Deep Reinforcement Learning with Function Properties in Mean Reversion Strategies

Sophia Gu

Sophia Gu
was a graduate student in the Department of Mathematics in the Courant Institute of Mathematical Sciences at New York University in New York, NY.
keying@nyu.edu

KEY FINDINGS

- Instead of building an RL agent from scratch, the author employs ready-to-use technology using PPO from its originator and tests it out on mean reversion strategies.
- The author introduces a general framework of incorporating human insight, in particular economically motivated function properties, when training DRL agents.
- By combining the preceding two methodologies, the author demonstrates a highly performant and convergent DRL solution to common decision-making financial problems.

ABSTRACT

Over the past decades, researchers have been pushing the limits of deep reinforcement learning (DRL). Although DRL has attracted substantial interest from practitioners, many are blocked by having to search through a plethora of available methodologies that are seemingly alike, whereas others are still building RL agents from scratch based on classical theories. To address the aforementioned gaps in adopting the latest DRL methods, the author is particularly interested in testing out whether any of the recent technology developed by the leads in the field can be readily applied to a class of optimal trading problems. Unsurprisingly, many prominent breakthroughs in DRL are investigated and tested on strategic games—from AlphaGo to AlphaStar and, at about the same time, OpenAI Five. Thus, in this writing, the author shows precisely how to use a DRL library that is initially built for games in a commonly used trading strategy—mean reversion. And by introducing a framework that incorporates economically motivated function properties, they also demonstrate, through the library, a highly performant and convergent DRL solution to decision-making financial problems in general.

Mean reversion strategy has been studied for decades. Lakonishok, Shleifer, and Vishny (1994) first looked at mean reversion in earnings, and by contrasting past and future growth rates, they found that earnings tend to regress to their historical mean over time. Following that, many mean reversion strategies have emerged. Avellaneda and Lee (2008) built a mean reversion model using trading signals generated from Principal Component Analysis (PCA) and sector exchange-traded funds and related the performance of mean reversion statistical arbitrage with the stock market cycle. Other authors, like Bertram (2009) and Lipton and Prado (2020), have attempted to derive analytical solutions for optimal trades when the underlying price follows an Ornstein–Uhlenbeck (OU) process. Although those are groundbreaking findings, they only admit explicit solutions in the specific cases being studied and are

often vulnerable to numerical errors in practice. Recent mean reversion strategies started to make use of growing computing power as well as reinforcement learning (RL) algorithms, a younger sibling to optimal control introduced by Bellman (1957). For example, Ritter (2017) used a classical RL algorithm—tabular Q-learning—for simple problems approximated with an OU-driven price process and achieved an average Sharpe ratio close to 2.07.

One advantage of deep RL (DRL) compared to tabular Q-learning is that it permits continuous state and action spaces, which is often the limiting factor that prevents a trained model from achieving its theoretical optimal performance. That said, the landscape of a DRL policy function is often so complex that, in most situations, an agent searching for a good local optimum is like navigating a path covered in fog; It could easily get stuck in a small puddle before reaching anywhere close to its goal. However, in finance, investors often possess insight of optimal policies from economic principles. As an example, let us take a closer look at a mean-reverting price processes.

Due to short-term pricing inefficiencies, a price process in the stock market can exhibit a mean reversion property, which implies a near arbitrage in the system; when the price is too far out of its equilibrium, a trade betting that it returns back to the equilibrium has a slim chance of loss. In an RL context, it is equivalent to say that we expect the action suggested by the policy network to be monotonically decreasing with respect to price. So although one does not know the exact optimal solution to a trading problem, it is often not that difficult to come up with functional shapes that govern it. Surprisingly, this information is often absent when training RL agents but turns out to be critical because it effectively scopes the target function space. Thus, to some extent, solves the complex landscape problem that I mentioned at the beginning.

This article is organized as follows: "Preliminaries" introduces utility theory as well as the very algorithm that my chosen software is based upon on; "DRL Setup" and "Function Properties" are the main course where I lay out the theoretical framework for DRL with function properties, which is then followed by introducing the software and the neural net architecture in "Experimental Setup"; "Results" enumerates the findings on two representative mean-reverting price processes—an OU process and a more general Autoregressive Moving Average (ARMA) process.

The data supporting the results of this study, as well as the code implementing the trading environment and the DRL agents, are available in the GitHub repository: github.com/sophiagu/RLF.

PRELIMINARIES

Utility Theory

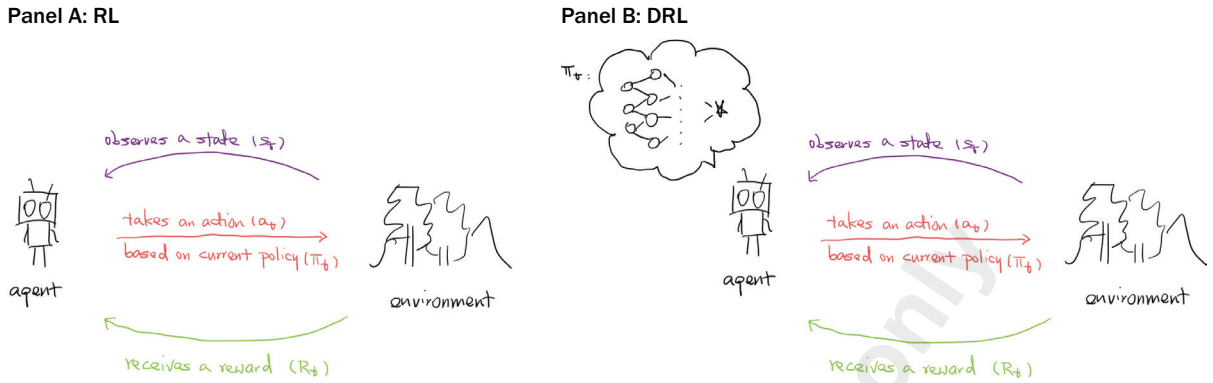
The reward function adopted in this article is based on the utility theory formalized by Pratt (1964) and Arrow (1971). Under their framework, suppose a rational investor invests in a stock over a finite time horizon: 1, 2, ..., T. She then chooses actions to maximize the expected utility of her terminal wealth

$$\mathbb{E}[u(w_T)] = \mathbb{E}\left[u\left(w_0 + \sum_{t=1}^{T} \delta w_t\right)\right]$$

where w_0 is the initial wealth and $\delta w_t = w_t - w_{t-1}$ is the change in wealth at each time step. The function $u : \mathbb{R} \to \mathbb{R}$ denotes the utility, which is a mapping from wealth to a real number. In order for the utility to make sense in the real world, u should be increasing. In addition, assume the investor is risk averse, then u is also concave. Refer to Pratt (1964) and Ingersoll (1987) for more details on the shape of u. It turns

EXHIBIT 1

One Iteration of RL and DRL Procedure



out that we can simplify this expectation by assuming the return of the stock follows a mean–variance distribution, defined in the following.

Definition 1 (mean–variance distribution). A random asset return r is said to follow a mean–variance equivalent distribution if it has a density $\rho(r)$, has finite first and second moments, and for any increasing utility function u and any initial wealth level w_0, there exists a constant $\kappa > 0$ such that the policy, which maximizes $\mathbb{E}[u(w_T)]$, is also optimal for the simpler problem:[1]

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

By writing $w_T = w_0 + \Sigma_{t=1}^{T}\delta w_t$, the expected utility can be further reduced to multi-period optimization problem

$$\max_{\pi}\sum_{t}\left\{\mathbb{E}[\delta w_t] - \frac{\kappa}{2}\mathbb{V}[\delta w_t]\right\}$$

DRL

The basic RL problem consists of an environment and an agent. At each iteration, the agent observes the current state of the environment and proposes an action based on a policy. After each interaction, the agent receives a reward from the environment and the environment updates its state. The change in the environment state can be autonomous (e.g., the stochastic evolution of stock price) or can be influenced by the agent's action (e.g., the change in the bid–offer spread after the agent exercises a trade). DRL differs from RL in that it trains a neural network to learn that policy.[2] Exhibit. 1 compares RL and DRL in a pictorial way.

Proximal Policy Optimization

When comes to picking a specific RL algorithm, I focused on model-free algorithms because model-free algorithms can be easily generalized to other problems with little tweak, also they have been studied more extensively than model-based algorithms.

[1] The return does not have to be normal. Any elliptical distribution is mean–variance equivalent; see Chamberlain (1983).

[2] In this writing, I call such a neural net a policy network and will use those two terms interchangeably.

Within the scope of model-free RL algorithms, the two big branches are Q-learning and policy optimization. Although I have experimented with both approaches, policy optimization yielded more promising results given the same amount of training time. For policy optimization, besides its built-in support for continuous state space and action space, it directly improves the policy as it updates the policy network by following the gradients w.r.t. the policy itself, whereas in Q-learning the updates are done based on the estimates of the value function, which only implicitly improves the policy. It turns out that Q-learning also tends to be less stable than policy optimization algorithms. For references, see Tsitsiklis and Roy (1997), Szepesvari (2009), and Chapter 11 of Sutton and Barto (2018).

As a result, I settled down to a policy-gradient-based DRL algorithm called proximal policy optimization (PPO). PPO is one of the actor-critic algorithms that have two neural networks, one for estimating the policy (actor) function and the other for the value (critic) function. The main policy gradient loss function is

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

Here, θ represents the parameters or weights of a neural net, $\pi_\theta(a_t|s_t)$ is the probability of choosing an action a_t based on a state s_t, and $\hat{A}_t$ is an estimate of the advantage function, that is, the relative value of the selected action to the base action. Specifically, $\hat{A}_t$ is positive when the selected action performs better than the base action and is negative otherwise.

To fully understand this equation and its subsequent variations, I refer to the well-written original paper on PPO, Schulman et al. (2017). Intuitively, this loss function tells an agent to put more weight on a good policy, in other words, to assign higher probabilities to actions that lead to higher critic values and vice versa.

Another thing to keep in mind is that PPO combines ideas from Advantage Actor Critic (A2C) (having multiple workers) and Trusted Region Policy Optimization (TRPO) (which uses a trust region to improve the actor). The gist is, in order to avoid overfitting, the new policy should be not too far from the old one after an update. For that, PPO uses clipping to avoid too large updates. Nevertheless, it is helpful to look at the training loop to understand how the agent learns; shown in Algorithm 1.

DRL SETUP

In this section, I break down the RL setup into three main components—a state space, an action space, and a reward function. I will not labor over the bulk of derivations here because they are already well-known in the field. For a rigorous derivation, see Ritter (2017).

Algorithm 1 PPO, Actor-Critic Style

for iteration := 1, 2, ... **do**
  for actor := 1, 2, ..., N **do**
    Run policy $\pi_{\theta old}$ in environment for T time steps
    Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
  end for
  Optimize surrogate L w.r.t. θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} := \theta$
end for

State Space

The term state, in RL, refers to the state of the environment; It is a data structure consisting of all the information that an agent needs for choosing an action. I use s to denote the state of the environment and index it by time t, meaning it is the information available at that particular time step. For mean reversion in the stock market, clearly we will need the current holding of the stock, h_t, and its current price, p_t. Besides, I also include the stock price from an earlier time step, p_{t-1}. Recall that underlying every RL problem is a Markov decision process, which means that an agent should be able to make a decision at time step t solely based on the information present in s_t rather than any information in the prior states, $s_{\leq t}$. Including the earlier price, which tells the agent not only the current price but also the change in price, allows the agent to learn the underlying price dynamics at the same time.

Action Space

Because we are building an agent to maximize a reward function, the action space is just the trade that the agent makes. For mean reversion, this is equal to $h_t - h_{t-1}$ at time step t.

Reward Function

The goal of RL is reward maximization, in which the reward is typically picked by a human. Although defining a reward can sometimes be a hard task in RL, for our problem, it is simply the expected utility of the final wealth that the agent achieves after a set time period of trading activities. Following the discussion in utility theory, we can approximate the reward at each time step t as a mean–variance equation

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

where, in our problem, δw_t is the Profit & Loss (P&L) at time step t and can be written precisely as $(p_t - p_{t-1})h_{t-1} - cost_t$. Notice we also account for transaction costs accompanying the trade.

The particular transaction cost model I use is rooted in Garleanu and Pedersen (2013). To compute trading cost, imagine we are looking at a limit order book: For fairly liquid securities, the bid–offer spread is either one tick or two ticks. In the United States, a tick is a penny. Typically, the displayed liquidity on the bid and offer is a few hundred shares. Let us suppose it is 100 shares. To buy shares, we cross the spread and take out the offer. In other words, we transact one lottery 100 shares at the offer, and we cross the spread to do it. If the spread is two ticks wide, then that is the cost of one tick relative to the midpoint. Extending the idea linearly, if we trade 10 lots, then we move the price 10 ticks.[3]

FUNCTION PROPERTIES

Inspired by Sill and Abu-Mostafa (1997), which introduced a monotonic hint for classification problems, in this article, I will instead focus on a DRL setting and build a framework of incorporating more general function properties into its training process.

[3] For larger trades, that may not be a good approximation, but for small trades that is not too far off reality.

I also give a few examples illustrating the application of the framework in problems like option pricing and statistical arbitrage—in particular, mean reversion strategies.

Motivation and Definitions

When searching for an optimal trading strategy using a neural network, we are effectively looking for an optimal function in a space parametrized by

$$\mathbb{P} = \{\textit{All distributions on } \mathbb{R} \textit{ that can be expressed by a neural network}\}$$

Notice that this space is huge. But stochastic gradient descent (SGD) is inherently only working in a low-dimensional subspace and cannot explore the whole space of the parameters. The subspace over which SGD operates is spanned by all the stochastic gradients along the trajectory, and these stochastic gradients are highly correlated with each other. Thus, the dimensionality of such a subspace is upperbounded by the number of SGD updates, which is typically a small number compared to the space itself. One way to reduce the function space is to encourage the underlying algorithm to search in a desirable subset of it. This leads me to introducing a *function penalty*. Before we embark on that, let us define some terms that will become useful later.

Definition 2 (policy function). Let $\mathbb{I}$ be an input space and $\mathbb{A}$ be the corresponding action space. $a : \mathbb{I} \to \mathbb{A}$ is a policy function that takes an input $i \in \mathbb{I}$ and produces either a deterministic or stochastic action $a \in \mathbb{A}$. For the set of problems we are interested in, a is parametrized by a set of parameters $\theta \in \Theta$, where Θ is specified by a given problem.

Definition 3 (function penalty). Let B be a subset in $\mathbb{I} \times \mathbb{A}$. A characteristic function $I^B : \mathbb{I} \times \mathbb{A} \to \{0, 1\}$ is called a function penalty if it takes an input and policy pair (i, a), outputs zero when $(i, a) \in B$ and a unit penalty otherwise.

One can think of B as an acceptable region and a function penalty as an indicator of whether an action suggested by a policy network is within that region. For example, let $x \in [-1, 1]$ be the x-coordinate of a vector in $\mathbb{R}^2$ and $y = a(x) \in [-1, 1]$ be the y-coordinate of the vector. If, say, one expects the vectors $(x, y)^T$ to lie within a unit circle centered at the origin, then she can define

$$B = \{(x, y) \in [-1,1] \times [-1,1] \mid x^2 + y^2 \leq 1\}$$

which results in

$$I^B(x; y) = \mathbb{1}_{\{\neg B\}} = \mathbb{1}_{\{x^2+y^2>1\}}$$

As you may start to see, if we sample more and more xs and get ys produced by $a(x)$, then according to the law of large numbers, we will be more and more confident about our understanding of the policy's performance by summing up the values of $I^B(x; y)$. That gives the idea of the next definition.

Definition 4 (cumulative function penalty). Given a set of inputs $I = \{i\}_{j \in \mathbb{N}}$, a group of function penalties $L = \{l\}_{k \in \mathbb{N}}$ and a policy a, define the cumulative function penalty f as

$$f(I, L, a) = \lambda \sum_{i_j \in I} \vee_{l_k \in L} l_k(i_j; a(i_j))$$

where λ is a scalar and $\vee$ denotes a logical OR operation.

It is important to pick λ so that $f(I, L, a)$ is on a similar scale to the remaining part of the reward function. Before pressing on to the next topic, I want to emphasize a

few places in finance where economically meaningful constraints can be incorporated into training using this framework.

Examples in Finance

Option pricing. Consider the model-free constraints on the shape of the option pricing function. For instance, the price of a European call should satisfy, assuming a compounded interest rate r,

$$c(S,\tau,K) \geq (S - Ke^{-r\tau})_+$$

where S is current stock price, K is strike price, and τ is time to maturity.

To see whether this inequality holds, consider no-arbitrage arguments and observe that holding a long call is no worse than holding its corresponding forward contract, and thus, $c(S, \tau, K) \geq S - Ke^{-r\tau}$; neither is holding a long call worse than holding nothing, which is equivalent to saying $c(S, \tau, K) \geq 0$.

A function penalty corresponding to the aforementioned constraint is simply

$$l_1(S,\tau,K;c) = \mathbb{1}_{\{c<0\}} \vee \mathbb{1}_{\{c<S-Ke^{-r\tau}\}}$$

which emits a unit penalty when either of the constraints is not satisfied. Using a similar argument, one can also show that

$$c(S,\tau,K) \leq Se^{-d\tau}$$

where d is a compounded dividend yield,[4] yielding

$$l_2(S,\tau,K;c) = \mathbb{1}_{\{c>Se^{-d\tau}\}}$$

It is worth pointing out that the inputs to a function penalty need not be unary. So instead of passing one stock price into l, one may choose to pass several prices. For instance, observe that the payoff $c(S, \tau, K)$ is a convex function in S because $\frac{\partial^2 c}{\partial S^2} \geq 0$. One way to construct a function penalty for constraining c to the set of convex functions in S is to directly apply its definition: A function $f(x)$ is convex *iff* $f(\kappa x + (1 - \kappa)y) \leq \kappa f(x) + (1 - \kappa)f(y)$ *for* $\kappa \in (0, 1)$. This gives, for two different stock prices S_1, S_2,

$$l(S_1, S_2; c) = \mathbb{1}_{\{c(\kappa S_1 + (1-\kappa)S_2) > \kappa c(S_1) + (1-\kappa)c(S_2)\}}$$

Statistical arbitrage. Now let us look at some examples exploiting arbitrage opportunities. Recall that, when a stock price has a mean reversion nature, optimal trades should be a decreasing function in price. We can, following a similar fashion, construct a cumulative function penalty for mean reversion strategies. Because we will code up this particular function penalty for the study, it deserves a special definition.

Definition 5 (cumulative function penalty for mean reversion). Given a set of stock prices generated for one epoch of training, $p_1, \ldots, p_T \in \mathbb{R}^+$, and consider a policy network that maps a stock price to a trade, a. Let i, j be two integers sampled uniformly from 1 to T such that $i \neq j$. To simplify the notation, write $a_i = a(p_i)$, $a_j = a(p_j)$, and denote $I = \{p_i, p_j\}_{i,j\in[1,T]}$.

[4] One derivation is available in Birke and Pilz (2009).

We use the following function penalty

$$l_{mr}(p_i, p_j; a) = \mathbb{1}_{\{(p_i < p_j) \oplus (a_i > a_j)\}}$$

and its corresponding cumulative function penalty for mean reversion

$$f_{mr}(I, l_{mr}, a) = \lambda \sum_{p_i, p_j \in I} l_{mr}(p_i, p_j; a)$$

where $\oplus$ represents a logical XOR operation.

I conclude this section with a closely related mean reversion strategy—pairs trading. Suppose a pair of stock prices $(p_t, q_t)^T$ has a cointegration vector $(1, -\alpha)^T$, where $\alpha \in \mathbb{R}$, that is, a portfolio formed by $\pi_t = p_t - \alpha q_t$ is stationary and hence mean reverting. A pairs-trading strategy takes a long position on the portfolio when its value drops and reverts its position when the value goes up. So the trade for π_t behaves exactly like the simple mean reversion strategy described earlier. As a result, in Definition 5, one can simply substitute p_i by π_i to get a free cumulative function penalty for pairs trading.

New Reward Function with Function Properties

To incorporate the function properties into the training process, we follow the Bayesian framework. In a nutshell, function properties or penalties tell us how likely or unlikely a candidate policy function, a, is given a specific problem setting, D. In the world of Bayesian, this is the a priori probability density of a candidate function. Here we denote it as $\mathbb{P}(a|D)$.

Let R be the original reward, then we can use Bayes's theorem to derive the posterior likelihood of a policy function given both the problem setting and the original reward

$$\mathbb{P}(a|D, R) \propto \mathbb{P}(R|D, a)\mathbb{P}(a|D)$$

Taking log on both sides and plugging in the cumulative function penalty derived in the prior section to obtain

$$\log\mathbb{P}(a|D, R) \propto \log\mathbb{P}(R|D, a) + \log\mathbb{P}(a|D) \propto R - f_{mr}$$

Putting everything together, the new reward function that will be used for the rest of the study is

$$R_t^* \approx \delta w_t - \frac{\kappa}{2}(\delta w_t)^2 - f_{mr}$$

EXPERIMENTAL SETUP

Software

I use OpenAI's improved version of its original implementation of PPO—Stable Baselines—running on TensorFlow. This release of Stable Baselines includes scalable, parallel implementations of PPO that use Message Passing Interface (MPI) for data passing.

Network Architecture

For both value and policy networks, I use a 64 × 64 feedforward neural net with Rectified Linear Unit (ReLU) activation function followed by an Long Short-Term Memory (LSTM) layer with 256 cells. This network is designed to be slightly bigger than the actual size of the problem for smoother optimization. Both training and hyperparameters tuning use an Adam optimizer with a learning rate of $1e-5$ and early stopping. I counter the problem of potential overfitting by adding $l1$ and $l2$ regularizations with rates of 0.01 and 0.05, respectively. These regularizations encourage smaller and more sparse learned weights.

Further, each model uses five random restarts and autoselects the parameters that give the best performance for out-of-sample testing. Doing several runs is highly encouraged for RL; as stated in Stable Baselines' documentation[5] that:

> The data used to train the [RL] agent is collected through interactions with the environment by the agent itself (compared to supervised learning where you have a fixed dataset for instance). This dependence can lead to vicious circle: if the agent collects poor quality data (e.g., trajectories with no rewards), then it will not improve and continue to amass bad trajectories. This factor, among others, explains that results in RL may vary from one run to another (i.e., when only the seed of the pseudo-random generator changes).

Simulation

I use simulated stochastic price processes—an OU process and an ARMA(2,1) process—for training and testing DRL models.

Suppose that there is some equilibrium price, p_e; a variance σ^2; and a positive mean reversion rate λ. Let $x_t = \log(p_t/p_e)$, then the OU process follows

$$dx_t = -\lambda x_t dt + \sigma dW_t$$

where W_t is a Wiener process.

Similarly, the ARMA(2,1) process is defined to have the dynamics

$$dx_t = -(\lambda_1 x_t + \lambda_2 x_{t-1})dt + \sigma_1 dW_t + \sigma_2 dW_{t-1}$$

Both processes are stationary[6] and therefore mean reverting.

For simplicity, I pick $dt = 1$ in the Monte Carlo simulations. It follows that $dW_t \approx \sqrt{dt}\,\epsilon_t = \epsilon_t$, where $\epsilon_t \sim N(0, 1)$ is iid white noise.[7]

Success Criteria

To compare performance across agents, the standard is to use an annualized Sharpe ratio along with its standard deviation, time steps to convergence, and so on.

[5] https://stable-baselines.readthedocs.io/en/master/guide/rl_tips.html.

[6] This can be verified using the unit root test.

[7] The trajectory of x_t can be sampled exactly. For example, in the OU process, one can instead draw samples from $\mathcal{N}\left(0, \frac{\sigma^2}{2\lambda}(1-e^{-2\lambda t})\right)$ with arbitrary time step t.

EXHIBIT 2

Sharpe Ratios of the Trades from 10,000 Out-of-Sample Simulations of OU Process

Model	Q-Learning (*benchmark*)	Model A	Model B (*with fn penalty*)
Mean	2.07	2.10	2.78
Std	NA	0.375	0.329
Time steps to convergence	1,000,000	7,000	4,000

EXHIBIT 3

Sharpe Ratios of the Trades from 10,000 Out-of-Sample Simulations of ARMA Process

Model	Model A	Model B (*with fn penalty*)
Mean	2.46	3.22
Std	0.479	0.268
Time steps to convergence	4,000	10,000

RESULTS

I ran 10,000 Monte Carlo simulations for evaluating different models' out-of-sample performance using Sharpe ratio for both an OU process and an ARMA process. Exhibits 2 and 3 display their Sharpe ratios, respectively. To make the notations easier to follow, I denote Model A for agents trained using the original mean–variance reward function and Model B for agents trained using the new augmented reward function. For the OU process, I also have a chance to compare my agents to the tabular Q-learning model from Ritter (2017).

Exhibits 4 and 5 show the kernel density estimates of the Sharpe ratios of all simulation paths. The idea of kernel density estimates is to plot the observed samples on a line and to smooth them so that they look like a density.

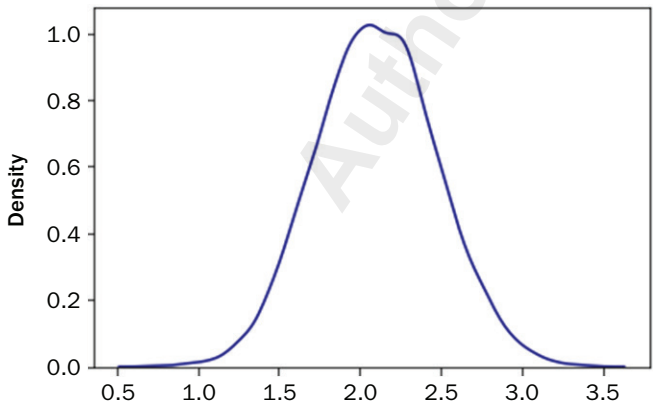
For both processes, we observe a noticeable increase in the average Sharpe ratio and a decrease in variance when incorporating a function property in training. Moreover, for each process, I performed a two-sample t-test to determine whether we have high confidence in the improvement in performance. The differences in Sharpe ratios are indeed highly statistically significant, with t-statistics of 135 and 139, respectively.

Another interesting observation is that, with DRL, the agents were able to converge within 10,000 time steps, and the fastest only took 4,000 steps, much faster than the tabular Q-learning agent, which took about one million training steps.

EXHIBIT 4

Kernel Density Estimates of the Sharpe Ratios from 10,000 Out-of-Sample Simulations of OU Process

Panel A: Model A



Panel B: Model B

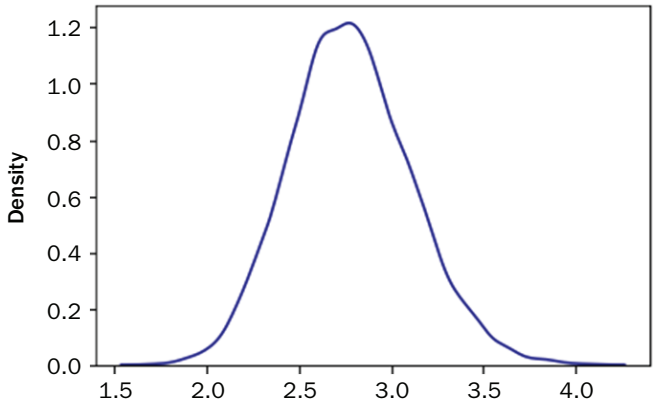
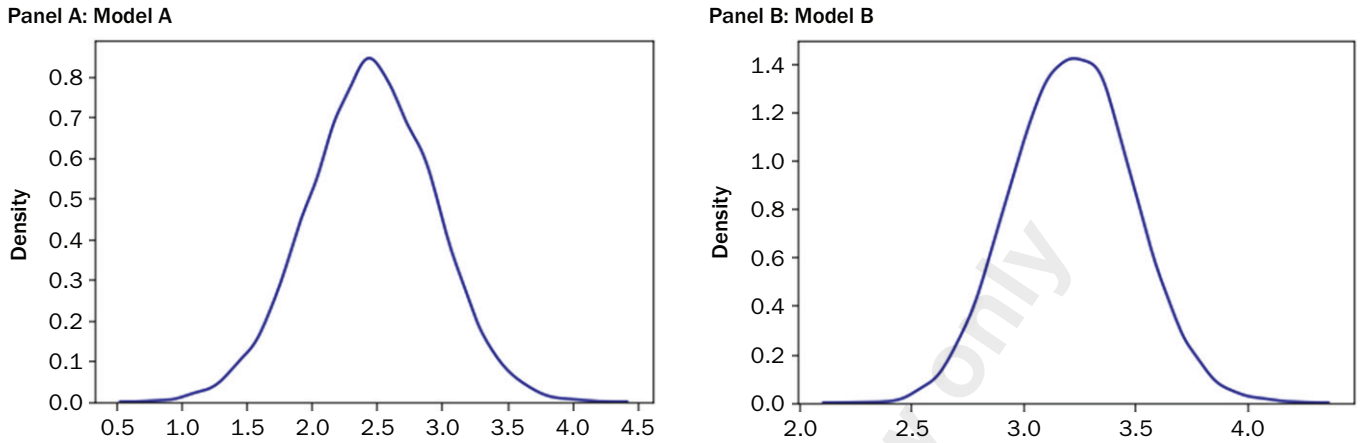


EXHIBIT 5

Kernel Density Estimates of the Sharpe Ratios from 10,000 Out-of-Sample Simulations of ARMA Process



CONCLUSION

This article demonstrates how to apply DRL to a typical decision-making financial problem using the latest technology developed by the pioneers in the field and how to use domain knowledge to encourage the underlying training algorithm for finding better local optima. I provide a proof of concept in a controlled numerical simulation, which permits an approximate arbitrage, and I verify that the DRL agent finds and exploits this arbitrage in a highly efficient way, while producing Sharpe ratios that surpass similar prior works, with high statistical significance.

Before concluding, I leave open two avenues to look into further:

1) With simple function penalties, I did not notice any increase in training time, but if you are to incorporate more convoluted properties that have high demand on computation power at each epoch, it may indeed slow down the training process. One potential solution is to build the function property directly into the network architecture (examples for convex functions and monotonic increasing functions are included in the accompanying GitHub repository);
2) I only trained the agents in a purely simulated environment. It would be interesting to see how they perform in the real market. One foreseeable challenge is that we may not have as much real-world data as simulated data. A potential workaround will be to train the model on simulated data first, finding a good initialization of network weights (for example, using an encoder-decoder structure) before training on any real market data.

ACKNOWLEDGMENTS

I am grateful to the faculty at NYU Courant for enlightening me on the subject, as well as for sponsoring High Performance Computing (HPC) resources that made all the computation possible. In particular, I greatly appreciate Gordon Ritter, who has had several meaningful discussions with me. His prior research is also the one that gives me a lot of inspirations. In addition, I would like to thank Oriol Vinyals, one of the leads on the AlphaStar project from Deepmind, for clearing my doubts about DRL in the early stage of this project.

REFERENCES

Arrow, K. J. *Essays in the Theory of Risk-Bearing.* North-Holland Pub. Co., 1971. Amsterdam, Netherlands.

Avellaneda, M., and J. Lee. 2008. "Statistical Arbitrage in the U.S. Equities Market." *SSRN,* availabel at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1153505.

Bellman, R. *Dynamic Programming.* Princeton, NJ: Princeton University Press, 1957.

Bertram, W. K. 2009. "Analytic Solutions for Optimal Statistical Arbitrage Trading." *SSRN*, available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1505073.

Birke, M., and K. F. Pilz. 2009. "Nonparametric Option Pricing with No-Arbitrage Constraints." *The Journal of Finance* 7 (2): 53–76.

Chamberlain, G. 1983. "A Characterization of the Distributions That Imply Mean–Variance Utility Functions." *Journal of Economic Theory* 29 (1): 185–201.

Garleanu, N., and L. H. Pedersen. 2013. "Dynamic Trading with Predictable Returns and Transaction Costs." *The Journal of Finance* 68 (6): 2309–2340.

Ingersoll, J. E. *Theory of Financial Decision Making.* Vol. 3. New York, NY: Rowman & Little-Field, 1987.

Kolm, P. N., and G. Ritter. 2019. "Modern Perspectives on Reinforcement Learning in Finance." *SSRN* available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3449401.

Lakonishok, J., A. Shleifer, and R. W. Vishny. 1994. "Contrarian Investment, Extrapolation, and Risk." *The Journal of Finance* 49 (5): 1541–1578.

Lipton, A., and M. L. Prado. 2020. "A Closed-Form Solution for Optimal Mean-Reverting Trading Strategies." *SSRN* available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3534445.

Pratt, J. W. 1964. "Risk Aversion in the Small and in the Large." *Econometrica: Journal of the Econometric Society* 32 (1/2): 122–136.

Ritter, G. 2017. "Machine Learning for Trading." *SSRN* available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3015609.

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

Sill, J., and Y Abu-Mostafa. *Advances in Neural Information Processing Systems*, Vol. 9. Cambridge, MA: MIT Press, 1997.

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

Szepesvari, C. *Algorithms for Reinforcement Learning.* Morgan & Claypool Publishers, 2009.

Tsitsiklis, J. N., and B. V. Roy. 1997. "An Analysis of Temporal-Difference Learning with Function Approximation." *IEEE Transactions on Automatic Control* 42 (5): 674–690.