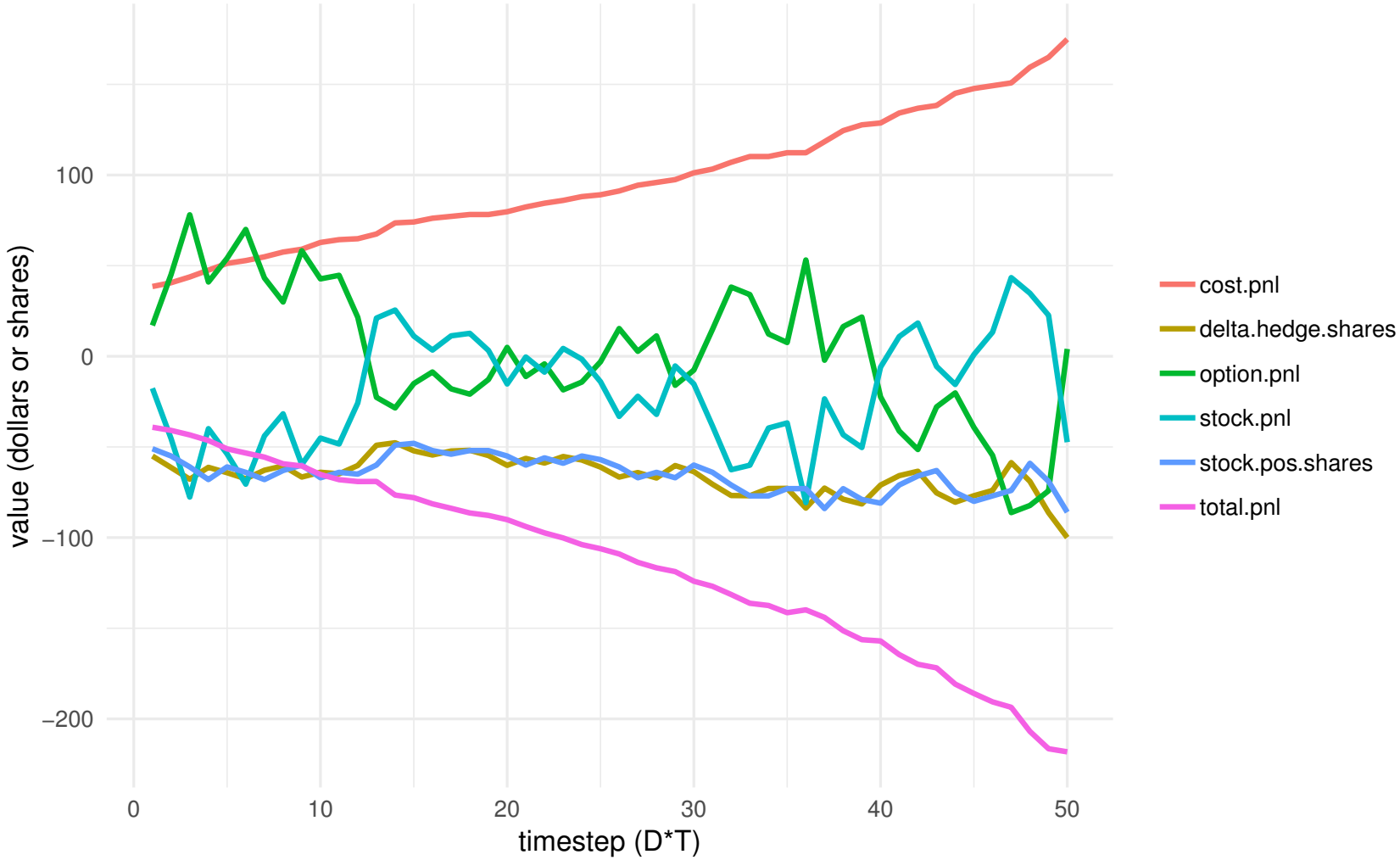
- Given that we are using a reward signal that converges to the mean-variance form of the utility function,

$$\mathbb{E}[w_T] - \frac{1}{2}\kappa\,\mathbb{V}[w_T],$$

  we naturally 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).
- 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 who just uses delta-hedging and ignores cost, and the RL trained agent who trades cost for realized volatility.

- Figure 3.2 shows one representative out-of-sample path of the baseline agent. We see that the baseline agent is over-trading and paying too much cost.
- Figure 3.3 shows the RL agent – we see that, while maintaining a hedge, the agent is trading in a cost-conscious way.
- The curves in Figure 3.2, representing the agent's position (stock.pos.shares), are much smoother than the value of $-100 \cdot \Delta$ (called delta.hedge.shares in Figure 3.2), which naturally fluctuates along with the GBM process.

Figure: Out-of-sample simulation of a baseline agent who uses policy "delta" or π_{DH}, defined in (3.1). 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.



Another example of the baseline agent: policy "delta" or π_{DH}.

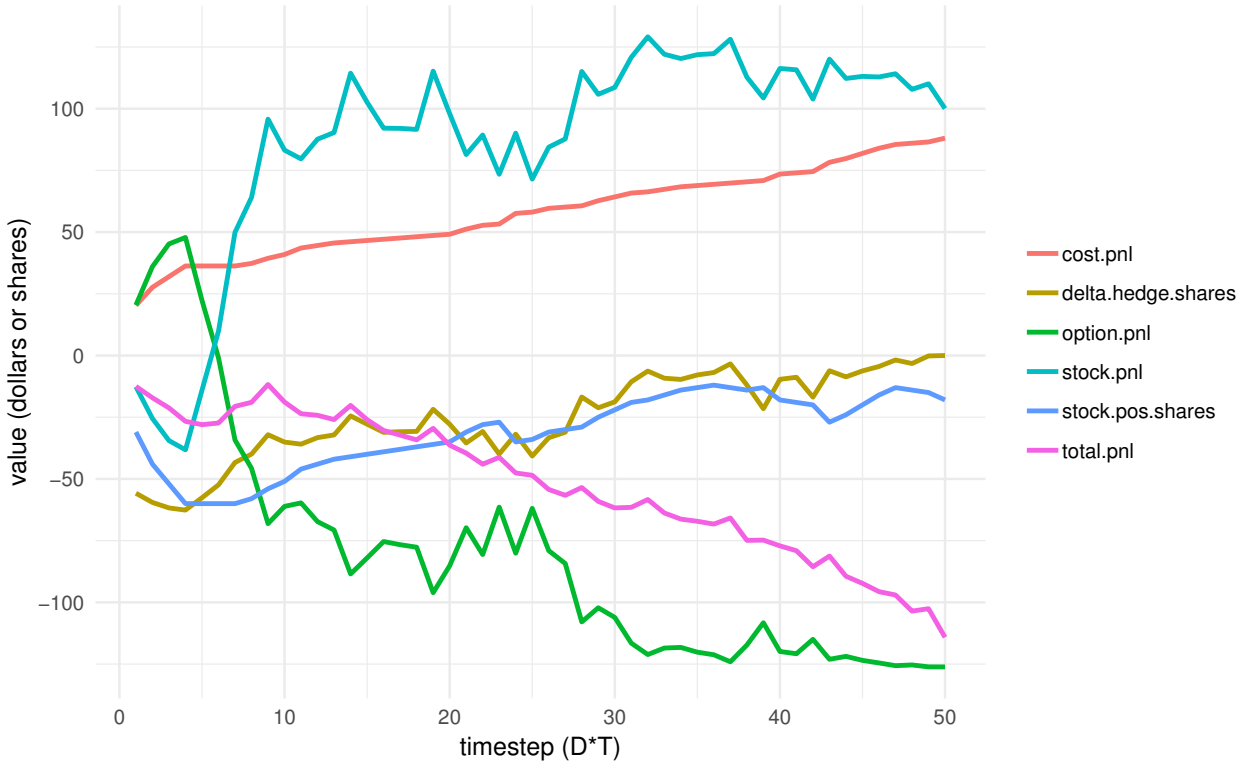


Figure: Out-of-sample simulation of our trained RL agent. The curve representing the agent's position (stock.pos.shares), controls trading costs and is hence much smoother than the value of $-100 \cdot \Delta$ (called delta.hedge.shares), which naturally fluctuates along with the GBM process.

Another example of the trained RL agent.

- Above we could only show a few representative runs taken from an out-of-sample set of $N = 10,000$ paths. To summarize all paths, we compute

$$\begin{aligned} &\text{for } i = 1, 2, \ldots, N: \\ &\quad \text{cost}_i = \text{sum}(\text{total.cost}_{i,t} \ : \ t = 1 \ldots, T) \\ &\quad \text{vol}_i \ = \text{sdev}(\text{total.pnl}_{i,t} \ : \ t = 1 \ldots, T) \end{aligned}$$

  We then plot kernel density estimates (basically, smoothed histograms) of cost_i and of vol_i, each a vector of length N.
- 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.

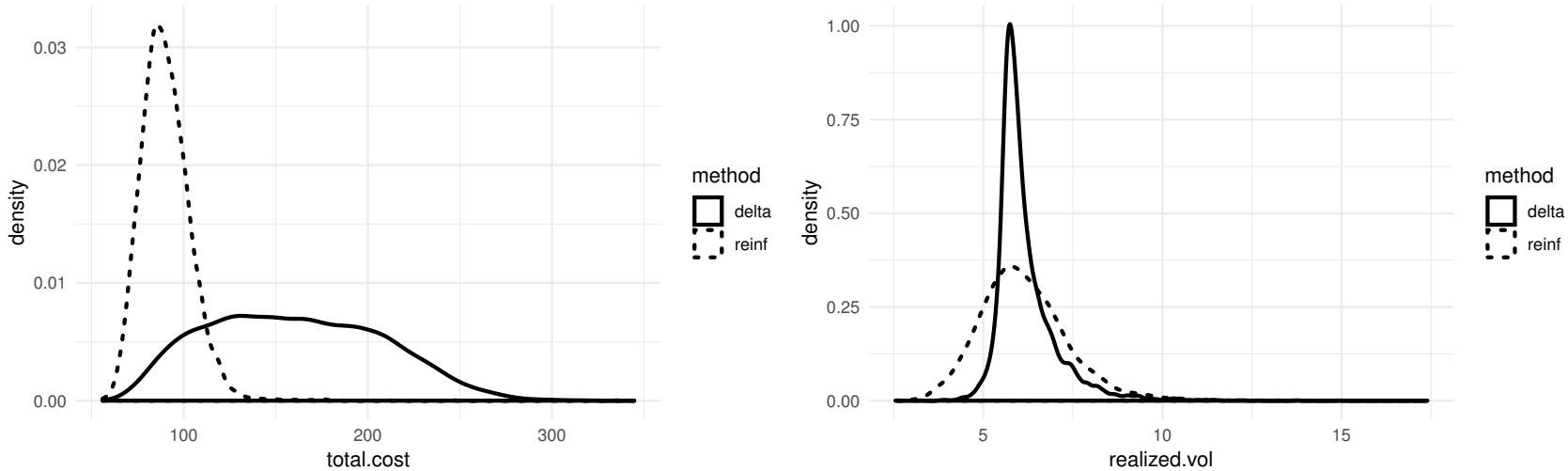


Figure: Kernel density estimates for total cost (left panel) and volatility of total P&L (right panel) from $N = 10,000$ out-of-sample simulations. Policy "delta" is π_{DH}, while policy "reinf" is the greedy policy of an action-value function trained by RL. The "reinf" policy achieves much lower cost (t-statistic $= -143.22$) with no significant difference in volatility of total P&L.

- To summarize, it does seem that machines can determine, on their own, whether or not a model (such as the Ornstein-Uhlenbeck model with Almgren-Chriss cost structure) is arbitrage-free (it isn't).
- Machines can be trained to behave in a way that is risk-averse and long-term greedy in the presence of costs.
- Paying careful attention to the actual nonlinear-function-approximation (aka supervised learning) that is used to approximate the value function, and using a prior that encourages the desired properties, is quite useful.

- The same methods can be used effectively to hedge derivatives in illiquid markets where trading costs are an extremely important contribution to wealth.
- Reinforcement learning agents can learn to price and hedge derivatives in markets where perfect replication would be impossible, or when perfect replication would be too costly.
- It can do this with just a good simulator, no other "help" from humans.


Bellman, Richard (1957). *Dynamic Programming*.


Sutton, Richard S and Andrew G Barto (1998). *Reinforcement learning: An introduction*. MIT press Cambridge.


— (2018). *Reinforcement learning: An introduction*. Second edition. MIT press Cambridge. URL: http://incompleteideas.net/book/the-book.html.