Deep Reinforcement Learning for Option Replication and Hedging

JIAYI DU, MUYANG JIN, PETTER N. KOLM, GORDON RITTER, YIXUAN WANG, AND BOFEI ZHANG

JIAYI DU
is a graduate student at New York University Center for Data Science in New York, NY.
jd4138@nyu.edu

MUYANG JIN
is a graduate student at New York University Center for Data Science in New York, NY.
mj1477@nyu.edu

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

GORDON RITTER
is an adjunct professor at the Courant Institute of Mathematical Sciences, New York University Tandon School of Engineering, Baruch College, and Rutgers University and a partner at Ritter Alpha, LP.
ritter@post.harvard.edu

YIXUAN WANG
is a graduate student at New York University Center for Data Science in New York, NY.
yw1708@nyu.edu

BOFEI ZHANG
is a graduate student at New York University Center for Data Science in New York, NY.
bz1030@nyu.edu

*All articles are now categorized by topics and subtopics. **View at PM-Research.com.**

KEY FINDINGS

- The authors propose models for the replication of options over a whole range of strikes subject to discrete trading, round lotting, and nonlinear transaction costs based on state-of-the-art methods in deep reinforcement learning including deep Q-learning and proximal policy optimization.
- The models allow the user to plug in any option pricing and simulation library and then train them with no further modifications to hedge arbitrary option portfolios.
- A series of simulations demonstrates that the deep reinforcement learning models learn similar or better strategies as compared to delta hedging.
- Proximal policy optimization outperforms the other models in terms of profit and loss, training time, and amount of data needed for training.

ABSTRACT: *The authors propose models for the solution of the fundamental problem of option replication subject to discrete trading, round lotting, and nonlinear transaction costs using state-of-the-art methods in deep reinforcement learning (DRL), including deep Q-learning, deep Q-learning with Pop-Art, and proximal policy optimization (PPO). Each DRL model is trained to hedge a whole range of strikes, and no retraining is needed when the user changes to another strike within the range. The models are general, allowing the user to plug in any option pricing and simulation library and then train them with no further modifications to hedge arbitrary option portfolios. Through a series of simulations, the authors show that the DRL models learn similar or better strategies as compared to delta hedging. Out of all models, PPO performs the best in terms of profit and loss, training time, and amount of data needed for training.*

TOPICS: ***Big data/machine learning, options, risk management, simulations****

Replication and hedging of derivatives are fundamental problems in finance. Following the seminal work of Black and Scholes (1973) and Merton (1973)—jointly referred to as BSM throughout this article—on option pricing and dynamic hedging, an impressive number of articles have focused on pricing and replicating options. At the heart of BSM is the insight that in a complete and frictionless market, one can perfectly replicate the option by using a continuously rebalanced dynamic trading strategy in the stock and riskless security.

Because of frictions such as trading costs, continuous trading of the underlying

stock is prohibitively costly in the real world. Consequently, the replicating portfolio can only be rebalanced at discrete times to keep trading costs low. Perfect replication is no longer possible, and an optimal hedging strategy depends on the trade-off between hedging error and trading costs.

Leland (1985) was the first to address discrete hedging under transaction costs. Since then, a number of other authors have contributed to this important area, including Boyle and Vorst (1992), Figlewski (1989), Grannan and Swindle (1996), Henrotte (1993), Martellini (2000), Toft (1996), and Whalley and Wilmott (1997). Although these articles mainly examine the case of proportional transaction costs, more recent studies consider option pricing and replication under nonlinear market impact, including work by Almgren and Li (2016); Bank, Soner, and Voß (2017); Rogers and Singh (2010); and Saito and Takahashi (2017). Notably, all of the aforementioned models are based on tools from intertemporal finance, such as stochastic control, dynamic programming, and classical utility theory.

Interestingly, the reinforcement learning (RL) literature developed largely independently from intertemporal finance.[1] RL provides a way to train computer models, referred to as *RL agents* or just *agents*, that, through reinforcement, learn to interact with an environment, with the goal of optimizing some reward over time. An agent does this through simple trial and error by receiving positive or negative feedback about each action it takes. Deep reinforcement learning (DRL), RL based on deep neural nets, has been shown to exceed the performance of traditional RL. Trained based on a penalty–reward mechanism, RL agents have been adopted in various real-world applications outside of finance. Perhaps the most well-known examples include deep *Q*-learning (DQN) systems that learned to play a large number of video games at a superhuman level (see Mnih et al. 2013, 2015) and AlphaGo Zero, which defeated the world's best Go player after learning the game from scratch without human guidance by playing against itself (see Silver et al. 2017). Additionally, policy gradient methods, including trust region policy optimization (TRPO) and proximal policy optimization (PPO), reach superhuman-level performance in various tasks such as robotics locomotion and game playing (see Schulman et al. 2015a, 2017). Recently, RL agents have been deployed in areas such as autonomous driving and healthcare. For instance, Wang, Chan, and de La Fortelle (2018) demonstrated that RL agents for autonomous driving can learn to make smooth and efficient lane changes under a diverse set of scenarios. Komorowski et al. (2018) developed an RL agent, called the artificial intelligence clinician (AIC), that can address the complex sequential decision-making problem arising in sepsis treatment. Remarkably, the outcome of the treatment selected by the AIC on average exceeds that of experienced human clinicians.

Recently, Kolm, and Ritter (2019) demonstrated how to apply RL to optimally hedge options in a realistic setting subject to discrete trading, round lotting, and nonlinear trading costs. Kolm and Ritter (2020) elucidated the direct link between the dynamic optimization problems in intertemporal finance and RL. They showed how to formulate intertemporal decision problems (e.g., option replication, optimal order execution, and dynamic portfolio optimization) as RL-based machine learning problems. A few other studies, including those by Buehler et al. (2018), Halperin (2019), Cao et al. (2020), and Cannelli et al. (2020), explored machine learning–based methods for option replication. Buehler et al. (2018) evaluated a neural network–based hedging approach under convex risk measures and proportional transaction costs. Halperin (2017) applied an RL-based model to pricing and hedging of options but does not consider transaction costs. Cannelli et al. (2020) compared the risk-averse contextual k-armed bandit to DQN for the hedging of options in the BSM setting, showing the former outperforms in terms of sample efficiency and hedging error. More closely related to our article, Cao et al. (2020) explored DRL methods for option replication in BSM and stochastic volatility setups, comparing the performance of accounting profit and loss (P&L) and cash flow approaches.

This article makes three main contributions. First, we develop a system based on the state-of-the-art DRL models, referred to as DQN; DQN with "preserving outputs precisely, while adaptively rescaling targets" (DQN with Pop-Art); and PPO, which learns to optimally replicate options with different strikes subject to discrete trading, round lotting, and nonlinear transaction costs.[2] We emphasize that each model is trained

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

[2] For DQN, DQN with Pop-Art, and PPO, we refer the reader to van Hasselt et al. (2016), Mnih et al. (2013, 2015), Schulman et al. (2015a, 2017), and Silver et al. (2017).

to hedge a whole range of strikes, and no retraining is needed when the user changes to another strike within the range.

Second, we show in a series of simulations that the DRL models learn similar or better strategies as compared to delta hedging. Out of all models, PPO performs the best in terms of P&L, training time, and amount of data needed for training.

Third, the models are general, allowing the user to plug in any option pricing and simulation library and then train them with no further modifications to hedge arbitrary option portfolios.

DEEP REINFORCEMENT LEARNING

In this section, we briefly review the aspects of DRL that we use throughout this article. RL provides a way to train computer models, referred to as agents, that learn to interact with an environment by means of the sequency of actions they take, with the goal of optimizing a cumulative reward over time. At each time step t, the agent observes the current state of the environment $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$. This choice influences both the transition to the next state, s_{t+1}, and the reward, R_{t+1}, the agent receives. The agent's goal is to choose actions to maximize the expected cumulative reward:

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

where the constant $\gamma \in [0, 1]$ is referred to as the discount factor. The sum in Equation 1 can be either finite or infinite, depending on the problem at hand. If rewards are bounded, then $\gamma < 1$ ensures convergence when the sum (Equation 1) is infinite. A policy π is a strategy for determining an action a_t, conditional on the current state s_t. Polices can be deterministic or stochastic. In the deterministic case, π maps $\mathcal{S} \to \mathcal{A}$; in the stochastic case, π maps a state $s \in \mathcal{S}$ to a probability distribution $\pi(a|s)$ on $\mathcal{A}$.

In this article, we focus on Q-learning and policy gradient methods based on deep neural networks (DNNs).

Q-Learning

The action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, also known as the Q-function, expresses the value of starting in state s, taking action a, and following policy π thereafter:

$$Q^\pi(s,a) := \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a] \tag{2}$$

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

$$V^\pi(s) := \mathbb{E}_\pi[G_t \mid S_t = s] = Q^\pi(s, \pi(s)) \tag{3}$$

Policy π is defined to be at least as good as π' if

$$V^\pi(s) \geq V^{\pi'}(s) \tag{4}$$

for all states s. An *optimal policy* is defined as one that is at least as good as any other policy. All optimal policies share the same optimal action-value function Q^*, the optimal action-value function. The goal of Q-learning is to learn Q^*. The optimal action-value function satisfies the Bellman equation:

$$Q^*(s,a) = \mathbb{E}[R + \gamma \max_{a'} Q^*(s', a') \,|\, s, a] \tag{5}$$

The basic idea of Q-learning is to turn the Bellman equation into the update $Q_{i+1}(s,a) = \mathbb{E}[R + \gamma \max_{a'} Q_i(s', a') \mid s, a]$ and iterate this scheme until convergence, $Q_i \to Q^*$ (see, for example, Sutton and Barto 2018). Once one has determined the optimal action-value function, the optimal policy can be computed via

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a) \tag{6}$$

Deep *Q*-Learning

In deep Q-learning the action-value function is approximated with a DNN, $Q(s, a; \theta) \approx Q^*(s, a)$, in which θ represents the network parameters. The DNN is then trained by minimizing the following sequence of loss functions:

$$L_i(\theta_i) = \mathbb{E}[L_\delta(Q(s,a;\theta_i) - R - \gamma \max_{a'} Q(s', a'; \theta_i^-)) \,|\, (s, a, R, s') \sim U(D)] \tag{7}$$

where L_δ is the Huber loss

$$L_\delta(x) = \begin{cases} \frac{1}{2}x^2, & |x| \le \delta, \\ \delta(|x| - \frac{1}{2}\delta), & \text{otherwise} \end{cases} \quad (8)$$

In the loss functions in Equation 7, $Q(s, a; \theta_i)$, $Q(s, a; \theta_i^-)$, and $U(D)$ are referred to as the policy network, target network, and behavior distribution, respectively. Although θ_i is updated at every iteration, θ_i^- is updated only every M steps. In our computational examples, we train the DQN models using experience replay and control the exploration and exploitation trade-off using the ϵ-greedy approach as done by Mnih et al. (2015, Algorithm 1).

Deep *Q*-Learning with Adaptive Scaling

DQN can experience stability issues in problems in which rewards vary significantly in magnitude, resulting in poor performance. Mnih et al. (2015) used reward clipping to address this issue. However, deciding on an acceptable range for the rewards is ad hoc and can also change the learning objective, thereby resulting in different policies.

van Hasselt et al. (2016) proposed an adaptive approach to normalize rewards that they called Pop-Art and demonstrated how it improves the stability and performance of DQN on a number of Atari 2600 games.

DQN with Pop-Art preserves the output of the unnormalized Q-function, Q, by creating a trainable square linear layer with weights $W \in \mathbb{R}^{n \times n}$ and bias $b \in \mathbb{R}^n$ to represent a modified Q-function

$$\tilde{Q}(s, a; \theta) = WQ(s, a; \theta) + b \quad (9)$$

where n is the size of the action space $\mathcal{A}$. Let us define the target $Y_j := R_j + \gamma \max_a Q(s_{j+1}, a; \theta_j)$. After initializing $W \equiv \Sigma = I$ and $b \equiv \mu = 0$ at the beginning of training, they are subsequently updated according to

$$W_{\text{new}} \leftarrow \Sigma_{\text{new}}^{-1} \Sigma W, \qquad b_{\text{new}} \leftarrow \Sigma_{\text{new}}^{-1} (\Sigma b + \mu - \mu_{\text{new}}) \quad (10)$$

where Σ_{new} and μ_{new} are chosen such that the scaled targets $\{\Sigma_{\text{new}}^{-1}(Y_j - \mu_{\text{new}})\}_{j=1}^{i}$ are normalized.

Policy Gradient Methods

We assume that each action a_t is generated by a stochastic policy such that $a_t \sim \pi_\theta(a_t|s_t)$ with parameters θ. In contrast to improving the approximation of the action-value function as is done in Q-learning, policy gradient methods aim to directly learn the policy π_θ^* that maximizes the cumulative reward (Equation 1) by performing a gradient update of θ, such that

$$\theta_{i+1} = \theta_i + \alpha_i \nabla_\theta J|_{\theta=\theta_i} \quad (11)$$

where $J(\theta) := \mathbb{E}_{\pi_\theta}[G_t]$ and $\alpha_i > 0$ is the learning rate at step i. Policy gradient methods differ in terms of how gradients are estimated. Most are guaranteed to converge to the optimal policy, and in practice they often converge faster than Q-learning.

Frequently, the following gradient estimator is used:

$$\hat{g}(\theta) = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t] \quad (12)$$

where $\hat{\mathbb{E}}_t$ denotes the empirical average over a finite batch of samples, and $\hat{A}_t$ is an estimate of the advantage function $A_t := Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$ at time step t (see Schulman et al. 2017). Computationally, it is convenient to work directly with the loss function $L^{PG}(\theta)$ defined such that $\nabla_\theta L^{PG}(\theta) \equiv \hat{g}(\theta)$; that is

$$L^{PG}(\theta) := \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t) \hat{A}_t] \quad (13)$$

Proximal Policy Optimization

A challenge with policy gradient methods is that estimates of the gradients have high variance, resulting in a lack of robustness and poor convergence. Additionally, basic policy gradient methods are often not data efficient. We choose to use PPO because it has been shown to be data efficient and robust in a number of practical applications (see Schulman et al. 2015a, 2017). PPO is based on several recent developments in policy gradients, and a full description is beyond the scope of this article. In the following, we highlight some aspects of PPO that are important for the models in this article and refer the reader to Schulman et al. (2017) for details.

By combining several objectives, the PPO loss function is defined as

$$L_t^{\text{PPO}}(\theta) := \hat{\mathbb{E}}_t[L_t^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t)] \quad (14)$$

where $L_t^{\text{CLIP}}(\theta)$ is a clipped surrogate objective, $L^{\text{VF}}(\theta) := (V^{\pi_\theta}(s_t) - V_t^{\text{target}})^2$ is the squared-error between predicted and realized value functions, and $S[\pi_\theta]$ denotes an entropy bonus. The entropy bonus controls the exploration–exploitation trade-off during training. The hyperparameters c_1, $c_2 \geq 0$ determine the trade-offs among the terms in the loss function. Schulman et al. (2017) proposed the clipped surrogate objective, $L_t^{\text{CLIP}}(\theta)$, as a modification of the surrogate objective used in TRPO (see Schulman et al. 2015a), defined as

$$L_t^{\text{CLIP}}(\theta) := \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\text{old}}(a_t|s_t)}, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\text{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\right)\hat{A}_t \quad (15)$$

where $\epsilon > 0$, $\pi_\theta(a_t|s_t)$ and $\pi_{\text{old}\theta}(a_t|s_t)$ are the new and current policies and $\hat{A}_t$ is an estimate of the advantage function at time step t. In our computational examples, we used general advantage estimation (GAE) (see Schulman et al. 2015b) to estimate the advantage function, $\hat{A}_t := \hat{A}_t^{\text{GAE}(\gamma,\lambda)}$. The clipping operation limits the policy to the interval $[1 - \epsilon, 1 + \epsilon]$, thereby reducing the variance of the policy gradient estimate.

AUTOMATIC HEDGING

Kolm and Ritter (2019) defined automatic hedging as "the practice of using trained reinforcement learning agents to handle the hedging of certain derivative positions." Although the approach in this article in principle can be applied to arbitrary portfolios of derivative securities, here we consider the training of DQN agents for the purpose of replicating static long positions in European call options of different strikes by trading the underlying stock. In a complete market with continuous and frictionless trading, such as in the BSM model, there is a dynamic trading strategy in the stock that replicates the option position perfectly. Specifically, the P&L of the hedged portfolio, defined as the P&L of the option position minus the replication strategy, has zero variance. When frictions are no longer absent and only discrete trading is possible, then the goal of replication becomes minimizing (1) the variance of the P&L of the hedged portfolio and (2) the cost of replication.

We consider a complete market in which stock prices follow a geometric Brownian motion (GBM) with initial price S_0 and a daily lognormal volatility of σ/day. We assume options are European and mature in T days and that the risk-free rate is zero.

In RL, at each time step t the agent observes the state of the environment, takes an action, and then receives a reward. In the following, we describe the state and action spaces, reward function, and trading costs we use in training the models.

State Space

The state must contain the information relevant for making the optimal decision. Information that is not relevant to the problem, or that can be derived from other state variables, does not need to be included. For the replication of European options with different strikes, a natural state space is given by

$$\mathcal{S} := \mathbb{R}_+^2 \times \mathbb{Z} \times \mathbb{N} = \{(S, \tau, n, K) | S > 0, \tau > 0, n \in \mathbb{Z}, K \in \mathbb{N}\} \quad (16)$$

Consequently, at each time step t, the agent observes the four-dimensional state vector $s_t = (S_t, \tau, n_t, K)$ where S_t is the price of the stock at time t; $\tau := T - t > 0$ is the time remaining to maturity of the option; n_t is the current number of shares held; and K is the option strike. We emphasize that the state does not need to include the option Greeks because they are functions of the variables the agent has access to via the state. With enough training data, we expect the agent will learn such nonlinear functions on its own as needed.

Action Space

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

$$\mathcal{A} := \{-100 \cdot L, \ldots, 100 \cdot L\} \quad (17)$$

where L is the number of option contracts held, each for 100 shares.

Reward Function

Next, we turn to deriving the reward function. We assume agents have quadratic utility,[3] which implies that their optimal portfolios are given as solutions of the mean–variance optimization problem

$$\max_{\pi}\left(\mathbb{E}[w_T]-\frac{\kappa}{2}\mathbb{V}[w_T]\right) \tag{18}$$

with κ denoting an agent's risk aversion and π the optimal policy, and where the final wealth w_T is the sum of individual wealth increments δw_t

$$w_T = w_0 + \sum_{t=1}^{T}\delta w_t \tag{19}$$

It is easy to see that wealth increments can be decomposed as

$$\delta w_t = q_t - c_t \tag{20}$$

where q_t follows a random walk, and c_t is the total trading cost in period t. Trading costs can include commissions, bid–offer spread cost, market impact cost, and other sources of slippage. Observe that when the risk-free rate is zero, from Equation 20, it immediately follows that $\mathbb{E}[\delta w_t] = -\mathbb{E}[c_t]$, and the objective function in Equation 18 then describes the trade-off between cost and variance of P&L as measured by the risk-aversion, κ.

To solve the replication problem using RL, we need to choose a reward R_t such that $\mathbb{E}(R_t) = \mathbb{E}[\delta w_t] - \frac{\kappa}{2}\mathbb{V}[\delta w_t]$. One possibility is

$$\delta w_t - \frac{\kappa}{2}(\delta w_t - \hat{\mu})^2 \tag{21}$$

where $\hat{\mu}$ is an estimate of $\mu := \mathbb{E}[\delta w_t]$, the expected wealth increment over one period. Often in trading problems in which the time increment t is small, $\hat{\mu}$ is negligible; hence, $\mathbb{E}[(\delta w_t - \hat{\mu})^2] \approx \mathbb{E}[(\delta w_t)^2]$.[4] We define the one-period reward function as

$$R_t := \delta w_t - \frac{\kappa}{2}(\delta w_t)^2 \tag{22}$$

and use it to compute the values in Equation 1.

Rebalancing and Trading Costs

We allow agents to rebalance their replicating portfolio D times per day such that each *episode* has a total of $T \cdot D$ time steps. All trades are subject to trading costs defined by

$$\text{cost}(n) = C \cdot \text{TickSize} \cdot (|n| + 0.01n^2) \tag{23}$$

where n is the number of shares traded and C is a cost multiplier. The term TickSize $\cdot$ $|n|$ represents the cost of crossing a bid–offer spread that is two ticks wide. The quadratic term in Equation 23 is a simplistic model for market impact. An advantage of the RL approach is that it does not make any assumptions about the form of the cost function (Equation 23). It will learn to optimize expected utility under whatever cost function one prefers.

BSM Benchmark and Parameter Settings

We compare the performance of the agents with the delta hedging baseline policy

$$\pi_{DH}(s_t) = -100 \cdot \text{round}(\Delta(s_t)) - n_t \tag{24}$$

where n_t is the current stock holdings of an agent following a standard delta hedging policy, 100 is the number of stocks controlled by one call option, and $\Delta(s_t)$ is the corresponding BSM delta of the option position at time t.

In the computational examples, we use the following parameter settings: $\sigma = 0.01$, $S_0 = 100$, $T = 10$, $D = 5$, $K \in \{98, 99, 100, 101, 102\}$, $C \in \{0, 1, 3, 5\}$, and TickSize = 0.1.

COMPUTATIONAL EXAMPLES

Simulation Environment

To train RL agents, one needs a lot of data. The majority of the most successful RL applications to date rely on generating training data in simulation

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

[4] See Ritter 2017 for an RL approach that avoids this approximation.

environments, rather than training on historical data. With a simulator of the environment, one can generate as much data as one needs to accurately train the models.

We implemented a simulation environment for options trading in OpenAI Gym using Python and PyTorch (see https://keras.io/ and Paszke et al. 2019). Although our simulation environment is similar to that of Kolm and Ritter (2019), who considered the training of an option with a chosen fixed strike price, our implementation allows training options with the same underlier for a range of strike prices simultaneously.

The simulator has three main steps. First, it generates training data by sampling episodes of stock returns from a price process. Second, it samples a strike price from a selected range of strikes and, based on the state representation, computes the corresponding option price for the current time step. Third, after receiving the action chosen by the RL agent, the current state vector is updated, and the simulator increments the time step. The environment computes the reward according to Equation 22, and the reward and state vector are recorded into the replay memory to be used in future training. One epoch consists of 3,000 episodes, in which each episode is one complete path of 50 price observations (i.e., $T \cdot L = 10 \cdot 5$). We update the data only every five epochs.

In the computational examples in this article, the price process is a GBM initialized as by Kolm and Ritter (2019), and we consider an agent holding a long position in a call option, controlling 100 shares of stocks, with strike K in some range.

Model Architecture and Hyperparameters

We use a standard multilayer perceptron architecture with five hidden layers with ReLU activation functions for DQN and PPO. In our computational experiments, we found that five hidden layers provided the best trade-off between training time and hedging performance. To improve the speed of convergence in the training phases, we use batch normalization before each ReLU in the hidden layers. For DQN, the network outputs a vector of the same size as the action space, each component representing the value of the Q-function conditional on the corresponding action. For PPO, the outputs from the final hidden layer are fed into two individual linear layers to produce a policy vector of the same size as the action space, where each component represents the probability of each action and a scalar for the value function (Equation 3). In the case of DQN with Pop-Art, we normalize the Q-function according to the formulas in Equations 9 and 10.

To train the DQN and PPO models, we use the Adam optimizer with an initial learning rate of 10−4. We train DQN with Pop-Art using stochastic gradient descent with a fixed learning rate of 10−4. In all of the models, we clip gradients such that their norm is less than one.

We perform grid search on the set of hyperparameters and select the configuration with the best out-of-sample performance. We found that a discount factor, γ, in the range of [0.8, 0.9] gave the best performance for DQN and for DQN with Pop-Art. For PPO, one needs to tune the hyperparameter λ used in GAE. Theoretically, λ should be close to one for the best performance. However, Schulman et al. (2015b) showed that large λ can lead to gradient estimates with high variance, resulting in slow convergence.

In our computational experiments, we observed that a larger entropy bonus in the PPO loss function (Equation 14) results in well-behaved and financially intuitive policies, whereas a smaller entropy bonus sometimes leads to unstable and nonintuitive polices. For the PPO loss function (Equation 14), we fix $c_1 = 0.5$ and tune c_2, finding that $c_2 = 0.2$ provides the best-performing policies.

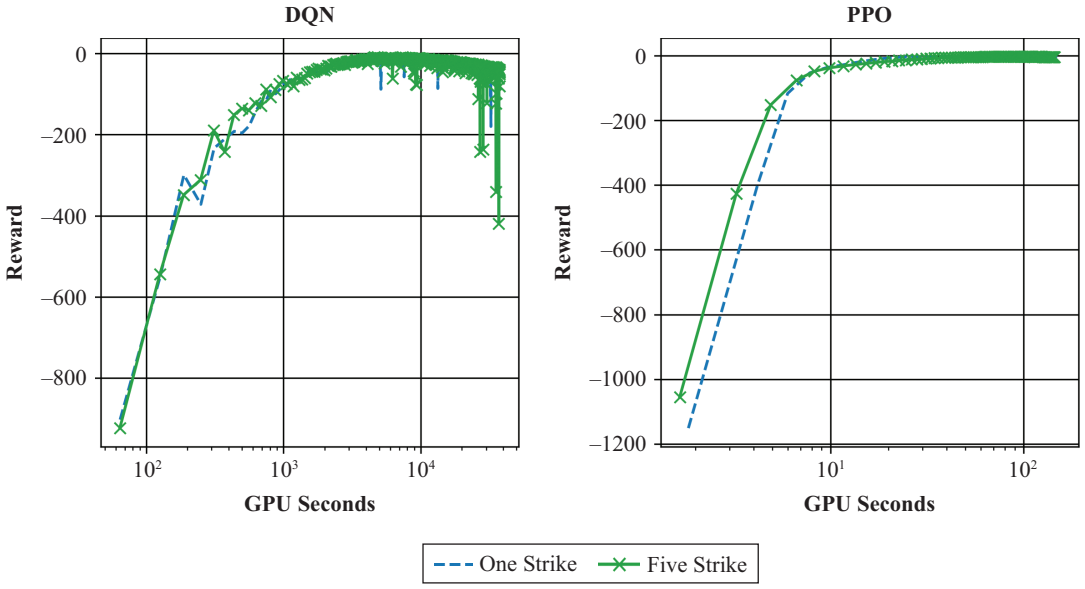
We conduct our computational experiments on an NVIDIA GTX 1080 GPU. Unlike Kolm and Ritter (2019), we simultaneously train a whole range of strikes, $K \in \{98, 99, 100, 101, 102\}$, for each model. For trading costs (Equation 23), we use the cost multipliers $C \in \{0, 1, 3, 5\}$, corresponding to no, low, moderate, and high costs. For out-of-sample validation, we use 38,500 randomly generated episodes as our test set.

Training Time and Convergence

It is well known that training networks for DRL can be time consuming (see, for example, Cruz, Du, and Taylor 2017). Therefore, we first examine training times and convergence of our models. Exhibits 1 and 2 show average time in GPU seconds and the number of data points used until convergence. Here, one data point represents the state and corresponding reward. We observe that, on average, it takes DQN about 10^4 GPU seconds to converge on a single GPU, whereas PPO only needs about 10^2 GPU seconds. PPO converges more quickly because, on average, DQN requires an order of magnitude more data points. In fact, the training

EXHIBIT 1

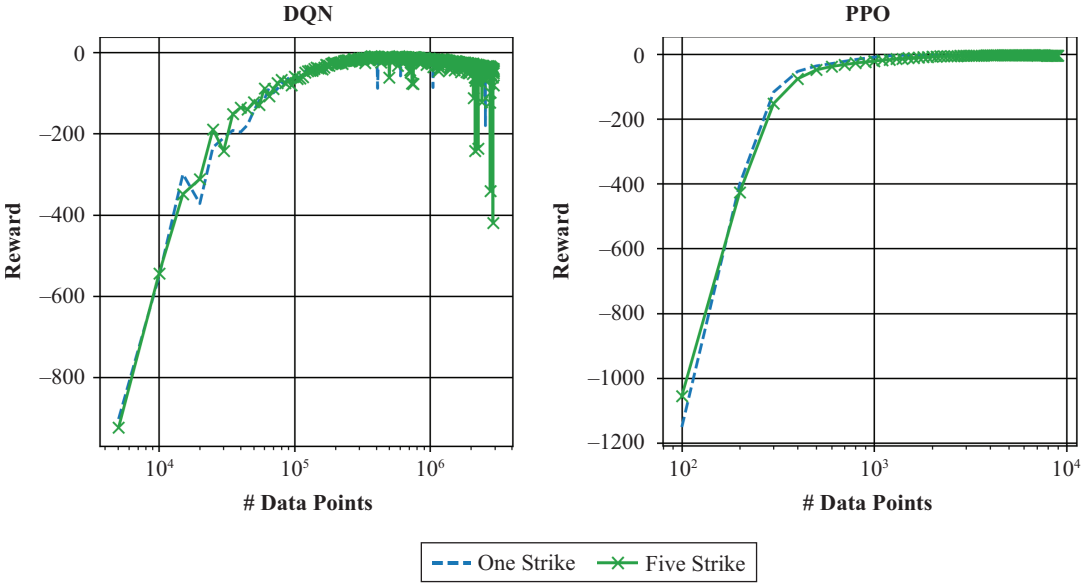
Average Reward versus Training Time for DQN and PPO



Notes: The left panel shows average reward versus GPU seconds for the DQN agent in the one (dashes) and five strike (crosses) scenarios. The right panel shows the average reward versus GPU seconds for the PPO agent in the one (dashes) and five strike (crosses) scenarios.

EXHIBIT 2

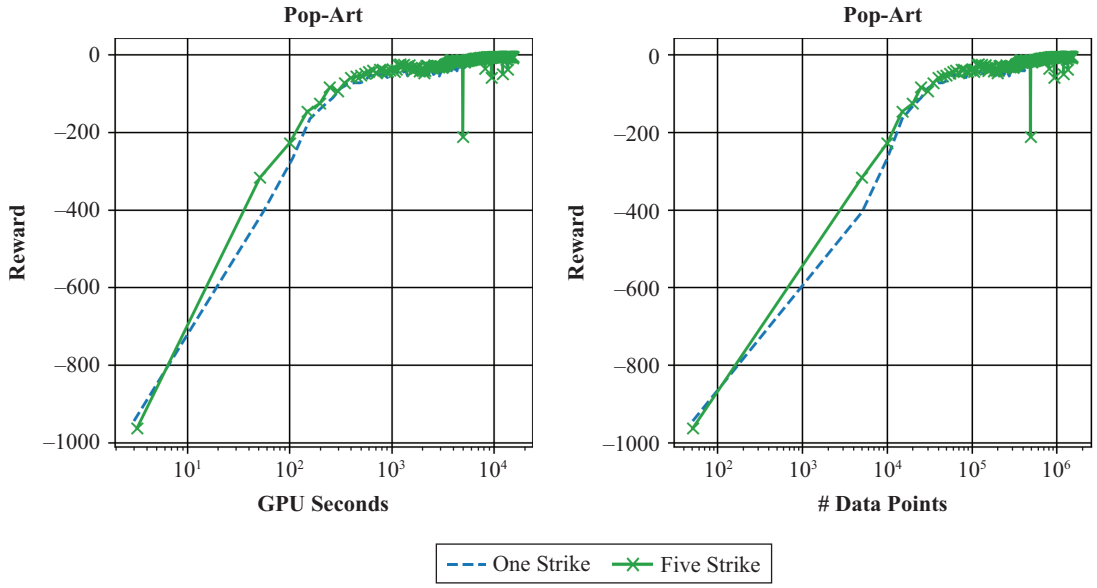
Average Reward versus Training Data Size for DQN and PPO



Notes: The left panel shows the average reward versus training data size for the DQN agent in the one (dashes) and five strike (crosses) scenarios. The right panel shows the average reward versus training data size for the PPO agent in the one (dashes) and five strike (crosses) scenarios.

EXHIBIT 3

Average Reward versus Training Time and Data Size for DQN with Pop-Art



Notes: The left panel shows the average reward versus training time by the DQN with Pop-Art agent in the one (dashes) and five strike (crosses) scenarios. The right panel shows the average reward versus training data size by the DQN with Pop-Art agent in the one (dashes) and five strike (crosses) scenarios.

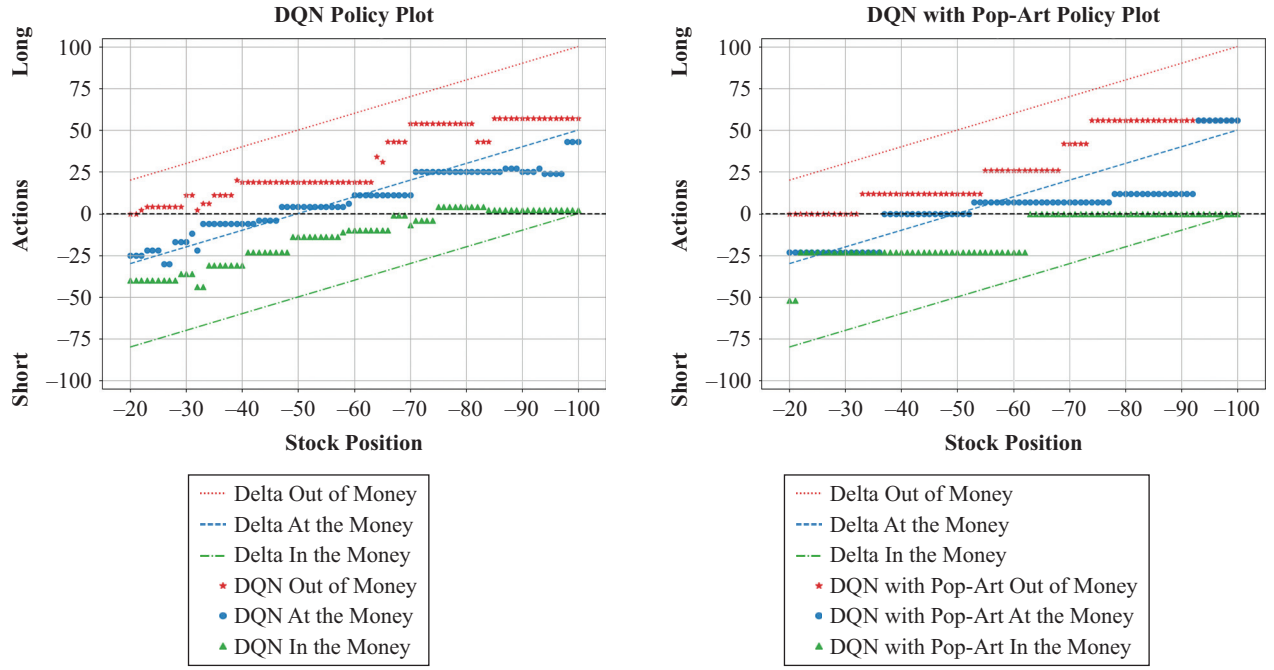
of DQN involves sampling around $5 \cdot 10^5$ versus 10^4 data points for PPO; equivalently, DQN and PPO need approximately 150 versus 6 epochs to converge.

The left panels in Exhibits 1 and 2 indicate some instability issues in the training of DQN. We determined that these are due to scaling issues of the rewards. By applying the Pop-Art normalization to DQN as discussed earlier, we obtain stable training behavior (see Exhibit 3).

Exhibit 4 provides a comparison of the policies of DQN and DQN with Pop-Art for an option struck at $K = 100$, close to maturity, and with cost multiplier $C = 1$. In the exhibit, one can examine the policies for out-of-, at-, and in-the-money cases corresponding to the stock price at 98, 100, and 102, respectively. The upward sloping lines represent the corresponding policies from the BSM model (i.e., the trades resulting from changes in delta). We observe in the left and right panels that the agents have correctly learned that, under discrete trading with costs, the resulting actions are piecewise constant as a function of the current stock position. Additionally, they have rightly identified the existence of a no-trade region—that is, a range of states (stock positions) for which the action is a zero trade. Notice the DQN policies (in the left panel) are not strictly increasing but exhibit some wiggles, indicating that the training of the agent did not fully converge. However, more training does not improve upon this behavior, as already suggested by the results from Exhibits 1 and 2. Perhaps a different training approach for DQN than ours may lead to better performance. Nonetheless, after examining our DQN training data, we reached the conclusion that data scaling issues cause the instabilities. Rather than modifying our approach to training DQN, we chose to implement the Pop-Art normalization described previously. As we see in the right panel in Exhibits 3 and 4 and 3, DQN with Pop-Art leads to stable training performance and strictly increasing policies.

To summarize, the aforementioned results show that (1) training is quite rapid, especially for PPO; (2) DQN with Pop-Art addresses the instability issues of DQN; (3) for all models, the training time is not affected by whether one or several options of the same underlier are trained; and (4) for all the models, the training time for each model is not affected by the choice of cost multiplier. For the remaining computational examples in this article, we train all models for the multistrike case.

EXHIBIT 4
Policy Plot for DQN and DQN with Pop-Art



Notes: Policies for DQN (left panel) and DQN with Pop-Art (right panel) for strike $K = 100$, close to maturity and cost multiplier $C = 1$. Policy plots show three different situations: (1) out of the money ($S_t = 98$, stars for agents, dotted line for delta), (b) at the money ($S_t = 100$, dots for agents, dashed line for delta), and in the money ($S_t = 102$, triangles for agents, dash-dotted line for delta).

Out-of-Sample Performance

In this section, we examine the out-of-sample performance of the trained models. For this purpose, we generate 38,500 random out-of-sample paths using our simulator.

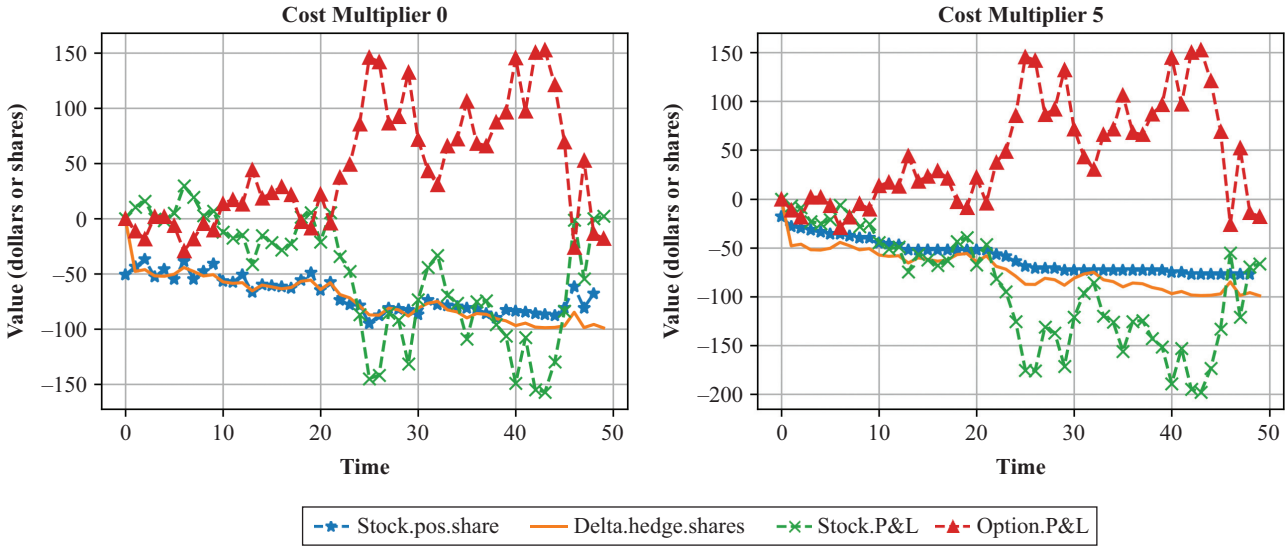
First, we examine the behavior of the models for one representative out-of-sample path. In Exhibit 5, the left and right panels depict the PPO agent's performance for the at-the-money no-cost and high-cost cases for this sample path. The performance of DQN and DQN with Pop-Art is similar to that of PPO and is omitted. In the no-cost cases, we notice that the agents' stock positions track the BSM delta position, showing that all agents have learned to hedge. In the high-cost case, although able to maintain a hedge, the agents are trading in a more cost-conscious way, suggested by a much smoother curve than the BSM delta, which naturally fluctuates with the GBM process.

To summarize the results from all 38,500 out-of-sample paths, we compute the realized volatility, total cost, and P&L of each path, representing them as kernel density plots. Exhibit 6 shows the results for at the money options in the high-cost setting. The left and right panels demonstrate that DQN agents are able to learn policies that, in comparison to the BSM model, result in lower realized P&L volatility at lower cost. One can evaluate efficacy of an automatic hedging model by how often the total P&L (including the hedging and trading costs) is significantly less than zero. The middle panel in Exhibit 6 displays density plots of the *t*-statistics of total P&L for the agents and BSM model. We observe that DQN and PPO perform the best in that their *t*-statistics are more frequently close to zero than the other models.

CONCLUSIONS

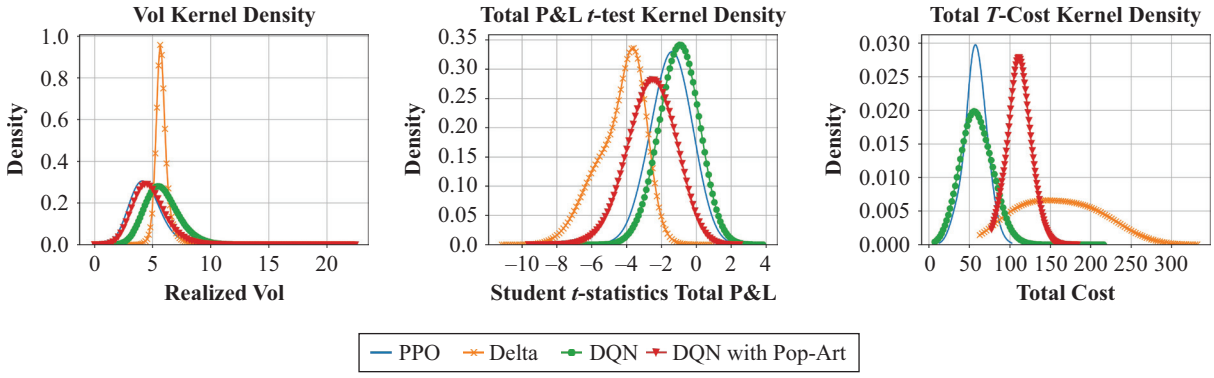
In this article, we developed a system based on (1) deep Q-learning, (2) deep Q-learning with Pop-Art, and (3) proximal policy optimization, all state-of-the-art DRL models that learn to optimally replicate options with different strikes subject to realistic conditions, including discrete trading, round lotting, and nonlinear transaction costs. A feature of the system is that each

EXHIBIT 5
An Out-of-Sample Simulation of PPO



Notes: The left panel shows cumulative stock, option, and total P&L; RL agent's position in shares (stock.pos.shares); and −100 · round(Δ(s$_t$)) (delta.hedge.shares) for cost multiplier C = 0. The right panel shows cumulative stock, option, and total P&L; RL agent's position in shares (stock.pos.shares); and −100 · round(Δ(s$_t$)) (delta.hedge.shares) for cost multiplier C = 5.

EXHIBIT 6
Kernel Density Plots for At-the-Money Options



Notes: Kernel density plots of realized volatility (left panel), t*-statistic of total P&L (middle panel), and total costs (right panel) of the BSM model and the PPO, DQN and DQN with Pop-Art agents hedging options struck at the money (K = 100) for cost multiplier C = 5.*

model is trained to hedge a whole range of strikes, and no retraining is needed when the user changes to another strike within the range.

In a series of simulations, we demonstrated that the DRL models learn similar or better strategies as compared to delta hedging. Out of all models, we concluded that PPO performs the best in terms of P&L, training time, and amount of data needed for training.

Our proposed models are general, allowing the user to plug in any option pricing and simulation library and then train them with no further modifications to hedge arbitrary option portfolios.

In closing, we comment on some of the advantages of the RL system we proposed for the replication of derivatives. The system is model-free, not requiring many assumptions. Specifically, no assumptions are required

about price processes of the derivatives and hedging securities or transaction costs incurred from trading. All the system needs is a good simulator, in which price processes and transaction costs are accurately simulated. The system does not depend on the existence of a perfect dynamic trading strategy replicating the derivatives. Rather, it learns to trade off variance and cost as best as possible using any hedging securities provided. In particular, the system will find the best minimum-variance dynamic replication strategy, whether or not the minimum-variance is actually zero, in contrast to classical derivative pricing models in complete markets. This is an important point because in the real world markets cannot be assumed to be complete; hence, some securities required for perfect replication may not exist.

REFERENCES

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. Voß. 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.

Cannelli, L., G. Nuti, M. Sala, and O. Szehr. 2020. "Hedging Using Reinforcement Learning: Contextual *k*-Armed Bandit versus Q-Learning." *arXiv* 2007.01623.

Cao, J., J. Chen, J. C. Hull, and Z. Poulos. 2020. "Deep Hedging of Derivatives Using Reinforcement Learning." *SSRN* 3514586.

Cruz Jr., G. V., Y. Du, and M. E. Taylor. 2017. "Pre-Training Neural Networks with Human Demonstrations for Deep Reinforcement Learning." *arXiv* 1709.04083.

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

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.

———. 2019. "The QLBS Q-Learner Goes NuQLear: Fitted Q Iteration, Inverse RL, and Option Portfolios." *Quantitative Finance* 19 (9): 1543–1553.

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

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

Kolm, P. N., and G. Ritter. 2019. "Dynamic Replication and Hedging: A Reinforcement Learning Approach." *The Journal of Financial Data Science* 1, no. 1 (Jan): 159–171.

———. 2020. "Modern Perspectives on Reinforcement Learning in Finance." *The Journal of Machine Learning in Finance* 1 (1).

Komorowski, M., L. A. Celi, O. Badawi, A. C. Gordon, and A. A. Faisal. 2018. "The Artificial Intelligence Clinician Learns Optimal Treatment Strategies for Sepsis in Intensive Care." *Nature Medicine* 24 (11): 1716–1720.

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

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. 2013. "Playing Atari with Deep Reinforcement Learning." *arXiv* 1312. 5602. http://arxiv.org/abs/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 no. 7540 (Feb): 529–533.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In *Advances in Neural Information Processing Systems*, 32nd ed., edited by, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, pp. 8024–8035. Red Hook, NY: Curran Associates, 2019.

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.

Schulman, J., S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. 2015a. "Trust Region Policy Optimization." *arXiv* 1502.05477.

Schulman, J., P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. 2015b. "High-Dimensional Continuous Control Using Generalized Advantage Estimation." *CoRR* abs/1506.02438.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. 2017. "Proximal Policy Optimization Algorithms." *arXiv* 1707.06347.

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: MIT Press, 2018.

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.

van Hasselt, H. P., A. Guez, M. Hessel, V. Mnih, and D. Silver. "Learning Values across Many Orders of Magnitude." In *NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems*, edited by D. D. Lee and U. von Luxburg, pp. 4287–4295. Red Hook, NY: Curran Associates, 2016.

Wang, P., C. Chan, and A. de La Fortelle. "A Reinforcement Learning Based Approach for Automated Lane Change Maneuvers." In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1379–1384.

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.

ADDITIONAL READING

Dynamic Replication and Hedging: *A Reinforcement Learning Approach*

Petter N. Kolm and Gordon Ritter

The Journal of Financial Data Science
https://jfds.pm-research.com/content/1/1/159

ABSTRACT: *The authors of this article address the problem of how to optimally hedge an options book in a practical setting, where trading decisions are discrete and trading costs can be nonlinear and difficult to model. Based on reinforcement learning (RL), a well-established machine learning technique, the authors propose a model that is flexible, accurate and very promising for real-world applications. 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 that it needs is a good simulator, in which transaction costs and options prices are simulated accurately.*

Greedy Online Classification of Persistent Market States Using Realized Intraday Volatility Features

Peter Nystrup, Petter N. Kolm, and Erik Lindström

The Journal of Financial Data Science
https://jfds.pm-research.com/content/2/3/25

ABSTRACT: *In many financial applications, it is important to classify time-series data without any latency while maintaining persistence in the identified states. The authors propose a greedy online classifier that contemporaneously determines which hidden state a new observation belongs to without the need to parse historical observations and without compromising persistence. Their classifier is based on the idea of clustering temporal features while explicitly penalizing jumps between states by a fixed-cost regularization term that can be*

calibrated to achieve a desired level of persistence. Through a series of return simulations, the authors show that in most settings their new classifier remarkably obtains a higher accuracy than the correctly specified maximum likelihood estimator. They illustrate that the new classifier is more robust to misspecification and yields state sequences that are significantly more persistent both in and out of sample. They demonstrate how classification accuracy can be further improved by including features that are based on intraday data. Finally, the authors apply the new classifier to estimate persistent states of the S&P 500 Index.

A Simple Framework for Time Diversification

Frank J. Fabozzi, Sergio M. Focardi,
and Petter N. Kolm

The Journal of Investing
https://joi.pm-research.com/content/15/3/8

ABSTRACT: *In this article the authors provide a simple but rigorous mathematical framework for time diversification. Based on this framework, we provide a measure of time diversification that can be computed for any return distribution model and any risk measure; this measure of time diversification can be empirically ascertained with non-parametric estimates of risk and with bootstrap techniques to simulate the return distribution. The authors argue that the critical issue of time diversification is not how to interpret time diversification in sequences of IID returns, but how to make long-term forecasts. The latter involves complex issues related to the distributional properties of returns, as well as memory effects and regime shifts. The authors then discuss how the distributional properties of stock returns, long memory, and regime shifts affect time diversification.*