Reinforcement Learning With Continuous States

Gordon Ritter and Minh Tran

Two major challenges in applying reinforcement learning to trading are: handling high-dimensional state spaces containing both continuous and discrete state variables, and the relative scarcity of real-world training data. We introduce a new reinforcement-learning method, called supervised-learner averaging, that simultaneously solves both problems, while outperforming Q-learning on a simple baseline.

Introduction

Recently we showed that reinforcement learning can be applied to discover arbitrage opportunities, when they exist (Ritter, 2017). Under Ornstein-Uhlenbeck dynamics for the log-price process, even with trading costs, a reinforcement-learning algorithm was able to discover a high-Sharpe-ratio strategy without being told what kind of strategy to look for.

According to economic theory going back to Arrow (1963) and Pratt (1964), optimal traders maximize expected *utility* of wealth, and not expected wealth. The purpose of our previous work on this topic was primarily to argue that, in light of the modern understanding of utility theory, reinforcement learning systems for trading applications should use reward functions that converge to utility of wealth (or an equivalent mean-variance form of utility). The simpler alternative, maximizing expected wealth, cannot possibly maximize Sharpe ratio, nor can it account for investors' heterogeneous levels of risk tolerance.

The purpose of Ritter (2017) was not to investigate the methodology of how reinforcement learning is accomplished; in fact, the simplest possible methodology (tabular Q-learning) was used. Such methods represent the action-value function by a lookup table, usually implemented as a matrix. Advantages of this approach include that it is very simple to implement, and the way the system learns from new data is very easy to interpret. By the theorem of Robbins and Siegmund (1985), the method is known to converge under certain asymptotic bounds on its parameters.

Nonetheless, tabular methods are severely limited by the curse of dimensionality. Tabular methods require that the state space $\mathcal{S}$ be finite, and standard implementations typically further assume that an array of length $|\mathcal{S}|$ fits in computer memory, with similar requirements for the action space. They require enough training time to visit each state. If the state space is $\mathbb{R}^k$ with k large, or even a discrete k-dimensional lattice, then those memory requirements won't scale as k increases, hence the term "curse of dimensionality."

We now explain in more detail the main application, which is to multi-period trading problems with costs, and argue that the curse of dimensionality will render tabular methods inadmissible for all but the simplest problems.

In trading applications, the goal is usually to train an agent to interact in an electronic limit-order-book market. Each limit-order book for a given security has a *tick size*, defined to be the smallest permissible non-zero price interval between different orders.

The space $\mathcal{A}$ of available actions in the limit-order book for a single security is limited to placing quotes at one-tick intervals near, or inside, the current inside market (best bid and offer). One could also consider different order types as part of the action, but in any case $\mathcal{A}$ is naturally a small finite set, easily stored in computer memory. By contrast, the most natural representation of the state space is an embedding within $\mathbb{R}^k$ where k is moderate to large, as we now explain.

The term *state*, in reinforcement learning problems, usually refers to the *state of the environment*. In trading problems, the "environment" should be interpreted to mean all processes generating observable data that the agent will use to make a trading decision. Let s_t denote the state of the environment at time t; the state is a data structure containing all of the information the agent will need in order to decide upon the action. This will include the agent's current position, which is clearly an observable that is an important determinant of the next action.

At time t, the state s_t must also contain the prices p_t, but beyond that, much more information may be consid-

ered. In order to know how to interact with the market microstructure and what the trading costs will be, the agent should observe the bid-offer spread and liquidity of the instrument. Any predictive signals must also be part of the state, or else they cannot influence the decision.

This means that even a discretization of the true problem involves a k-dimensional lattice in $\mathbb{R}^k$. Consequently, any algorithm that needs to either visit a representative sample of states, or to store a state vector as an array, is intrinsically non-scalable, and will become intractable for moderate to large k.

Further progress requires a method that allows many real-valued (continuous and/or discrete) predictors to be included in the state. Furthermore, the method must handle non-linear and non-monotone functional forms for the value function. Another desirable property is efficient sample use, by which we mean, roughly, the ability to converge to a useful model on relatively small training sets. This is desirable when applying the model to real data, or when training time is a bottleneck. A final desirable property is that the new method should outperform Q-learning on the baseline problem presented by Ritter (2017).

In this paper we present a reinforcement-learning method, which we call supervised-learner averaging (SLA), and show that it has all of the desirable properties listed above. The method is likely to have broad applicability to a wide range of machine learning problems, but this paper is concerned primarily with the application to trading of illiquid assets.

Value Functions

> *The key idea of reinforcement learning, generally, is the use of value functions to organize and structure the search for good policies.*
>
> —Sutton and Barto (2018)

The foundational treatise on value functions was written by Bellman (1957), at a time when the phrase "machine learning" was not in common usage. Nonetheless, reinforcement learning owes its existence, in part, to Richard Bellman.

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

In a *Markov* decision process, once we observe the current state of the system, we have the information we need to make a decision. In other words, (*assuming* we know the current state), then it would not help us (ie. we could not make a better decision) to also know the full history of past states which led to the current state. This history-independence (or memoryless property) is closely related to Bellman's principle:

> *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*
>
> —Bellman (1957)

Following the notation of Sutton and Barto (2018), the sequence of rewards received after time step t is denoted $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$. The agent's goal is to maximize the expected cumulative reward, denoted by

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

The agent then searches for policies which maximize $\mathbb{E}[G_t]$. The sum in (1) can be either finite or infinite. The constant $\gamma \in [0, 1]$ is known as the *discount rate*, and is especially useful in considering the problem with $T = \infty$, in which case γ is needed for convergence.

There are principally two kinds of value functions in common usage; at optimality, one is a maximization of the other. The *state-value function* for policy π is

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

where $\mathbb{E}_\pi$ denotes the expectation under the assumption that policy π is followed. Similarly, the *action-value function* expresses the value of starting in state s, taking action a, and then following policy π thereafter:

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

Policy π is defined to be at least as good as π' if $v_\pi(s) \geq v_{\pi'}(s)$ for all states s. An *optimal policy* is defined to be one which is at least as good as any other policy. There need not be a unique optimal policy, but all optimal policies share the same optimal state-value function $v_*(s) = \max_\pi v_\pi(s)$ and optimal action-value function $q_*(s, a) = \max_\pi q_\pi(s, a)$. Also note that v_* is the maximization over a of q_*.

Let $p(s', r \,|\, s, a)$ denote the probability that the Markov decision process transitions to state s' and the agent receives reward r, conditional on the event that the Markov process was previously in state s and, in that

state, the agent chose action a. The optimal state-value function and action-value function satisfy Bellman optimality equations

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

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

where the sum over s', r denotes a sum over all states s' and all rewards r. In a continuous formulation, these sums would be replaced by integrals.

If we possess a function $q(s,a)$ which is an estimate of $q_*(s,a)$, then the *greedy policy* is defined as picking at time t the action a_t^* which maximizes $q(s_t, a)$ over all possible a, where s_t is the state at time t. Convergence of policy iteration requires that, in the limit as the number of iterations is taken to infinity, every action will be sampled an infinite number of times. To ensure this, standard practice is to use an *ϵ-greedy policy*: with probability $1 - \epsilon$ follow the greedy policy, while with probability ϵ uniformly sample the action space.

Given the function q_*, the greedy policy is optimal. Hence any iterative method which converges to q_* constitutes a solution to the original problem of finding the optimal policy.

General Policy Iteration

Let π be any deterministic policy, not necessarily the optimal one. Let π' be any other deterministic policy having the property that,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \text{ for all } s \in \mathcal{S}.$$

Then the policy π' must be as good as, or better than, π; this is called the *policy improvement theorem.*

Generalized policy iteration (GPI) generally refers to a broad class of reinforcement-learning algorithms which let policy evaluation and policy improvement processes interact. Moreover,

> *...if both the evaluation process and the improvement process stabilize, that is, no longer produce changes, then the value function and policy must be optimal. The value function stabilizes only when it is consistent with the current policy, and the policy stabilizes only when it is greedy with respect to the current value function. Thus, both processes stabilize only when a policy has been found that is greedy with respect to its own evaluation function. This implies that the Bellman optimality equation holds.*
>
> —Sutton and Barto (2018)

In what follows we shall describe a new kind of GPI in which the action-value function is represented internally by a model-averaging procedure applied to a sequence of supervised-learning models.

Model-based Policy Iteration

We start with a given function $\hat{q}$ which represents the current estimate of the optimal action-value function; this estimate is often initialized to be the zero function, and will be refined as the algorithm continues. We also start with a policy π that is defined as the ϵ-greedy policy with respect to $\hat{q}$. Note the distinction between $\hat{q}$, which denotes our current estimate of the optimal action-value function, and q_π which represents the true action-value function of the policy π.

Let S_t be the state at the t-th step in the simulation. An action A_t is chosen according to the policy π which is ϵ-greedy w.r.t. $\hat{q}$. Let $X_t := (S_t, A_t)$ be the state-action pair. The *update target* Y_t can be any approximation of $q_\pi(S_t, A_t)$, including the usual backed-up values such as the full Monte Carlo return or any of the n-step Sarsa returns discussed by Sutton and Barto (2018). For example, the one-step Sarsa target is

$$Y_t = R_{t+1} + \gamma\, \hat{q}(S_{t+1}, A_{t+1}) \tag{2}$$

Our $\hat{q}(s,a)$ is calculated by a form of model averaging. There is a list $\mathcal{L} = \{F_1, F_2, \ldots\}$, initialized to be empty, where the k-th element F_k is a model for predicting Y_t from $X_t = (S_t, A_t)$. The precise form of this model, and the methods for how to fit the model to a set of training data, are independent of the rest of the algorithm. In other words, *almost any supervised learning setup* can be plugged into the procedure at the point where we fit the F_j.

The working estimate of the optimal action-value function is:

$$\hat{q}(s,a) := \frac{1}{K} \sum_{k=1}^{K} F_k(s,a), \quad \text{where } K = |\mathcal{L}|. \tag{3}$$

This averages the predictions of all the supervised-learners in the model list $\mathcal{L}$. For this reason, we name the method *supervised-learner averaging*, or SLA. The full algorithm is given below.

Initially, the model list is empty, and the initial estimate of $\hat{q}(s,a)$ is zero. At the end of each batch, a new model is added to $\mathcal{L}$ which implies that said new model will be included in the model-averaging in definition (3) for all subsequent calculations of $\hat{q}$. In particular, $\hat{q}(s,a)$

is only updated when a new model is added to $\mathcal{L}$, and this only happens at the end of a batch.

As mentioned above, various supervised-learning methods may be used for the core estimation of the model $Y_t \sim F(X_t)$. For the applications below, we chose the M5' model tree method due to Quinlan (1992), and with improvements due to Wang and Witten (1997). This choice makes sense *a priori* for our examples because this family of supervised learners is well-suited to functions which are piecewise-smooth with relatively few breakpoints, and also to mixtures of continuous and discrete variates.

Definition 1. For brevity, we shall refer to SLA as *model-tree averaging* or MTA, when the supervised learner is a model tree.

Algorithm: Supervised-Learner Averaging

This section describes the algorithm we call supervised-learner averaging (SLA), and which is a member of the family of algorithms known as generalized policy iteration in reinforcement learning.

Initialize a list $\mathcal{L}$ to be empty. Repeat the following steps until the policy has converged.

1. Interact with the environment (often a simulation) for n_{batch} time-steps using the ϵ-greedy policy derived from $\hat{q}$, where $\hat{q}$ is always computed as (3), without changing the policy during the batch. Let $\mathcal{B}$ denote the collection of all instances X_t and Y_t generated in the current batch, where $X_t = (s_t, a_t)$ and Y_t is defined in (2).

2. Build a new supervised-learning model F_k suited for the prediction problem $Y = F_k(X)$, using only the samples in $\mathcal{B}$ to construct training sets, test sets and validation sets, using cross-validation (or pruning or related model-selection technology) within $\mathcal{B}$.

3. Add F_k to the list $\mathcal{L}$ and increment k. Return to step 1.

After each policy update (each time $\mathcal{L}$ is augmented), the new policy is evaluated by estimating the cumulative reward in simulation. The algorithm terminates when the policy's estimated cumulative reward stabilizes.

Quinlan (2014) discusses the use of *committees* in a supervised-learning setup. In the context of building a classifier, this is directly analogous to the human concept of committee: each "committee member" classifies the instance, which is taken as a "vote" for which category it belongs to. In the context of predicting a continuous variable, the use of committees can be considered roughly analogous to the model-averaging method described above, but Quinlan is working in a supervised-learning setup whereas we have adapted the concept to reinforcement learning. In reinforcement learning, one is naturally driven to the use of committees because generalized policy iteration (GPI) produces a sequence of data sets, and each data set can potentially be used to improve the policy by adding one new committee member, and each new member is actually trained on a new policy.

Trading a Very Illiquid Asset

As a numerical example to elucidate model-based policy iteration, we study the same simulated market as in Ritter (2017), but we change the cost function to simulate a very illiquid asset. This allows us to better illustrate some interesting features which are artifacts of high trading cost, eg. the *no-trade zone*, defined to be a region in price space over which, starting from zero, it would never be optimal to trade.

Fifty-five years of theory since Arrow (1963) suggest that we train the learner to optimize expected utility of final wealth, $\max \mathbb{E}[u(w_T)]$ for an increasing, concave utility function $u : \mathbb{R} \to \mathbb{R}$. By mean-variance equivalence for elliptical distributions, in the examples below it is a mathematical fact that for some $\kappa > 0$, we can equivalently maximize the mean-variance quadratic form

$$\mathbb{E}[w_T] - (\kappa/2)\mathbb{V}[w_T]. \tag{4}$$

The parameter κ is a local representation of the trader's risk aversion around the current wealth level. In the examples below $\kappa = 10^{-4}$.

For a reinforcement learning approach to match (4) we need R_t to be an appropriate function of wealth increments, such that the following relation is satisfied:

$$\mathbb{E}[R_t] = \mathbb{E}[\delta w_t] - \frac{\kappa}{2}\mathbb{V}[\delta w_t]$$

One such function is,

$$R_t := \delta w_t - \frac{\kappa}{2}(\delta w_t - \hat{\mu})^2 \tag{5}$$

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

Definition 2. In the examples that follow, we refer to out-of-sample annualized Sharpe ratio of profit/loss (P&L) after costs as the *performance metric*.

We could equivalently use sample estimates of (4) as the performance metric, but strategies maximizing (4) also maximize Sharpe ratio subject to constraints on volatility, and the Sharpe ratio is easier to interpret and to connect to other investment problems.

In what follows, learning methods will be compared using the performance metric. Each computation of the performance metric is done using a monte carlo simulation with 500,000 time steps–enough so that the estimation error of the performance metric itself is very low.

To create a testing environment for various learning methods, we make the (somewhat unrealistic) assumption 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 a hedged relative-value trade. There is an "equilibrium price" p_e such that $x_t = \log(p_t/p_e)$ has dynamics

$$dx_t = -\lambda x_t + \sigma\,\xi_t \tag{6}$$

where $\xi_t \sim N(0,1)$ and ξ_t, ξ_s are independent when $t \neq s$. This is a standard discretization of the Ornstein-Uhlenbeck process, and it means that p_t tends to revert to its long-run equilibrium level p_e with mean-reversion rate λ.

An action is simply to trade a certain number of shares; the action space is then a subset of the integers, with the sign of the integer denoting the trade direction. For illustration purposes, we take the action space to be limited to round lots of up to 200 shares in either direction:

$$\mathcal{A} = 100 \times \{n \in \mathbb{Z} : |n| \leq 2\}.$$

A real-world system would not have so restrictive a limit, but this contributes to ease of visualization; see the figures below.

The space of possible prices is:

$$\mathcal{P} = \text{TickSize} \cdot \{1, 2, \ldots, 1000\} \subset \mathbb{R}_+$$

We do not allow the agent, initially, to know anything about the dynamics. Hence, the agent does not know λ, σ, or even that some dynamics of the form (6) are valid.

The agent also does not know the trading cost. For a trade size of n shares we define

$$\text{cost}(n) = \text{multiplier} \times \text{TickSize} \times (|n| + 0.01n^2); \tag{7}$$

where in the examples below, we take multiplier $= 10$.

This is a rather punitive cost function: to trade $n = 100$ shares costs 2,000 ticks, or 200 dollars if TickSize $= 0.1$, or 2 dollars per share traded. Hence if we buy at $p_e - 4$ and sell at p_e, we net zero profit after costs. Hence a rough estimate of the no-trade zone is $[p_e - 4, p_e + 4]$, in agreement with Figure 2. In any case, with these cost assumptions, we expect that the Sharpe ratio of a model based on a pure-noise forecast will be strongly negative. More generally, we expect most simple/naive strategies to have negative Sharpe ratio net of costs, both here and in reality.

The state of the environment $s_t = (p_t, n_{t-1})$ will contain the security prices p_t, and the agent's position, in shares, coming into the period: n_{t-1}. The agent then chooses an action $a_t = \delta n_t \in \mathcal{A}$ which changes the position to $n_t = n_{t-1} + \delta n_t$ and observes a profit/loss equal to

$$\delta v_t = n_t(p_{t+1} - p_t) - \text{cost}(\delta n_t),$$

and reward $R_{t+1} = \delta v_{t+1} - 0.5\kappa\,(\delta v_{t+1})^2$. We take $p_e = 50.0$, TickSize $= 0.1$, $\lambda = \log(2)/5$ (5-day half-life), $\sigma = 0.15$, max holding $= 1{,}000$ shares, $\alpha = 0.1$, $\gamma = 0.99$.

The goal of this procedure is to discover the optimal policy, not the value function itself (and hence, the estimated value function $\hat{q}(s,a)$ is only a tool for estimating the policy π). Even so, we find it useful to visualize the value functions produced by our learning methods. For example, we may identify the no-trade zone is by plotting the action-value function as a function of price, for each of the available actions; we shall then see the prices for which the optimal action is zero.

More generally, denoting the state s by a pair consisting of prior holding and current price $s = (h, p)$, we may then consider the $h = 0$ slice of the action-value function as a collection of $|\mathcal{A}|$ functions $p \to \hat{q}((0,p),a)$, for each $a \in \mathcal{A}$. For each price level, the action which would be chosen by the greedy policy can be found by considering the pointwise maximum of the functions shown.

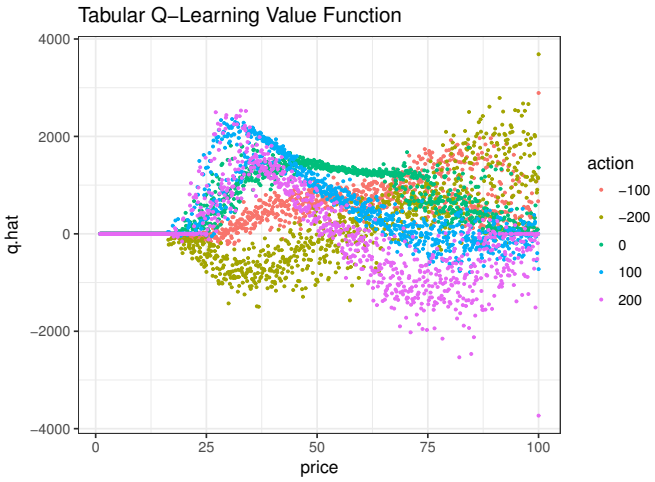

Figure 1: Value function $p \to \hat{q}((0,p),a)$, where $\hat{q}$ is estimated by the tabular method.

We run a sequence of 10 batches of 250,000 steps each, and also run a standard tabular Q-learning (TQL) setup for the same the total number of steps, 2.5 million. After these runs, each learner has seen precisely the same training data, so if the two methods were equally effec-

tive, we should expect the respective greedy policies to achieve similar performance.

The tabular method estimates each element $\hat{q}(s, a)$ individually, with no "nearest neighbor" effects or tendency towards continuity, as we see in Fig. 1.

In this example, the optimal action choice has a natural monotonicity, which we now describe intuitively. Suppose that our current holding is $h = 0$. Suppose that for some price $p < p_e$, the optimal action, given $h = 0$, is to buy 100 shares; it follows that for any price $p' < p$, the optimal action must be to buy at least 100 shares.

For large price values, the tabular value function seems to oscillate between several possible decisions, contradicting the monotonicity property. This is simply an aspect of estimation error and the fact that the tabular method hasn't fully converged even after 2.5 million iterations. The tabular value function also collapses to a trivial function in the left tail region, presumably due to those states not being visited very often – a property of the Ornstein-Uhlenbeck return process – whereas the model-tree method generalizes well to states not previously visited.

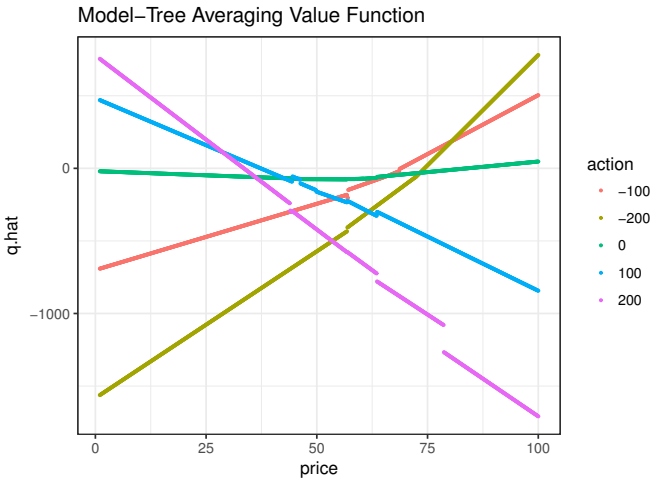

Figure 2: Value function $p \to \hat{q}((0, p), a)$ for various actions a, where $\hat{q}$ is estimated by MTA, and each model in cL is formed by the M5 model-tree method of Quinlan (1992).

Referring to Fig. 2, the model-averaging value function is easier to interpret than the tabular 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. There is a no-trade region in the center, where the green line is the maximum. There are then small regions on either side of the no-trade zone where a trade $n = \pm 100$ is optimal, while the maximum trade of ± 200 is being chosen for all points sufficiently far from equilibrium. The optimal action-choice displays the monotonicity property discussed above, and the optimal value function (the maximum of the functions in Fig. 2) is piecewise-continuous.

Our testing indicates that the model-averaging method not only produces a value-function estimate that is piecewise-continuous, but also outperforms the tabular method in the key performance metric, Sharpe ratio. Running each policy out of sample for 500,000 steps, we estimate Sharpe ratio of 2.78 for TQL, and 3.03 for MTA. The latter is better able to generalize to conditions unlike those it has already seen, as evidenced by the left-tails in Figs. 1–2. For smaller training sets, the difference is even more dramatic, as we discuss below.

Efficient Sample Use

In the previous example, we took advantage of the simulation-based approach and the speed of the training procedure to train the model on millions of time-steps. In the analysis of real financial time series, it is unlikely we will ever have so much data, so it is naturally of interest to understand the properties of these learning procedures in data-scarce situations.

This is related to the notion in statistics of *sample efficiency*, by which we mean the typical improvement of the performance metric per training sample. In this context, one reinforcement-learner is said to display greater sample efficiency than another, if it needs fewer training samples to achieve a given level of performance. We will show that the model-tree averaging (SLA) method introduced in this paper displays more efficient sample use than a tabular Q-learner (TQL).

For this exercise, we consider a single "experiment" to be $n_{batch} = 6$ batches, each batch of size 5,000, for a total of 30,000 samples. We train a tabular Q-learner (TQL) on the full set of 30,000 samples, and simultaneously update an SLA; the latter adds a new model to the list $\mathcal{L}$ after each batch of 5,000.

We consider two SLA methods, where the supervised-learner F_k for the prediction problem $Y = F_k(X)$ takes one of two possible forms:

1. The M5' model tree method of Quinlan (1992) with improvements by Wang and Witten (1997).

2. Bootstrap aggregating, where each of the base learners (ie. learners trained on bootstrap replicates of the training set) is an M5' model tree.

In the second variant, we further improve the M5' models via **b**ootstrap **agg**regation (Breiman, 1996), which was given the acronym "bagging" by Breiman. The

latter builds an ensemble of learners by making bootstrap replicates of the training set, and using each replicate to train a new model; the actual prediction is then the ensemble average. Breiman (1996) points out that bagging is especially helpful when the underlying learning method is unstable, or potentially sensitive to small changes in the data, which is the case for most tree models.

For each learning method, we are interested in the sampling distribution (over training sets of the given size) of the performance metric. We estimate this distribution by collecting values of the performance metric from 500 experiments, and plotting a nonparametric kernel density estimator.

Observe from Figure 3 that Q-learning with this sample size completely fails to overcome trading cost in all 500 experiments – all sharpe ratios are negative.

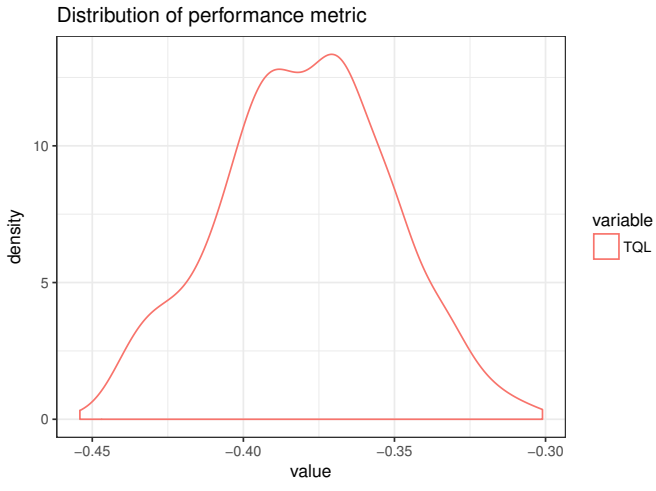

Figure 3: Distribution of the performance metric over 500 experiments, each with 30,000 samples, using TQL. This method completely fails to overcome trading cost in all 500 experiments that we ran – all sharpe ratios are negative.

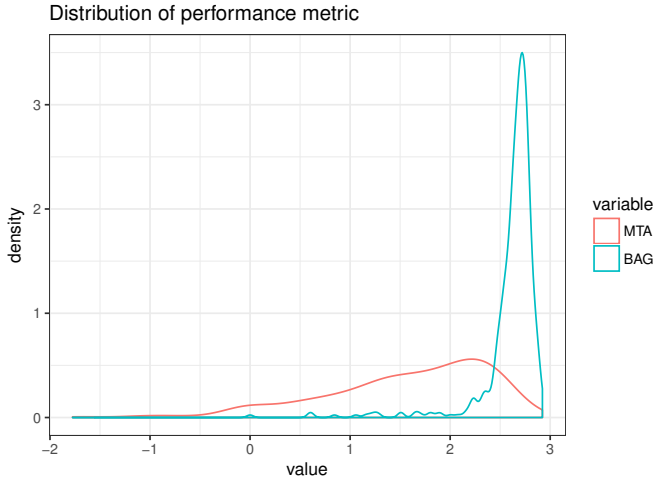

Figure 4: Distribution of the performance metric over 500 experiments, each with 30,000 samples, using SLA where the base learner is either an unbagged M5 model tree (MTA) or bagging an ensemble of M5 learners (BAG).

The MTA method generally does overcome t-costs, even with a scarcity of data, as Figure 4 shows, but there is relatively high variance in the performance metric. The best method of all is SLA where each model $Y = F(X)$ in step 2 of the algorithm is a bagged ensemble of M5 model trees. With SLA using bagged M5s, the Sharpe ratio is rarely below 2.0 for these experiments.

Conclusions

Motivated by trading applications, we have introduced a form of reinforcement learning, SLA, in which the internal representation of the action-value function is a model-averaging procedure:

$$\hat{q}(s,a) := \frac{1}{K}\sum_{k=1}^{K} F_k(s,a), \qquad \text{where } K = |\mathcal{L}|$$

where $\mathcal{L}$ is a list of models. The individual models in the list are built from batches, where each batch is run using the ϵ-greedy policy based on $\hat{q}(s,a)$ with the previously-learned models. Each batch generates a data set in which the output target

$$Y_t = R_{t+1} + \gamma\, \hat{q}(S_{t+1}, A_{t+1}) \tag{8}$$

is associated with the state-action pair $X_t := (S_t, A_t)$ that generated it. The next model is trained on this data set and added to $\mathcal{L}$.

The SLA family of reinforcement-learning methods solves two significant problems at once: it can make ex-

tremely efficient use of small samples, and it can operate on high-dimensional state space containing both continuous and discrete state variables and predictors. It essentially inherits both of these properties from the supervised-learners used to estimate $Y = F(X)$ in step two of the algorithm. Ensembles of M5 model trees work very well as the supervised-learners. Like deep neural networks, they are universal function approximators, but for the types of problems we consider, they converge more quickly and require no specialized hardware. The SLA technique thus overcomes the curse of dimensionality and is generalizable to high-dimensional problems, while simultaneously outperforming tabular Q-learning on the baseline problem (trading an illiquid mean-reverting asset).

This research opens up a path to handle arbitrary numbers of continuous and discrete predictors in the reinforcement-learning approach to trading. This should dramatically expand the range of optimal-trading problems that can be fruitfully approached using reinforcement-learning techniques. Another possible application is to automatic hedging: given a position in a derivative contract, can a machine learn to hedge the position?

References

Arrow, Kenneth J (1963). *Liquidity preference, Lecture VI in "Lecture Notes for Economics 285, The Economics of Uncertainty", pp 33-53*. In:

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

Breiman, Leo (1996). *Bagging predictors*. In: Machine learning 24.2, pp. 123–140.

Pratt, John W (1964). *Risk aversion in the small and in the large*. In: Econometrica: Journal of the Econometric Society, pp. 122–136.

Quinlan, J Ross (2014). *C4. 5: programs for machine learning*. Elsevier.

Quinlan, John R (1992). "Learning with continuous classes". In: *5th Australian joint conference on artificial intelligence*. Vol. 92. Singapore, pp. 343–348.

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

Robbins, Herbert and David Siegmund (1985). "A convergence theorem for non negative almost supermartingales and some applications". In: *Herbert Robbins Selected Papers*. Springer, pp. 111–135.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. Second edition, in progress. MIT press Cambridge. URL: http://incompleteideas.net/book/bookdraft2018jan1.pdf.

Wang, Yong and Ian H Witten (1997). "Inducing model trees for continuous classes". In: *Proceedings of the Ninth European Conference on Machine Learning*, pp. 128–137.