MODERN PERSPECTIVES ON REINFORCEMENT LEARNING IN FINANCE

PETTER N. KOLM AND GORDON RITTER

Petter N. Kolm is Clinical Professor and Director of the Mathematics in Finance Master's Program at NYU's Courant Institute of Mathematical Sciences, New York, NY
petter.kolm@nyu.edu

Gordon Ritter is Adjunct Professor at NYU's Courant Institute of Mathematical Sciences, the NYU Tandon School of Engineering, Baruch College, Rutgers University and Partner at Ritter Alpha, LP.
ritter@post.harvard.edu

Abstract. We give an overview and outlook of the field of reinforcement learning as it applies to solving financial applications of intertemporal choice. In finance, common problems of this kind include pricing and hedging of contingent claims, investment and portfolio allocation, buying and selling a portfolio of securities subject to transaction costs, market making, asset liability management and optimization of tax consequences, to name a few. Reinforcement learning allows us to solve these dynamic optimization problems in an almost model-free way, relaxing the assumptions often needed for classical approaches.

A main contribution of this article is the elucidation of the link between these dynamic optimization problem and reinforcement learning, concretely addressing how to formulate expected intertemporal utility maximization problems using modern machine learning techniques.

1. Introduction

Intertemporal choice is at the heart of many modern financial applications involve. These are decision making problems in which the timing of costs and benefits are spread out over time and where choices at one time influence the possibilities available at other points in time. In finance, common problems of this kind include pricing and hedging of contingent claims,

Key words and phrases. Dynamic programming; Finance; Hedging; Intertemporal choice; Investment analysis; Machine learning; Optimal control; Options; Portfolio optimization; Reinforcement learning.

investment and portfolio allocation, buying and selling a portfolio of securities subject to transaction costs, market making, asset liability management and optimization of tax consequences, to name a few. In this article we elucidate the link between these dynamic optimization problem and reinforcement learning (RL), machine learning models that enables RL agents to learn to make a sequence of decisions through "trial and error" incorporating feedback from its actions and experiences. In particular, we show how to map and solve expected intertemporal utility maximization using modern RL techniques.

Fundamental to these dynamic optimization problems is to determine the best actions possible that maximize the relative value between two or more payoffs at different points in time. Probably, the most common approach for solving dynamic optimization problems of this kind is *dynamic programming* (DP).[1] DP refers to a collection of algorithms that can be used to explicitly find solutions to the Bellman equation and hence compute optimal policies given a perfect model of the environment as a Markov decision process (MDP) (see, for example, Sutton and Barto (2018)). While DP can be applied in deterministic or stochastic and discrete-time or continuous-time settings, it relies on several assumptions that are rarely true in practice, including: (i) one accurately knows the dynamics of the environment, (ii) one has enough computational resources to complete the computation of the solution and (iii) the Markov property.

For many real-world financial applications, one is generally not able to implement the DP solution exactly because one or several of these assumptions are violated. Indeed, the assumptions are even violated in simple two-player board games which are surely simpler than most problems of intertemporal financial decision making. For example, although assumptions (i) and (iii) present no problems for the game of backgammon, the second is a major impediment. Because the game has about 10^{20} states, solving the Bellman equation would, due to the *curse of dimensionality*, be prohibitive.

In many problems a complete model of the system is not available and we will hence be in violation of (i). For instance, many stochastic systems in finance are complex and it is difficult to derive or estimate correct expressions of their dynamics. This is referred to as the *curse of modeling*.

[1]Another approach is that of the maximum principle (MP). The MP is more general than DP. In particular, one can show that the Bellman equations implies the MP, but not vice versa (see, for example, Intriligator (2002)).

RL provides a way of overcoming these two curses by means of building agents that act intelligently, allowing efficient solutions to problems that were considered intractable via DP. The roots of the success of RL comes from leveraging several well-known areas, including DP, Monte Carlo simulation, function approximation and machine learning (ML). For more than twenty years, the standard reference on RL has been Sutton and Barto (2018), which has been recently updated and remains close to the current state of the field.

Perhaps the most useful mathematical object from classical control theory that is being adapted for use in ML is the *value function*. A value function is a mathematical expectation in a certain probability space. The underlying probability measure is the one associated with a Markov process. When the Markov process describes the state of a system it is sometimes called a *state-space model*. The foundational treatise on value functions was written by Richard Bellman at a time when ML was not in common usage ((see, Bellman (1957)). Nonetheless, modern ML owes its existence, in part, to him.

Interestingly, RL developed largely independently from classical utility theory in economics and finance. It provides a way to train artificial agents which learn through positive reinforcement to interact with an environment. Their goal is to optimize the cumulative reward over time. Intuitively, the RL agent does this through simple "trial and error" by receiving feedback by means of the amount of reward resulting from each action it takes.

Mathematically speaking, RL is a way to solve multi-period optimal control problems. The RL 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 multi-period optimal control problem. Training refers to the process of improving on the approximation of the value functions as more training examples are made available.

Although the state of the art of RL is still evolving, the vast majority of the most successful applications in recent years utilize a simulation of the environment to generate training data (as opposed to, say, training on historical data). For example, in the by now famous example by Mnih et al. (2013) and Mnih et al. (2015), a deep RL system learned to play video games on a super-human level. The system 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 world's best Go player "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."[2]

Using simulated environments of course has the advantage that millions of training examples can be generated, limited only by computer hardware capabilities. The financial examples we present in this article follow the same pattern. The RL agents we construct are trained by interacting with a simulator.

The outline of this article is as follows. In section 2 we review the core elements of reinforcement learning (RL). We discuss some common intertemporal decision problems in trading and portfolio optimization in section 3. In section 4 we elaborate on the link between expected utility maximization and the construction of rewards needed to train RL agents to solve trading problems. RL requires the specification of state and action variables. In section 5 we discuss common state and action specifications pertaining to financial trading and portfolio optimization. In this article, we provide two concrete applications. First, in section 6 we expand upon an example originally given by Ritter (2017) of using RL to trade mean-reversion. Here, we introduce a continuous state space formulation and provide an explicit graphical representation of the resulting value function. Second, section 7 describes a RL-based approach for the hedging and replication of derivatives subject to market frictions and non-continuous trading. We provide detailed numerical simulation results, demonstrating the effectiveness of the method even in a setting with non-differentiable and nonlinear transaction costs. Section 8 concludes.

2. Overview of the Elements of RL

In this section we review the core elements of RL including Markov decision processes, value functions, policies and the Bellman equations. The notation introduced in this section will be used throughout the remainder of this article. For more details about these elements, we refer to Sutton and Barto (2018).

[2]For many simpler examples of RL applications, see Sutton and Barto (2018).

2.1. **Markov Decision Processes.** In RL a *Markov decision process (MDP)* serves the role of providing a model of the sequential decision-making problem at hand where a decision maker interacts with a system in a sequential fashion. Intuitively, when in addition to a Markov process one has the possibility of choosing an *action* from a menu of available possibilities (the "action space"), with some reward metric that tells us how good our choices were, then the augmented structure is an MDP.

The "environment" is defined simply as the part of the system outside of the RL agent's *direct* control. At each time step t, it 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, R_t, the RL agent receives. Figure 1 depicts this situation.

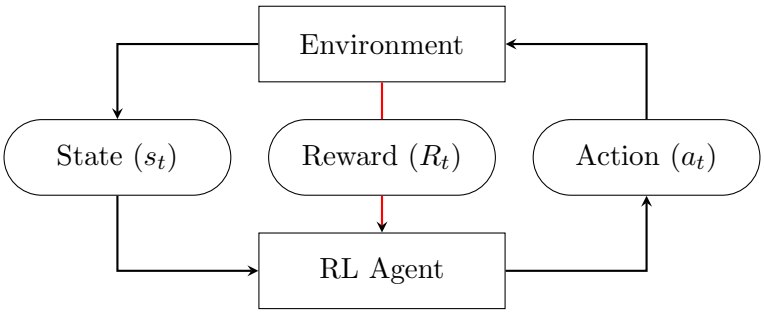

FIGURE 1. The standard RL situation. At each time step t, the RL agent observes the current state of the environment $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$. This choice influences both the transition to the next state, as well as the reward, R_t, it receives.

Underlying every MDP is a multivariate conditional probability distribution

$$p(s', r \mid s, a) \tag{1}$$

for the joint probability of transitioning to state s' and receiving reward r, conditional on the previous state being s and the RL agent taking action a when in state s. In classical applications of DP, the researcher would build a model of the system, which would allow, at least in principle, the calculation of probabilities (1). In RL, the distribution (1) is typically not known to the RL 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 one is in. More formally, a policy is a mapping from states to

probability distributions over the action space. If the RL agent is following policy π, then in state s it will choose action a with probability $\pi(a \mid s)$. To define an ordering on policies, one must specify a goal function, a real-valued random variable that is desired to be optimal in expectation.

There are two versions of goal functions in common usage: the discounted reward goal and the average-reward (or reward-per-unit time) goal. As it leads to simpler expressions of the Bellman equation, we first discuss the discounted goal function defined as

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2}$$

where R_t is the reward at time t and $\gamma \in (0, 1)$ is a discount factor expressing the notion that rewards further in the future are worth less to the RL agent than rewards which are closer in time.

RL is the search for policies which maximize the expectation of the goal function, namely

$$\max_{\pi} \mathbb{E}[G_t]\,. \tag{3}$$

2.2. **Value Functions and Policies.** The *action-value function* is the expected goal function, assuming we start in state s, take action a and then follow some fixed policy π from then on

$$q_\pi(s, a) := \mathbb{E}[G_t] \text{ starting from } s, \text{ taking } a, \text{ then following } \pi\,. \tag{4}$$

The *state-value function* for policy π is defined as

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \tag{5}$$

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 relationship between them given by

$$v_\pi(s) = \sum_a \pi(a \mid s) q_\pi(s, a)\,. \tag{6}$$

The state-value function defines a partial ordering on policies. Policy π is defined to be *at least as good as* π' if

$$v_\pi(s) \geq v_{\pi'}(s) \tag{7}$$

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

$$v_*(s) = \max_\pi v_\pi(s) \tag{8}$$

and optimal action-value function

$$q_*(s,a) = \max_\pi q_\pi(s,a). \tag{9}$$

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

$$q_*(s,a) = \mathbb{E}[r_{t+1} + \gamma v_*(s_{t+1}) \,|\, s_t = s, a_t = a]\,. \tag{10}$$

Furthermore, if we knew the q-function corresponding to the optimal policy, say q_*, we would know the optimal policy itself, namely

$$\text{choose } a \in \mathcal{A} \text{ to maximize } q_*(s_t,\, a)\,. \tag{11}$$

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_*.

2.3. The Bellman Equations.

> *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.*
>
> — Sutton and Barto (2018)

It is straightforward to establish that the optimal state-value function and action-value function satisfy the Bellman equations

$$v_*(s) = \max_a \sum_{s',r} p(s', r \,|\, s, a)[r + \gamma\, v_*(s')] \tag{12}$$

$$q_*(s,a) = \sum_{s',r} p(s', r \,|\, s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{13}$$

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

The basic idea of several RL algorithms is to associate the value on the right-hand side of the Bellman equation, and specifically the quantity in brackets in (13)

$$Y = r + \gamma \max_{a'} q_*(s', a') \tag{14}$$

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 do not 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.

Imagine a scenario where we simulate the underlying Markov process, or perhaps even rely on the "simulation" that is known as the "real world." Performing the computations above in our "simulation" leads to a sequence of (X, Y) pairs where

$$X_t = (s_t, a_t) \tag{15}$$

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

$$Y_t = r_t + \gamma \max_{a'} \hat{q}_t(s', a') \tag{16}$$

with $\hat{q}_t$ denoting the best approximation of q_* as it existed at the t-th time step. Then, the Bellman equation (13) implies that

$$q_*(s, a) \quad = \quad \mathbb{E}[Y \,|\, X] \,.$$

Hence determining $q_*(s, a)$ is driven by learning the association between Y and X, a supervised learning problem.

RL methods which focus on learning the association $Y = f(X) + \text{noise}$ with (X, Y) as above are referred to as *function approximation* by Sutton and Barto (2018). The function $f(X)$ in this relationship is typically non-linear as we shall see in the examples below. RL is sometimes presented as fundamentally different from supervised an unsupervised learning. Rather, we think of it as a way of directing supervised learning methods towards maximizing a longer-term goal.

3. Some Intertemporal Decision Problems in Trading and Portfolio Optimization

Many financial trading and portfolio optimization problems can be expressed in terms of value functions. In this section we review a few examples.

Example 3.1 (Replication of a derivative contract). Given a derivative contract, suppose there is a dynamic trading strategy which gives the same payout as this contract in every state of the world; such world-states are usually referred to as Arrow-Debreu states in economics. In other words, this strategy is a replicating strategy. Suppose the policy π is to follow the

replicating strategy. Then the no-arbitrage price of the option is

$$v_\pi(s_0) \tag{17}$$

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-Merton (BSM) or lognormal models.

We note that in Example 3.1 the policy is a trading strategy. A generalized version of this example applies in the presence of transaction costs when perfect replication is not possible. In these situations, one seeks the policy which maximizes a mean-variance form (or another desired utility function) which includes expected cost and a penalty for the hedging variance. The hedging variance is nonzero whenever the hedge is not perfectly replicating the derivative. We shall discuss an example of this form in greater detail in section 7.

Example 3.2 (Optimal order execution). Let us consider liquidating an order of size X using the framework of Almgren and Chriss (1999) and Almgren and Chriss (2001). Almgren and Chriss suggest to determine the sequence of child orders by maximizing expected mean-variance utility. In this case, the value function is the expected integrated revenue and variance

$$v_\pi(s) = \mathbb{E}\left[\int_0^T \Big(x_t r_t - \frac{\lambda}{2} x_t^2 \sigma_t^2 - f(\dot{x}_t)\Big) dt \;\middle|\; x_0 = s\right] \tag{18}$$

where $f(\dot{x}_t)$ is some function of the time-derivative $\dot{x}_t := dx_t/dt$ approximating market impact.

Example 3.3 (Dynamic portfolio rebalancing with time-dependent return predictions and market impact costs). Buy-side quant traders are interested in maximizing their expected utility of wealth subject to trading costs. Using mean-variance utility results in the model by Gârleanu and Pedersen (2013). This approach is conceptually similar to that of optimal order liquidation above, but with the added complexity of time-varying return predictions ("alpha").

To use this approach in practice requires three models: (1) a model of expected returns, (2) a risk model which forecasts portfolio variance and (3) a (pre-trade) transaction cost model. Gârleanu and Pedersen (2013) require the transaction cost model to be quadratic and return predictions to be autoregressive. Kolm and Ritter (2015) present and solve a more

general model that allows for non-linear transaction costs and general return predictions. Solution techniques for these problems are likely also useful in Bayesian statistics and vice versa. For example, the model of Kolm and Ritter (2015) was further generalized by Irie and West in a 2019 paper (Irie and West, 2019), which gave birth to the technique known as *Bayesian emulation*.

In the setting of multi-period portfolio selection, RL methods can in principle be applied without directly estimating any of these three models, or they can be applied in cases where one has one model but not all three. For example, given a security return prediction model, ML techniques can be used to infer the optimal strategy without directly estimating the cost function.

Example 3.4 (Portfolio allocation subject to capital gains taxes). Example 3.3 can be extended to include capital gains and other taxes, resulting in a dynamic portfolio rebalancing model for after-tax investing (see, for example, Constantinides (1984), Garlappi, Naik, and Slive (2001), Dammon, Spatt, and Zhang (2004), DeMiguel and Uppal (2005), and Haugh, Iyengar, and Wang (2016)). Just as in the previous example, RL methods can in principle be applied to solve these after-tax problems.

4. Economically Motivated Reward Signals

RL possesses a rather obvious connection with games and game theory and hence with game-theoretic approaches to economics. The player is the "agent" and the "environment" consists of the other players or the simulated environment of the game. For many games including backgammon, Go and some video games, the best player in the world is an AI trained using reinforcement learning (see, for example, Mnih et al. (2013), Mnih et al. (2015), Tian and Zhu (2015), and Silver et al. (2017)).

Von Neumann and Morgenstern (1945) show that if a decision-maker

(i) is faced with risky (probabilistic) outcomes of different choices and
(ii) has preferences satisfying four axioms of "rational behavior"

then their choices can be determined by maximizing the expected value of some function, u, called the utility function, defined over the potential outcomes. The utility function quantifies the decision-maker's preferences for different outcomes. This function is typically concave, increasing and differentiable. When one applies expected utility theory to the trading of

financial securities, the outcomes are typically different levels of wealth w_T at some future time T. A rational agent would then maximize the expected utility of future wealth, $\mathbb{E}[u(w_T)]$.

In finance, an *optimal trading strategy* is usually defined as a strategy that maximizes expected utility of future wealth, where future wealth is the sum of a number of wealth increments over shorter time periods. Hence, to find the optimal trading strategy we seek to solve the problem

$$\max \mathbb{E}[u(w_T)] \tag{19}$$

$$\text{where } w_T = w_0 + \sum_{t=1}^{T} \delta w_t \,. \tag{20}$$

Here $\delta w_t := w_t - w_{t-1}$ is the wealth increment from $t-1$ to t and w_0 denotes initial wealth.

It is well-known that under certain assumptions on the return distribution, maximizing expected utility is equivalent to maximizing a mean-variance form of the problem (Chamberlain, 1983). The main assumption for this to be justified is that the isoprobability contours of the return distribution exhibit elliptical symmetry. Note that these so-called *elliptical distributions* comprise a large class of distributions that includes many 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 (19) is equivalent to

$$\max \ \mathbb{E}[w_T] - \frac{\kappa}{2}\mathbb{V}[w_T] \tag{21}$$

Suppose now that we could invent some definition of reward, R_t, so that

$$\mathbb{E}[w_T] - \frac{\kappa}{2}\mathbb{V}[w_T] \approx \sum_{t=1}^{T} R_t \,. \tag{22}$$

Then (21) would become a "cumulative reward over time" problem that we can solve through RL and maximizing the right-hand side of (22) is equivalent to maximizing average reward.

Among the various algorithms provided by RL, a number of them seek to maximize $\mathbb{E}[G_t]$ where

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \tag{23}$$

which, by (22), would then maximize approximate expected utility as long as $\gamma = 1 - \epsilon$ with ϵ small.

Let us consider the reward function

$$R_t := \delta w_t - \frac{\kappa}{2}(\delta w_t - \hat{\mu})^2 \tag{24}$$

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

$$\frac{1}{T}\sum_{t=1}^{T} R_t = \underbrace{\frac{1}{T}\sum_{t=1}^{T} \delta w_t}_{\to \mathbb{E}[\delta w_t]} - \frac{\kappa}{2}\underbrace{\frac{1}{T}\sum_{t=1}^{T}(\delta w_t - \hat{\mu})^2}_{\to \mathbb{V}[\delta w_t]} \tag{25}$$

for large T, such that the two terms on the right hand side approach the sample mean and variance, respectively. Thus with this choice of reward function (24), if the RL agent learns to maximize cumulative reward it should also approximately maximize the mean-variance form of utility.

Strictly speaking, what appears on the left side of equation (25) is *average reward*, and not cumulative discounted reward. Of course, if the process is stationary then a policy which is the optimal cumulative discounted reward should also have favorable average-reward properties. Nonetheless, as is clear from this formulation, we naturally have a preference for average reward rather than discounted reward as the goal G_t.

Following Sutton and Barto (2018), in the average reward setting the "quality" of a policy π is defined as

$$\begin{aligned} r(\pi) &:= \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[R_t \,|\, S_0, A_{0:t-1} \sim \pi] && (26)\\ &= \lim_{t\to\infty} \mathbb{E}[R_t \,|\, S_0, A_{0:t-1} \sim \pi] && (27)\\ &= \sum_s \mu_\pi(s) \sum_a \pi(a \,|\, s) \sum_{s',r} p(s', r \,|\, s, a) r\,. && (28) \end{aligned}$$

Here, μ_π denotes the steady-state distribution, which is assumed to exist for any π and to be independent of S_0. This particular assumption about the MDP is known as *ergodicity*. Ergodicity means intuitively that where the MDP starts, or any early decision made by the RL agent, can have only a temporary effect.

We can now 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) suggest 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

$$G_t := R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + \dots \tag{29}$$

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 γ's and replace all rewards by the difference between the reward and the true average reward. We refer to Sutton and Barto (2018) for details.

5. States and Actions for Trading Problems

The state variable s_t is a data structure which intuitively must contain everything the RL agent needs to make a trading decision and nothing else. Candidate state variables include:

(i) the current position or holding in the security,
(ii) the values of any signals which are believed to be predictive,
(iii) the current state of the market, including current price and any relevant microstructure / limit-order book details, and
(iv) if contingent claims are involved, additional variables such as time to expiry to properly define the contract.

In trading problems, the most obvious choice for the action a_t is the number of securities to trade. If the RL 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 RL agent could decide which execution algorithm to use, whether to cross the spread or be passive, its target participation rate, etc. If one of the securities is a contingent claim, there may be additional actions available, such as early exercise or, more generally, a pre-specified set of dates or other conditions at which the user can exercise various kinds of optionality.

One must choose whether to use mathematical methods which require a finite state space and enumeration of all possible states (such as Watkins Q-learning), or methods which allow a continuous state space. From our discussion above, we infer that in trading problems the state vector will

typically be an object which is "inconveniently" high dimensional. For example, with k variables describing the current state of the limit order book and n predictive signals we obtain a state space which is naturally embedded within $\mathbb{R}^{n+k}$. There is no natural way to represent $\mathbb{R}^{n+k}$ as a finite enumeration of state vectors and, moreover, any method requiring an enumeration of state vectors is unlikely to scale well as $n+k$ becomes large. Therefore, for realistic trading applications we focus on continuous state-space methods. Tabular methods which enumerate all states are useful for simple proof-of-concept applications and for classroom examples.

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

$$X_t = (s_t, a_t), \quad Y_t = r_t + \gamma \max_{a'} \hat{q}_t(s', a') \tag{30}$$

with $\hat{q}_t$ denoting the best approximation of q_* as it existed at the t-th time step. Function approximation methods learn the unknown function f where $Y = f(X) + \text{noise}$. This is of course the well-known nonlinear regression problem in statistics that can be solved by many methods including artificial neural networks, basis functions, etc. For many regression methods, even nonlinear ones, the representation of X as an n-dimensional vector is not problematic, even for large n. For a review of nonlinear regression techniques which apply to the function estimation problem discussed here, we refer to Friedman, Hastie, and Tibshirani (2001).

6. Trading Mean-Reversion with RL

In this section, we expand upon an example originally given by Ritter (2017) using RL to trade mean-reversion. We extend Ritter (2017) by introducing a continuous state space formulation and providing an explicit graphical representation of the resulting value function. We will see later from the value function that the RL agent has learned the existence of a no-trade zone in a neighborhood around the equilibrium price. For background and more details about this problem, we refer to the original article and the references therein.

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

$$dx_t = -\lambda x_t + \sigma\, \xi_t \tag{31}$$

where $\xi_t \sim N(0,1)$ and ξ_t, ξ_s are independent when $t \neq s$ and $\sigma > 0$. This means that p_t tends to revert to its long-run equilibrium level p_e with mean-reversion rate $\lambda > 0$. Note that these assumptions imply existence of an alpha strategy. In particular, positions taken in the appropriate direction while away from equilibrium have small probability of loss and extremely asymmetric loss-gain profiles.

Initially, we do not provide the RL agent with any knowledge of the price dynamics. Hence, it does not know λ, σ, or even that some dynamics of the form (31) are valid.

The RL 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. In this article we will use the following representation of the spread cost

$$\text{SpreadCost}(\delta n) = \text{TickSize} \cdot |\delta n| \,. \tag{32}$$

In addition, we assume that there is price impact in our economy which has a linear functional form. Specifically, each round lot traded is assumed to move the price one tick, hence leading to a dollar cost of $|\delta n_t| \times \text{TickSize/LotSize}$ per share traded, for a total dollar cost for all shares

$$\text{ImpactCost}(\delta n) = (\delta n)^2 \times \text{TickSize/LotSize} \,. \tag{33}$$

Taken together, the total trading cost becomes

$$\text{Cost}(\delta n) = \text{multiplier} \times (\text{SpreadCost}(\delta n) + \text{ImpactCost}(\delta n) \,. \tag{34}$$

This is a common trading cost specification both in practice as well as in the academic literature (see, for example Almgren and Chriss (2001) and Almgren (2003)). The multiplier allows us to easily test the RL agent in regimes of more or less liquidity.

We will use the following state representation of the environment

$$s_t = (p_t, n_{t-1}) \tag{35}$$

where p_t is the price of the security and n_{t-1} denotes the RL agent's position coming into the period, in number of shares. In contrast to Ritter (2017), here we represent the state vector as a vector in a (continuous) Euclidean space – $\mathbb{R}^2$ in this problem. This allows us to use the nonlinear regression techniques discussed above to learn the association between X and Y that

is the action-value function. We use an ensemble of model trees as our nonlinear regression learner, but that is certainly not the only possible choice – it is likely that an artificial neural network would work as well. For an overview of continuous state and action space approaches see, for example, Van Hasselt (2012), Nichols (2014), and Sutton and Barto (2018).

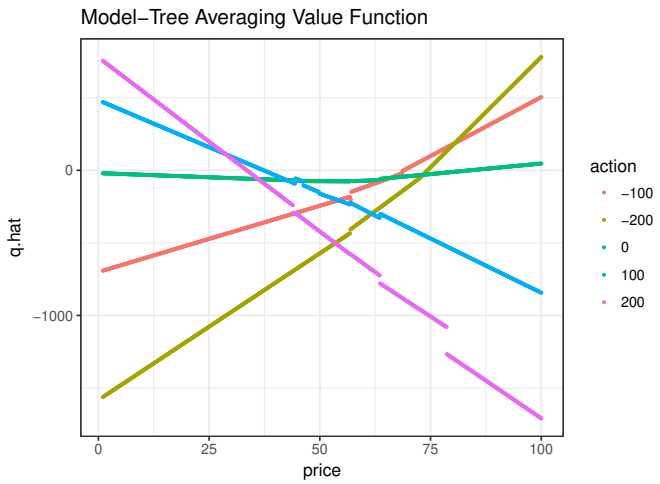

FIGURE 2. The learned value function $p \to \hat{q}((0,p),a)$ for various actions a, where $\hat{q}$ is estimated by using continuous state-space methods. The red, gold, green, blue and purple lines represent the action of trading $-100, -200, 0, 100$ and 200 shares respectively. The resulting decision at each price level (assuming zero initial position) is obtained by taking the maximum of the piecewise-linear functions.

Figure 2 depicts the learned value function. The relevant decision at each price level (assuming zero initial position) is the maximum of the various piecewise-linear functions shown in the figure. The RL agent has learned the existence of a no-trade region in the center, where the zero-trade action line is the maximum. Notice that there are regions on both sides 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.

To evaluate the RL agent out-of-sample, we record its performance from trading on $5,000$ new samples from the stochastic process (31). Figure 6 shows the RL agent's P/L.

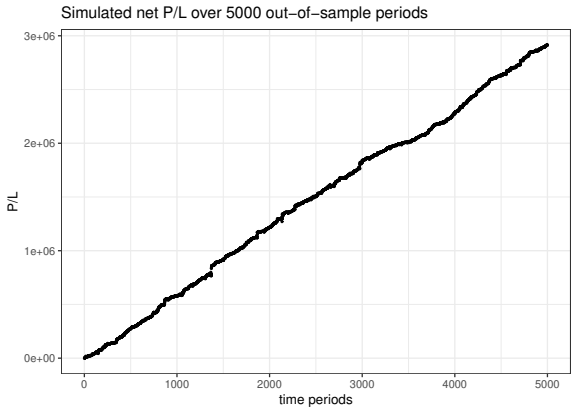

FIGURE 3. Cumulative simulated out-of-sample P/L of trained model.

We believe these results are encouraging: we constructed a system wherein we know there is an alpha strategy, analogous to a game where winning strategies exist. Our results demonstrates that (i) a RL agent can learn to play this game and (ii) it learns a reasonable strategy.

We think 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 the framework presented here, an arbitrage is defined as a high-Sharpe trading strategy after trading costs. This suggests the following no-arbitrage theorem: *there is no arbitrage if and only if the optimally-trained RL agent cannot achieve a high Sharpe ratio.*

7. Optimal Hedging of Derivatives with RL

In this section, we explore another application of interest to traders that fits within the general framework presented in Section 4: the replication and hedging of an option position. This is a fundamental problem in finance. Since the seminal work of Black and Scholes (1973) and Merton (1973) on option pricing and dynamic hedging (jointly referred to as BSM), a vast number of articles have addressed the problem of optimal replication and hedging under more general assumptions.[3] The core idea of BSM is that in a complete and frictionless market there exists a continuously rebalanced

[3]While 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 subsequently followed by others; see Kolm and Ritter (2019) for a discussion.

dynamic trading strategy in the stock and riskless security that perfectly replicates the option.

Of course, in practice continuous trading of arbitrarily small amounts of stock is prohibitively costly. Therefore, the portfolio replicating the option is rebalanced at discrete times to minimize trading costs. As a consequence, perfect replication is impossible and an optimal hedging strategy depends on the desired trade-off between replication error and trading costs. That is to say that the hedging strategy chosen by an RL agent depends on their risk aversion.

We look at the simplest possible hedging 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 RL 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. We note that a version of the model below appeared in Kolm and Ritter (2019).

For European options, the state must minimally contain the current price S_t of the underlying and the time to expiration

$$\tau := T - t > 0\,, \tag{36}$$

as well as the RL agent's current position of n shares. Thus, the state is an element of

$$\mathcal{S} := \mathbb{R}_+^2 \times \mathbb{Z} = \{(S, \tau, n) \mid S > 0, \tau > 0, n \in \mathbb{Z}\}. \tag{37}$$

We stress that the state *does not* need to contain the option Greeks, as they are (nonlinear) functions of the variables the RL agent has access to via the state. We expect it 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 and similar models.

First, we consider an economy without trading costs and answer the question of whether it is possible for a RL agent to learn what we teach students in their first semester of business school: the formation of the dynamic replicating portfolio strategy. Unlike our students, the RL agent can only learn by observing and interacting with simulations.

We put the RL agent at a disadvantage by not letting it know any of the following pertinent pieces of information: (i) the strike price K, (ii) that the stock price process is a geometric brownian motion (GBM), (iii) the

volatility of the price process, (iv) the BSM formula, (v) the payoff function $(S-K)_+$ at maturity and (vi) any of the Greeks. The RL agent must infer the relevant information concerning these variables, insofar as it affects the value function, by interacting with a simulated environment.

Each out-of-sample simulation of the GBM is different. Figure 4 shows a typical example of the trained agent's performance.

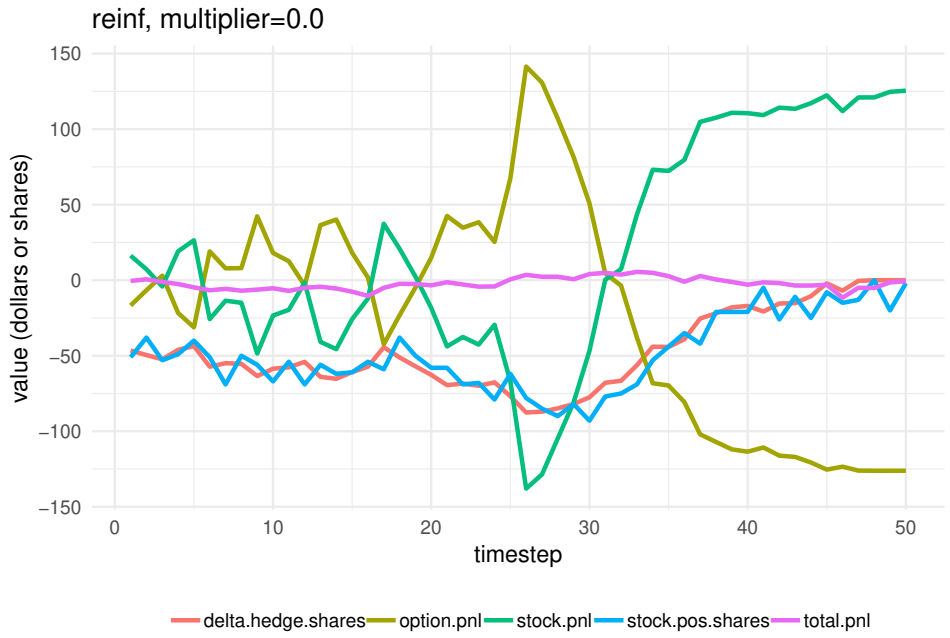

FIGURE 4. Comparison of a simple delta-hedging agent with (out-of-sample) simulation of a trained RL agent. We depict cumulative stock, option and total P/L; RL agent's position in shares (stock.pos.shares), and $-100 \cdot \Delta$ (delta.hedge.shares). Observe that (a) cumulative stock and options P/L 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 the agent was not provided with a computation of the delta.

A key strength of the RL approach is that it does not make any assumptions about the form of the cost function (34). It will learn to optimize expected utility, under whatever cost function one provides.

As we need a baseline, we define π_{DH} to be the policy which always trades to hedge delta to zero according to the BSM model, rounded to the nearest integer number of shares. 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 expected mean-variance utility (21), 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 generated $N = 10,000$ out-of-sample simulations and 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 5 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 6 shows the RL agent – we see that, while maintaining a hedge, the RL agent is trading in a cost-conscious way.

The curves in Figure 5, representing the RL agent's position (stock.pos.shares), are much smoother than the value of $-100 \cdot \Delta$ (called delta.hedge.shares in Figure 5), which naturally fluctuates along with the GBM process.

FIGURE 5. Out-of-sample simulation of a baseline agent who uses policy "delta" or π_{DH}. 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. The right-hand plot shows only the two most relevant curves from the left.

FIGURE 6. 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. The right-hand plot shows only the two most relevant curves from the left.

Above we could only show a few representative runs taken 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. Figure 7 shows kernel density estimates (basically, smoothed histograms) of total costs and volatility of total P/L of all paths. 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.

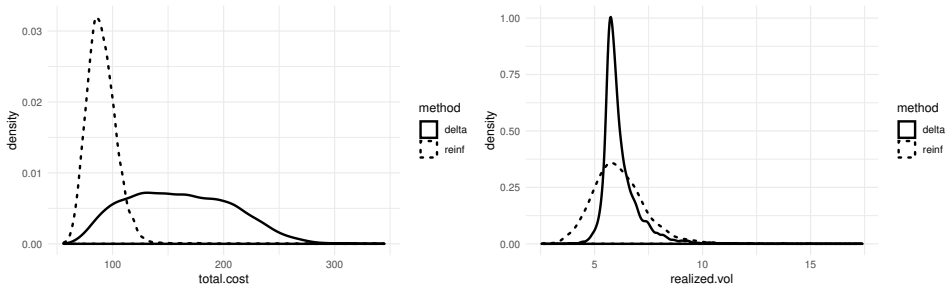

FIGURE 7. Kernel density estimates for total cost (left panel) and volatility of total P/L (right panel) from $N = 10,000$ out-of-sample simulations. Policy "delta" is π_{DH}, while policy "reinf" is the greedy policy of an action-value function trained by RL. The "reinf" policy achieves much lower cost (t-statistic $= -143.22$) with no significant difference in volatility of total P/L.

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 figure 8. The "reinf" method is seen to outperform as its t-statistic is much more often close to zero and insignificant.

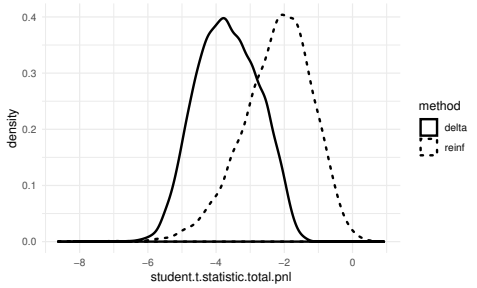

FIGURE 8. Kernel density estimates of the t-statistic of total P/L for each of our out-of-sample simulation runs and for both policies represented above ("delta" and "reinf"). The "reinf" method is seen to outperform in the sense that the t-statistic is much more often close to zero and insignificant.

Lastly, we recall the well-known fact that, in continuous time, the optimal replicating portfolio strategy for a derivative contract is a solution to a nonlinear PDE given by the Feynman-Kac theorem (see Simon (1979) for an exposition). Therefore, one might guess that if reinforcement learning can determine the optimal replicating portfolio strategy in an approximate

numerical sense, then it is performing numerical solution of the resulting (Feynman-Kac) PDE. Indeed, Weinan, Han, and Jentzen (2017) find that using deep reinforcement learning (RL in which the function approximation is done by means of a neural network) can provide accurate numerical solutions of certain parabolic PDEs. What we have done above is essentially to solve the Black-Scholes PDE numerically using RL.

8. Conclusions

In this article we showed how reinforcement learning (RL) – machine learning models that enable agents to learn to make a sequence of decisions through "trial and error" incorporating feedback from its actions and experiences – can be used to solve modern financial applications of intertemporal choice. In finance, common problems of this kind include pricing and hedging of contingent claims, investment and portfolio allocation, buying and selling a portfolio of securities subject to transaction costs, market making, asset liability management and optimization of tax consequences, to name a few.

Option hedging, optimal execution and optimal trading of alpha forecasts all share the property that the value function is the expected integrated revenue and variance

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

where $f(\dot{x}_t)$ is some function of the time-derivative $\dot{x}_t := dx_t/dt$ approximating market impact.

RL allows us to solve these dynamic optimization problems in a close to "model-free" way, relaxing the assumptions often needed for dynamic programming (DP) approaches. In finance, underlying stochastic dynamics are often complex and therefore difficult to derive or estimate correctly. Additionally, realistic specification of microstructure effects and transaction costs add an additional layer of complexity due to their nonlinear and non-differentiable behavior. In this article, we contended that RL shows great promise in addressing these issues in a general and flexible way.

Weinan, Han, and Jentzen (2017) find that deep reinforcement learning gives new algorithm for solving parabolic partial differential equations (PDEs) and backward stochastic differential equations (BSDEs) in high dimension. The PDEs include the typical Feynman-Kac PDEs that appear

in option pricing, so it is perhaps not surprising that RL agents can learn to hedge options. This technique is especially promising because it immediately generalizes to more realistic problems including transaction costs. Computationally, deep reinforcement learning is easily implemented with the help of existing numerical libraries such as TensorFlow.

While RL is promising for financial applications it is not without challenges, including:

- Correct specification of the model: this entails the representation of the state, choice of the actions and design of the reward. Each one of these is problem specific, both in terms of specification as well as in difficulty.
- Acquisition of sufficient data for training and online training: the majority of the most successful non-finance RL applications in recent years utilize a simulation of the environment to generate training data as opposed to training on historical data (see, for example, the Mnih et al. (2013), Mnih et al. (2015), and Silver et al. (2017) who use RL systems to play video games and Go on a super-human level). A clear benefit of simulation is that data scarcity is not an issue. That said, we believe it will be important for financial applications to develop RL techniques that can train on historical and online data. The traditional way of training on scarce historical data is achieved by first developing a stochastic model of the environment with relatively few parameters, fitting those few parameters to historical data and then training a RL system on as many simulations as needed from the stochastic model. In effect, one uses a stochastic model to "impute the missing data" from the environment. While this can serve as a starting point, training then becomes dependent on the assumptions and specifications of the chosen stochastic model. More general and flexible data models are warranted.
- Generalization of RL models: There is a growing body of research in the RL community addressing the need to take agents trained for one task and adapt them for a similar task, without having to start from scratch. Some approaches include transfer learning (see, for example, Taylor and Stone (2009)) and progressive models (see, for example, Rusu et al. (2016)), to name a few. These and related techniques are likely to be useful also in financial applications.

We expect in the coming years to see a lot of exciting new results connecting the fields of finance, machine learning and numerical solutions of PDEs; these fields all share reinforcement learning as the common thread which connects them.

References

Almgren, Robert and Neil Chriss (1999). "Value under liquidation". In: *Risk* 12.12, pp. 61–63.

– (2001). "Optimal execution of portfolio transactions". In: *Journal of Risk* 3, pp. 5–40.

Almgren, Robert F (2003). "Optimal execution with nonlinear impact functions and trading-enhanced risk". In: *Applied mathematical finance* 10.1, pp. 1–18.

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

Black, Fischer and Myron Scholes (1973). "The pricing of options and corporate liabilities". In: *Journal of Political Economy* 81.3, pp. 637–654.

Chamberlain, Gary (1983). "A characterization of the distributions that imply mean–variance utility functions". In: *Journal of Economic Theory* 29.1, pp. 185–201.

Constantinides, George M (1984). "Optimal stock trading with personal taxes: Implications for prices and the abnormal January returns". In: *Journal of Financial Economics* 13.1, pp. 65–89.

Dammon, Robert M, Chester S Spatt, and Harold H Zhang (2004). "Optimal asset location and allocation with taxable and tax-deferred investing". In: *The Journal of Finance* 59.3, pp. 999–1037.

DeMiguel, Victor and Raman Uppal (2005). "Portfolio investment with the exact tax basis via nonlinear programming". In: *Management Science* 51.2, pp. 277–290.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2001). *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin.

Garlappi, Lorenzo, Vasant Naik, and Joshua Slive (2001). "Portfolio selection with multiple assets and capital gains taxes". In: *Available at SSRN 274939*.

Gârleanu, Nicolae and Lasse Heje Pedersen (2013). "Dynamic trading with predictable returns and transaction costs". In: *The Journal of Finance* 68.6, pp. 2309–2340.

Haugh, Martin, Garud Iyengar, and Chun Wang (2016). "Tax-aware dynamic asset allocation". In: *Operations Research* 64.4, pp. 849–866.

Intriligator, Michael D (2002). *Mathematical optimization and economic theory*. SIAM.

Irie, Kaoru and Mike West (2019). In: *Bayesian Analysis* 14.1, pp. 137–160.

Kolm, Petter N and Gordon Ritter (Mar. 2015). "Multiperiod portfolio selection and bayesian dynamic models". In: *Risk* 28.3, pp. 50–54.

– (2019). "Dynamic replication and hedging: A reinforcement learning approach". In: *The Journal of Financial Data Science* 1.1, pp. 159–171.

Leland, Hayne E (1985). "Option pricing and replication with transactions costs". In: *The Journal of Finance* 40.5, pp. 1283–1301.

Merton, Robert C (1973). "Theory of rational option pricing". In: *The Bell Journal of Economics and Management Science*, pp. 141–183.

Mnih, Volodymyr et al. (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602*.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, p. 529.

Nichols, Barry D (2014). "Reinforcement learning in continuous state-and action-space". PhD thesis. University of Westminster.

Ritter, Gordon (2017). "Machine Learning for Trading". In: *Risk* 30.10, pp. 84–89. URL: `https://ssrn.com/abstract=3015609`.

Rusu, Andrei A et al. (2016). "Sim-to-real robot learning from pixels with progressive nets". In: *arXiv preprint arXiv:1610.04286*.

Silver, David et al. (2017). "Mastering the game of go without human knowledge". In: *Nature* 550.7676, pp. 354–359.

Simon, Barry (1979). *Functional integration and quantum physics*. Vol. 86. Academic press.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. Second edition. MIT press Cambridge. URL: `http://incompleteideas.net/book/the-book.html`.

Taylor, Matthew E and Peter Stone (2009). "Transfer learning for reinforcement learning domains: A survey". In: *Journal of Machine Learning Research* 10.Jul, pp. 1633–1685.

Tian, Yuandong and Yan Zhu (2015). "Better computer go player with neural network and long-term prediction". In: *arXiv preprint arXiv:1511.06410*.

Van Hasselt, Hado (2012). "Reinforcement learning in continuous state and action spaces". In: *Reinforcement learning*. Springer, pp. 207–251.

Von Neumann, John and Oskar Morgenstern (1945). *Theory of games and economic behavior*. Princeton University Press Princeton, NJ.

Weinan, E, Jiequn Han, and Arnulf Jentzen (2017). "Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations". In: *Communications in Mathematics and Statistics* 5.4, pp. 349–380.