Available online at www.sciencedirect.com

ScienceDirect

The Journal of Finance and Data Science 6 (2020) 16–30

http://www.keaipublishing.com/en/journals/jfds/

Deep deterministic portfolio optimization

Ayman Chaouki [a,c], Stephen Hardiman [a], Christian Schmidt [a,b], Emmanuel Sérié [a,*], Joachim de Lataillade [a]

[a] *Capital Fund Management, 23 rue de l'Université, 75007, Paris, France*
[b] *Chair of Econophysics and Complex Systems, École Polytechnique, Palaiseau, France*
[c] *École Centrale-Suplec, Gif-Sur-Yvette, France*

Received 19 March 2020; accepted 26 June 2020
Available online 9 July 2020

Abstract

Can deep reinforcement learning algorithms be exploited as solvers for optimal trading strategies? The aim of this work is to test reinforcement learning algorithms on conceptually simple, but mathematically non-trivial, trading environments. The environments are chosen such that an optimal or close-to-optimal trading strategy is known. We study the deep deterministic policy gradient algorithm and show that such a reinforcement learning agent can successfully recover the essential features of the optimal trading strategies and achieve close-to-optimal rewards.

Keywords: Reinforcement learning; Stochastic control; Portfolio optimization

1. Introduction

The fusion of reinforcement learning (RL) with deep learning techniques, aka. deep reinforcement learning (dRL), has experienced an astonishing increase in popularity over the last years.[1] Undoubtedly, dRL has been successfully applied to a broad range of applications with major success ranging from playing games, controlling robots, trading and even solving complex physics problems.[2–10]

Finance has been among the many branches that devoted much attention to reformulate their problems in a dRL framework to find new algorithmic avenues to solve them.[11,12] One subject of interest has been the application of (d) RL to dynamical portfolio allocation.[11,6,13–15]

Besides the astonishing practical results, dRL has been criticized to possess a reproducibility issue.[1,16,17] And indeed, to the best of our knowledge, the dRL algorithms for portfolio allocation that can be found in the literature are typically only compared with respect to sub-optimal reference strategies (with some restricted exceptions, such as[15]).

* Corresponding author.
E-mail address: emmanuel.serie@cfm.fr (E. Sérié).
Peer review under responsibility of China Science Publishing & Media Ltd.

https://doi.org/10.1016/j.jfds.2020.06.002

Little effort has been devoted to the question of whether dRL can learn known optimal trading strategies for the dynamical portfolio allocation problem. In this note we address this question and explore the potentials and pitfalls of dRL applied to portfolio allocation. We do so by testing the widely used deep deterministic policy gradients (DDPG) algorithm against an ensemble of three different trading problems for which the *optimal (or close-to-optimal) control strategies* are known.

1.1. Dynamic portfolio optimization and reinforcement learning

Formally, the RL problem is a (stochastic) control problem of the following form:

$$\begin{aligned} &\max_{\{a_t\}} \quad \mathbb{E}\left[\sum_{t=0}^{T\,1} \mathrm{rwd}_t(s_t, a_t, s_{t+1}, \xi_t)\right] \\ &\text{s.t.} \quad s_{t+1} = f_t(s_t, a_t, \eta_t)\,, \end{aligned} \tag{1}$$

where $a_t \in A$ indicates the control, aka. actions, $s_t \in S$ the state of the system at time t, η_t and ξ_t are noise variables, and rwd_t is the reward received at every time step. In RL the second line in the above equation is usually referred to as the 'environment'. The 'agent' intends to choose its actions a_t, given the state s_t, so as to maximize the total expected accumulated reward.

Comparing this with the setting of dynamic portfolio optimization reveals the link between the two problems. For the sake of simplicity we will restrict the following discussion to the one dimensional case. In dynamic portfolio optimization the investor aims to dynamically allocate his weight $\pi_t \in \mathbb{R}$ on an asset, that yields returns r_t, such that the expected utility $U(\bullet)$ of future wealth is maximized:

$$\begin{aligned} &\max_{\{\pi_t\}} \quad \mathbb{E}\left[U\left(\sum_{t=0}^{T-1} \mathrm{PnL}_{t,t+1}\right)\right] \\ &\text{s.t.} \quad \text{constraints.} \end{aligned} \tag{2}$$

where $\mathrm{PnL}_{t,t+1}$ stands for 'profit and loss', i.e. $\mathrm{PnL}_{t,t+1} = \mathrm{gain}_{t+1} - \mathrm{cost}_{t,t+1}$, with $\mathrm{gain}_t = \pi_t r_t$ the profit that the portfolio yields at every time step and $\mathrm{cost}_{t,t+1}$ some cost function that incorporates trading cost and/or other fees and effective costs.

In general this latter formulation (2) encompasses a much bigger class of problems (and in particular the former (1)). In this work we are interested in problems that lay at the intersection of (1) and (2). In particular we shall consider problems for which the utility maximization can be brought into the following form:

$$\max_{\{\pi_t\}} \mathbb{E}\left[\sum_{t=0}^{T-1} \pi_{t+1} r_{t+1} - \mathrm{cost}(|\pi_{t+1} - \pi_t|) - \mathrm{risk}(\pi_{t+1})\right]. \tag{3}$$

The risk term originates from the shape of the utility function and can be thought of as a regularizer that penalizes risky portfolio weights. For example $\mathrm{risk}(\pi_t) = \pi_t^2$ punishes large positions for being more risky. One important class of problems that fall into this framework are the 'mean-variance equivalent' problems.[18,19]

In order to bring the above problem into the RL framework it is natural to define the actions as the trades between subsequent time steps $a_t = \pi_{t+1} - \pi_t$. It is less obvious what variables should play the role of the state s_t according to which the agent takes its action. A standard assumption is that the returns r_t can be decomposed into a predictable and an unpredictable noise term, p_t and $\eta_t^{(r)}$ respectively:

$$r_{t+1} = p_t + \eta_t^{(r)} \tag{4}$$

The dynamic portfolio optimization can now be recast as an RL problem:

$$\max_{\{a_t\}} \mathbb{E}\left[\sum_{t=0}^{T-1} \pi_{t+1} r_{t+1} - \text{cost}(|a_t|) - \text{risk}(\pi_{t+1})\right]$$
$$\text{s.t.} \begin{cases} \pi_{t+1} &= \pi_t + a_t \\ p_{t+1} &= f(p_t) + \eta_t^{(p)} \\ r_{t+1} &= p_t + \eta_t^{(r)} \end{cases}. \tag{5}$$

where we additionally assumed that p_t follows a Markovian dynamic.

Most control problems of the above form are not known to have closed form solutions and require some sort of algorithmic approach. The need for RL comes to bear, when problem (5) has difficult reward or state transition functions that result in non-linear control policies. The *deep* RL framework becomes particularly necessary when the control and/or action spaces are continuous and traditional methods, such as Q-learning, fail. It is most challenging in the model-free context where no further model assumptions are made (neither on the reward function, nor on the dynamical equations of the state) and the only source of information are the accumulated rewards from which the trading policy and/or value function must be deduced. This will be the setting considered in this manuscript.

1.2. Contributions

Applying RL to portfolio optimization is no new idea,[15,13] indeed it is almost as old as the model-free RL algorithms.[20,21] However, to the best of our knowledge, besides several publications on modern dRL strategies for trading, these attempts never seem to have been systematically evaluated against known optimal strategies.

Our contribution is to propose an ensemble of dynamic portfolio optimization problems for which optimal (or close-to-optimal) control policies are known and to test DDPG against these strategies. This reveals the potential and pitfalls of current dRL approaches to this problem. The set of problems that we propose are also of interest beyond dynamic portfolio optimization as they provide testing environments with conceptually simple, but mathematically challenging control policies. The environments and code is accessible through our repository.[1]

1.3. Related work

Modern portfolio optimization has a long history that goes back to Markowitz[18] and Merton.[22] In these initial formulations trading costs were typically neglected. If, however, cost is present the optimal strategy must (a) plan ahead to take into account possible auto-correlation in the predictor and (b) trade lightly to take into account the cost.[23–28]

There are special cases of problem (5) that can be solve in a closed form[26,23,27] and that we will take as reference solutions. Further details will be discussed in section 2. In general, however, closed form solutions of such stochastic control problems are scarce and it is necessary to resort to approximations.

Traditional methods are often (but not always) based on dynamic programming,[29,26,25] model-predictive control and convex optimization, cf. 30 and references therein. Model-free reinforcement learning is an alternative approach that does not assume a model of the system and takes decision solely from the information received at every time step through the rewards in (5). Early works that are applying this idea to dynamic portfolio allocation can be found in15,25,31,13. However, these approaches were mostly limited to low-dimensional, discrete state and action spaces. The rise of deep techniques has revived the interest in applying dRL strategies to more complicated and/or continuous settings[32] and more modern dRL approaches for dynamical portfolio allocation can e.g. be found in[6,14].

A somewhat related approach to what we are doing was followed by the authors of[33] that investigated if dRL strategies successfully learn known optimal algorithmic strategies for online optimization problems.

[1] https://github.com/CFMTech/Deep-RL-for-Portfolio-Optimization.

2. Setting the stage

2.1. The environments and their reference solutions

This section establishes the three environments that the algorithm is tested against. We start by introducing the state space, i.e. the dynamics $s_t \mapsto s_{t+1}$ and then define the three different reward functions (5), each of which defines a different environment with distinct optimal reference solutions, i.e. controls.

Throughout the manuscript we assume that the predictor p_t is an autoregressive (AR) process with parameter ρ, normalized to unity equilibrium variance. The variables π_t, p_t, r_t thus evolve according to

$$\pi_{t+1} = \pi_t + a_t \tag{6}$$

$$p_{t+1} = \rho p_t + \eta_t^{(p)} \tag{7}$$

$$r_{t+1} = p_t + \eta_t^{(r)} \,. \tag{8}$$

Note that the agent's state $s_t = (\pi_t, p_t)$ only contains the observables available at time t.

It is important to note that (5) together with (6)−(8) in principle permits to eliminate the returns $\{r_t\}$ entirely; simply by replacing $r_{t+1} \mapsto p_t$ in the rewards. In this case, the agent has perfect state observations in that the reward received at time t is entirely composed of observables accessible to the agent: π_t, p_t and a_t. In practice, however, this is an unrealistic assumption and one should rather work with the rewards that contain the additional noise from the returns.

In the first case one simply asks if RL can find back (almost) optimal stochastic control strategies *under perfect state information*. In the second case the problem becomes a bit harder, as the rewards contain *additional noise* due to unobserved variables. We will start our experiments by considering the simpler former case and then add the additional noise and compare.

The reference solutions that will be derived below are valid in both these cases.

2.1.1. Environment with quadratic cost and quadratic risk control

If the cost- and risk-terms are quadratic and the above dynamics (6)−(8) hold, problem (5) becomes a finite-horizon, discret-time linear−quadratic regulator (LQR).

$$\begin{aligned} &\max_{\{a_t\}} \ \mathbb{E}\left[\sum_{t=0}^{T-1} \pi_{t+1} p_t - \Gamma a_t^2 - \lambda \pi_{t+1}^2\right] \\ &\text{s.t.} \quad (6) \text{ and } (7) \,. \end{aligned} \tag{9}$$

When considering the average expected gain within an infinite horizon as described in[29] (i.e. dividing the global gain by T and taking the limit $T \to \infty$), one can derive a closed-form solution for the optimal control:

$$a_t = -K s_t \,, \tag{10}$$

where K follows from the discrete Riccati-equations and which has a linear dependence on the state.

Without cost term, the solution would be to adapt the portfolio at every time step according to the Markowitz-allocation

$$\pi_{t+1}^* = \pi_{t+1}^{(\mathrm{M})} = \frac{1}{2\lambda} p_t \,. \tag{11}$$

The cost-term adds friction and slows down the trading strategy as compared to the Markowitz allocation and the optimal solution for $t+1$ is a damped version of the Markowitz allocation (11) with the current portfolio.[26] Equivalently the optimal portfolio can be expressed as an exponential moving average of the predictors at $t, t-1, \ldots$.

$$\pi_{t+1}^* = (1-\omega)\pi_t + \omega\psi\pi_{t+1}^{(\mathrm{M})} \,, \tag{12}$$

with $\psi = \frac{\omega}{1-(1-\omega)\rho}$, $\omega = f_c\left(\sqrt{\frac{\lambda}{\Gamma}}\right)$ and $f_c(x) = \frac{2}{1+\sqrt{1+\frac{4}{x}}} = \frac{x}{2}(\sqrt{x^2+4} - x)$.

Note that both ω and ψ have values between 0 and 1.

In all reported experiments with this environment, we used $\Gamma = 1$, $\lambda = 0.3$ and $\rho = 0.9$.

2.1.2. Environment with proportional cost and quadratic risk

Another interesting setting is the case of proportional (or linear) costs:

$$\max_{\{a_t\}} \mathbb{E}\left[\sum_{t=0}^{T-1} \pi_{t+1}^{\top} p_t - \Gamma|a_t| - \lambda\pi_{t+1}^2\right] \tag{13}$$

$$\text{s.t.} \quad (6) \text{ and } (7)\,.$$

For this problem, the optimal stochastic control strategy can be derived[27] and takes the form of a no-trading band system around the (rescaled) predictor's value:

$$\pi_{t+1}^{*} = \begin{cases} u\left(\pi_{t+1}^{(\mathrm{M})}\right) & \text{if } \pi_t > u\left(\pi_{t+1}^{(\mathrm{M})}\right) \\ l\left(\pi_{t+1}^{(\mathrm{M})}\right) & \text{if } \pi_t < l\left(\pi_{t+1}^{(\mathrm{M})}\right) \\ \pi_t & \text{otherwise} \end{cases} \tag{14}$$

The edges $u(\bullet)$ and $l(\bullet)$ are rather non-trivial, but it is a good approximation[27,23,28] to consider the band as being symmetric around the (rescaled) predictor and of constant size:

$$u\left(\pi_t^{(\mathrm{M})}\right) = \pi_t^{(\mathrm{M})} + b \;\text{ and }\; l\left(\pi_t^{(\mathrm{M})}\right) = \pi_t^{(\mathrm{M})} - b \tag{15}$$

In this paper we will use this heuristic as our benchmark, with the parameter b simply given by a numerical grid search.

In all reported experiments with this environment, we used $\Gamma = 4$, $\lambda = 0.3$ and $\rho = 0.9$.

2.1.3. Environment with proportional cost and maxpos risk

Finally, it is also meaningful to consider other forms of risk constraints. In order to test this, we consider the 'maxpos' constraint which imposes a maximum constraint on the absolute positions: $|\pi_t| \leq M$. Formally, that is:

$$\max_{\{|\pi_t| \leq M\}} \mathbb{E}\left[\sum\nolimits_{t=0}^{T-1} \pi_{t+1} p_t - \Gamma|a_t|\right] \tag{16}$$

$$\text{s.t.} \quad (6) \text{ and } (7)\,. \tag{17}$$

The optimal trading strategy is a threshold-based controller that trades to the permitted maximum position whenever the predictor overcomes a value q,[23] i.e.

$$\pi_{t+1}^{*} = \begin{cases} M & \text{if } p_t > q \\ -M & \text{if } p_t < -q\,. \\ \pi_t & \text{otherwise} \end{cases} \tag{18}$$

Again, the subtlety is to find the right value of q, which is derived in23. Here again, we will use a brute-force grid search to find a good approximation of this threshold.

In all reported experiments with this environment, we used $\Gamma = 4$, $M = 2$ and $\rho = 0.9$.

2.2. The algorithm

We selected Deep Deterministic Policy Gradient (DDPG)[34] for the task. It is simple, yet state-of-the-art in continuous control problems and further has been employed previously.[35]

For convenience of the reader we recap the main elements of the DDPG algorithm in Appendix B and outlined the details we employed for successful training. A summary of the DDPG algorithm used in this work is given in the pseudo-code below in Alg. 2.2, and an implementation is available through our repository.

The main ingredients that we added where (a) a prioritized replay buffer and (b) an additional soft thresholding function in the cost-term for the environment with maxpos risk control.

Algorithm 1. DDPG with PER

Algorithm 1 DDPG with PER

Define the predictor AR process
Initialize networks $\mathcal{Q}^{w}$ and ϕ^{Θ}, and the replay buffer of fixed size
Initialize target networks $\widetilde{\mathcal{Q}}^{\widetilde{w}}$ and $\widetilde{\phi}^{\widetilde{\Theta}}$
Initialize the environment
Initialize the exploration noises $\left(\eta_t^{(a)}\right)_{1\leq t\leq T_{pretrain}}$
for $t = 1$ to $T_{pretrain}$ **do**
    Observe state s_t
    Take action $a_t = \phi^{\Theta}(s_t) + \eta_t^{(a)}$
    Observe reward rwd_t and next state s_{t+1}
    Add $(s_r, a_t, \text{rwd}_t, s_{t+1})$ with priority $p = |\text{rwd}_t|$ to the replay buffer
for episode $= 1$ to n **do**
    Initialize the environment
    Initialize the exploration noises $\left(\eta_t^{(a)}\right)_{1\leq t\leq T}$.
    for $t = 1$ to T **do**
        Observe state s_t
        Take action $a_t = \phi(s_t) + \eta_t^{(a)}$
        Observe reward rwd_t and next state s_{t+1}
        Add $(s_r, a_t, \text{rwd}_t, s_{t+1})$ with the highest priority in the buffer
        if $t \equiv 0 \pmod{\tau}$ **then**
            Sample a batch $(s_{t_i}, a_{t_i}, \text{rwd}_{t_i}, s_{t_i+1})_{1\leq i\leq b}$ with prioritized sampling according to probabilities $P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}}$ with $0 \leq \alpha \leq 1$:
            Compute $\forall i \in \{1,\dots,b\} : \widetilde{Q}_i = \text{rwd}_{t_i} + \gamma\widetilde{\mathcal{Q}}\left(s_{t_i+1}, \widetilde{\phi}(s_{t_i+1})\right)$
            Compute $\forall i \in \{1,\dots,b\} : \delta_i = |\mathcal{Q}(s_i, a_i) - \widetilde{Q}_i|$
            Compute weights $\forall i \in \{1,\dots,b\} : \alpha_i = \left(\frac{1}{N}\frac{1}{P(i)}\right)^{\beta}$
            Normalize weights $\forall i \in \{1,\dots,b\} : \alpha_i \leftarrow \frac{\alpha_i}{\max_k(\alpha_k)}$
            Update priorities $\forall i \in \{1,\dots,b\} : p_i = |\delta_i|$
            Gradient update : $\mathcal{L}_{critic}(w) = \frac{1}{b}\sum_{i=1}^{b}\alpha_i\delta_i^2$
            Gradient update : $\mathcal{L}_{actor}(\Theta) = -\frac{1}{b}\sum_{i=1}^{b}\mathcal{Q}\left(s_{t_i}, \phi^{\Theta}(s_{t_i})\right)$
            Update target networks

$$\tilde{w} \leftarrow \tau_{critic} w + (1 - \tau_{critic})\,\tilde{w}$$
$$\widetilde{\Theta} \leftarrow \tau_{actor}\Theta + (1 - \tau_{actor})\,\widetilde{\Theta}$$

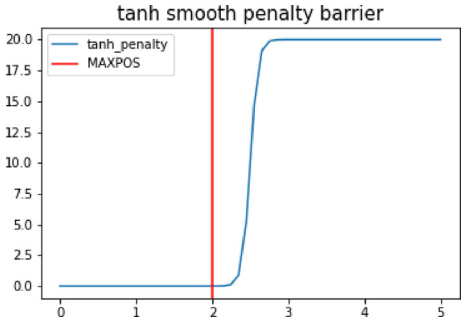

Fig. 1. tanh barrier penalty used to stabilize training.

Since the maxpos risk constraint is not differentiable we need an additional trick to adapt DDPG to enable better convergence. The first ingredient is to simply clip the positions resulting from the actor network's actions such that the constraint is not violated. However, since this leads to poor gradient estimations, we add an additional cost to the reward function, using a smooth penalty for positions with magnitude beyond the maxpos, as depicted on Fig. 1. The reward function becomes:

$$\mathrm{rwd}(p,\pi,a) = p\pi - \psi|a| - \beta\{\tanh[\alpha(|\pi+a| - (1+\gamma)M)] + 1\}\,.$$

In our experiments we chose $\beta = 10, \alpha = 10, \gamma = \frac{1}{4}$. The addition of the tanh-barrier is crucial to stabilize training. Without it, out of the 16 agents we trained, half diverged even under perfect state observations. The penalty prevents this divergence (at least under perfect state observability). The use of tanh is justified by its smooth and upper bounded nature, we tested a constant penalty and an exponential one with no success.

3. Results

We start by reporting the results of the training of the algorithm for all three environments in Fig. 2. The following conclusions can be drawn:

- For all three environments the algorithm successfully learns trading strategies with close-to-optimal rewards and PnLs. See also the tables for further quantitative comparison.
- The PnL achieves close-to-optimal performance faster than the rewards, hinting at the fact that the algorithm first learns to follow the signal and to control the cost before it fine-tunes the risk-control. This is also represented in the wider spread of the points in the early stages of training and more concentration in the end.
- The most challenging environment is the third environment, with linear cost and maxpos risk control for which not even all agents converged.
- The environments for which additional noise is present in the rewards are harder: lower overall performance is achieved and there is more variability in the results.[2]

Besides reporting the performance in terms of the accomplished rewards it is enlightening to take a look at the obtained policies. In Figs. 3−5 we compare the RL policies achieving highest reward for each of the three environments with their respective reference policies. The figures present trading strategies in two different ways: (a) by depicting the proposed trade a_t against the predictor p_t for three specific representative positions π_t. And (b) by depicting contour plots of the proposed trade for all points in the (π_t, p_t)-plane, such that $|a_t| \leq 5$. Similar figures for the experiments under additional noise in the rewards can be found in Appendix A.

The experiments on the environment with quadratic trading cost and risk control reliably yield high rewards and from Table 1 we further draw the conclusion, that the resulting trading strategies only show very small deviations with

[2] The noise-to-signal ratio applied during the training was $\sigma(\eta_t^{(r)}) = 10$ for the environments with quadratic risk penalty and $\sigma(\eta_t^{(r)}) = 4$ for the one with maxpos risk.

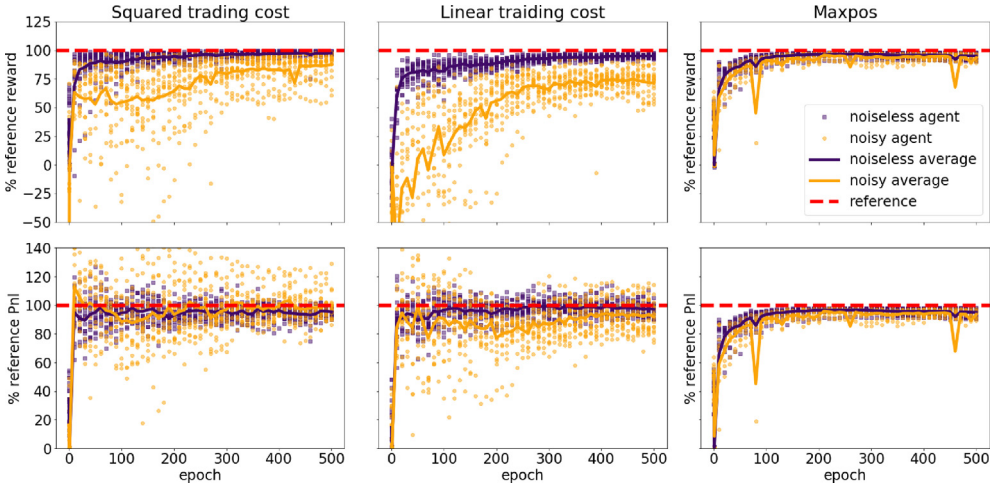

Fig. 2. Results for the setting where the agent has perfect state information (purple markets) and under additional noise in the returns (yellow markers). In both settings we report the theoretical rewards and PnLs under perfect state information. The red dashed horizontal line represents the reference agents from section 2 and we plot the achieved relative performance difference w.r.t. the reference. For every reported epoch each point represents one agent, i.e. a different random seed for the learning. The performances are measured as the mean over ten out-of-sample environments, each $T = 5000$ time steps long. The solid colored lines represent the average over the 16 different random seeds, i.e. the average among agents. For the 'maxpos' environment we report only those agents that converged, hence there are fewer points and the figure is strongly biased towards the positive results.

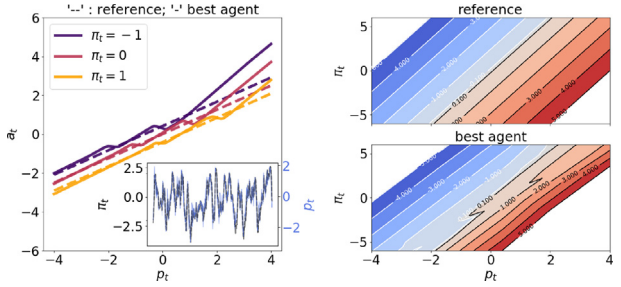

Fig. 3. Comparison of the highest-reward RL policy with the reference policy for the environment with quadratic cost and risk control. Left: the actions a_t are plotted against the predictor p_t for different positions $\pi_t \in \{-1, 0, 1\}$. The dashed line represent the reference strategies and the solid lines the RL agent. The inset compares the learnt policy with the reference on a small snapshot from simulated predictor's trajectories. Right: the contour-plots compare the reference policy (upper panel) with the RL policy (lower panel) in the whole (π_t, p_t)-plane.

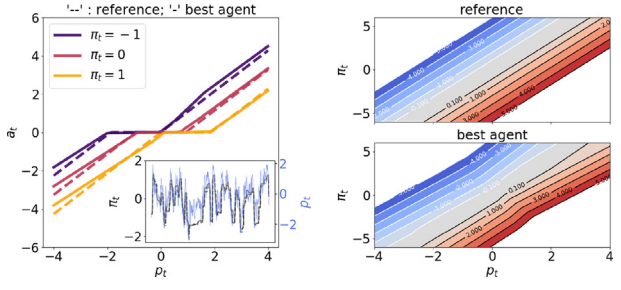

Fig. 4. As Fig. 3, but for the environment with linear trading cost and quadratic risk control.

respect to the reference. Nonetheless, Fig. 3 shows that the learned policies sometimes exhibit sub-optimal features. However, these deviations have little negative impact in terms of the reward. Indeed we often found that, in the early stages of learning, the algorithm would converge to policies very close to the reference and then become unstable and converge towards slightly sub-optimal forms.

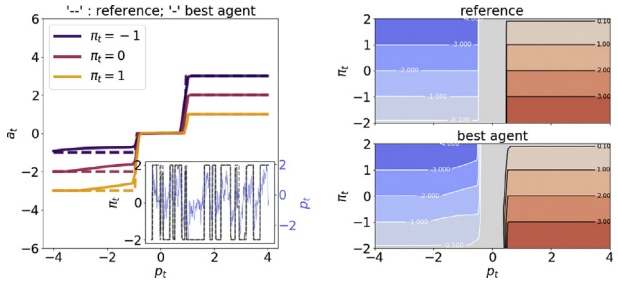

Fig. 5. As Figs. 3 and 4, but for the environment with linear trading cost and maxpos risk control.

Table 1
Summary of the performance achieved by the agents for the environment with quadratic trading cost and quadratic risk control. For 16 agents the 75%-tile corresponds to the worst of the four best agents etc. The 'Diff' calculated in the tables is the l_1 norm of the difference between the positions taken by the agent and those taken by the reference solution.

	Reward		PnL		Diff	
	no noise	noise	no noise	noise	no noise	noise
Reference	0.681		1.298		0	
Best	0.677	0.671	1.291	1.674	0.081	0.128
Mean	0.665	0.596	1.237	1.295	0.140	0.383
Worst	0.655	0.415	1.170	1.119	0.218	0.781
75%-tile	0.668	0.640	1.258	1.316	0.161	0.491
50%-tile	0.666	0.624	1.239	1.276	0.132	0.349
25%-tile	0.660	0.588	1.215	1.215	0.106	0.256

When looking for the optimal reference strategies with a grid-search one finds that the reward landscape w.r.t. the parameters is rather flat around the optimal value. Thus many different solutions can yield almost identical results (not only for this environment). Note also that the far ends of the policies have little influence on the resulting trading trajectories as they are visited with small probability, permitting for larger deviations in these regions without much negative impact on the rewards.

For the environment with linear cost and quadratic risk control the situation is rather similar. Clearly the RL agents retrieve the most essential features of the optimal policy. The no-trading zones are recovered with high precision and also the slopes outside of the no-trading zones are of an acceptable (close to linear) form. Especially in those regions that are most explored by the predictor. Typically we find that the agents first learn to trade into the direction of the signal and to avoid cost, before the slopes in the trading zones are fine tuned to improve risk control. In terms of the rewards the solution is a little less close to the reference performances as can be gathered from Table 2. This should be expected for a more challenging environment.

Fig. 5 presents the best learnt policies for the maxpos environment. The agents successfully learn to take large trades once the expected gain overcomes the cost and not to trade below this threshold. The obtained rewards

Table 2
Summary of the performance achieved by the agents for the environment with linear trading cost and quadratic risk control.

	Reward		PnL		Diff	
	No noise	Noise	no noise	noise	no noise	noise
reference	0.254		0.492		0	
best	0.248	0.225	0.518	0.562	0.063	0.131
mean	0.241	0.181	0.478	0.452	0.093	0.250
worst	0.234	0.135	0.442	0.364	0.126	0.441
75%-tile	0.244	0.196	0.491	0.486	0.101	0.301
50%-tile	0.239	0.188	0.482	0.455	0.090	0.244
25%-tile	0.238	0.167	0.464	0.414	0.080	0.181

Table 3
Summary of the performance achieved by the agents for the environment with linear trading cost and maxpos risk control, '-' means that some agents diverged.

	Reward		PnL		Diff	
	no noise	Noise	no noise	noise	no noise	noise
reference	0.901		0.901		0	
best	0.884	0.876	0.884	0.876	0.101	0.143
mean	0.856	0.842	0.856	0.842	0.198	0.246
worst	0.815	0.803	0.815	0.803	0.321	0.346
75%-tile	0.849	0.849	0.849	0.849	0.148	0.210
50%-tile	0.862	–	0.862	–	0.184	–
25%-tile	0.826	–	0.826	–	0.239	–

(whenever the training converged) are rather close to optimal and the situation is similar to the previously reported environments as is further quantified in Table 3.

The environment with maxpos risk control is the most challenging one algorithmically. The optimal threshold control is conceptually simple, but challenging to learn with a model-free continuous control algorithm, such as DDPG. We found that it was necessary to add additional tricks for this environment in order to obtain reliable results. We experimented with different ways to incorporate the maxpos constraint and found that the most reliable approach is to combine a clipping of the position in the environment with an additional soft threshold in the reward functions (cf. algorithmic section). One problem is simply the shape of the function, which poses a challenge because of the non-continuity around the non-trading region. Another challenge is that the variance of the portfolio is not controlled: the optimal trading strategy is to hold the position constant most of the time and only scarcely change the position, which causes a lot of variability in the rewards that are observed for a (π_t, p_t)-pair.

Overall it can be said that DDPG successfully recovers the essential features of the reference trading strategies. A remarkable fact is the reliable recovery of the no-trading region of non-trivial width and trading regions of appropriate slope. This shows that the RL agents learn all of the required features: (a) to exploit the auto-correlation of the predictor (b) to balance risk, cost and gain appropriately. Both in the setting under perfect state information and under additional noise in the rewards, the PnLs and rewards get very close to optimal. In the appendix and the tables we provide some additional material. Further more we provide figures, similar to Figs. 3−5 for all trained agents as supplemental material.

However, to obtain reliable results, the DDPG algorithm required fine adjustments, especially in the more challenging environment with 'maxpos' constraints. It is reasonable to believe that we could have obtained faster convergences and more accurate results with more adjustments and/or better heuristics. For example, instead of using an ε-greedy exploration scheme, one could have considered using a parameter space noise as reported in.[36]

4. Conclusions

The aim of this paper was to demonstrate the potential of model-free reinforcement learning methods to derive trading policies. We first established some reference portfolio construction problems for which we are either able to derive analytically the trading policy or approximate it well with simple optimization procedures. We then compared this baseline policies with the one derived by the DDPG-RL method.

Overall it can be concluded that the DDPG-RL agents successfully recover the established baselines. This is already a non-trivial task. Moreover, the resulting trading strategies are very close to optimal and a high quantitative agreement with the reference strategies have been obtained. Beyond these results, we hope that the different RL environments we built around trading optimization will be able to serve as reference test-cases in further studies.

Finally, one can notice that a specificity of the model-free RL approach we used is that the structure of the reward function is assumed unknown. However, for the studied problems the differentiable structure of the reward function could have been exploited to design more efficient algorithms. Such a model-based approach could be studied in future work.

Conflict of interest

The authors declare that they have no known competing financial interest or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research was partly conducted within the Econophysics & Complex Systems Research Chair, under the aegis of the Fondation du Risque, the Fondation de l'École Polytechnique, the Ecole polytechnique and Capital Fund Management.

We would like to thank Nathanaël Foy, Alexis Sar and Julien Lafaye for regular discussions on the content of this article, and Charles-Albert Lehalle for sharing his expertise on the topic.

Appendix A. Results under additional noise in the rewards

Here we present the results when the observed rewards contain an additional noise in the returns, instead of the actual values of the predictor. The resulting signal has a noise-to-signal-ratio of 10 for the cases with quadratic risk control, and 4 for the maxpos risk control since its convergence is less stable.

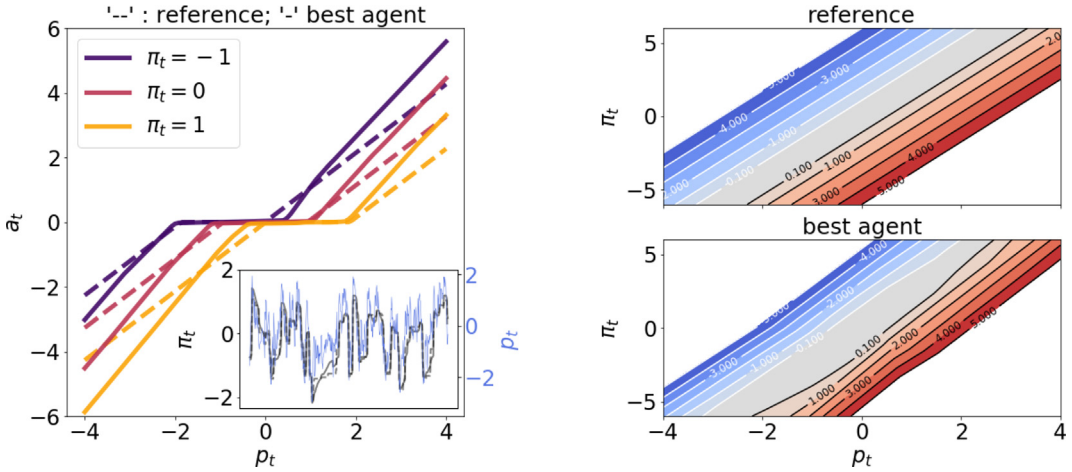

Figure A.6. Highest-reward RL agent for the environment with quadratic trading cost and quadratic risk control. As Fig. 3, but with additional noise in the rewards.

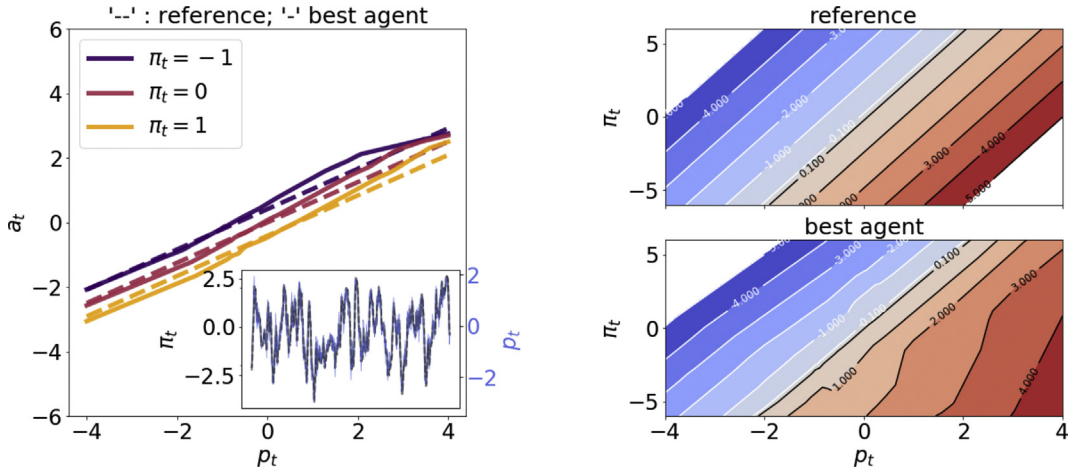

Figure A.7. Highest-reward RL agent for the environment with proportional trading cost and quadratic risk control. As Fig. 4, but with additional noise in the rewards.

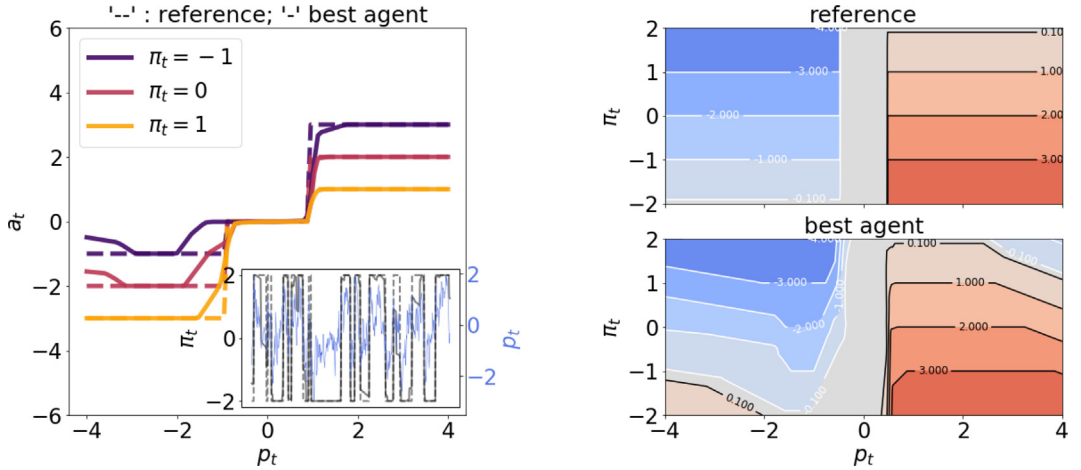

Figure A.8. Highest-reward RL agent for the environment with proportional trading cost and maxpos risk control. As Fig. 5, but with additional noise in the rewards.

Appendix B. Details of the algorithm

Let us briefly recall the basic elements of DDPG. DDPG is an actor-critic architecture where the actor estimates a deterministic policy while the critic estimates the state-action value function Q. This algorithm is training off-policy relying on the off-policy deterministic policy gradient theorem 37.

Formally, let $\beta(a|s)$ denote a behaviour policy that generates the training samples and ρ^β be the state distribution under β. The objective function in the off-policy setting as defined in 38 is

$$J^\beta(\Theta) = \int_{s\in S} \rho^\beta(s) V_\Theta(s) ds \qquad = \int_{s\in S} \rho^\beta(s) Q_\Theta(s, \phi_\Theta(s)) ds.$$

Here ϕ_Θ stands for our parametric deterministic policy with parameter Θ, S the state space and $Q_\Theta = Q^{\phi_\Theta}$ the state-action value function of policy ϕ_Θ.

The off-policy policy gradient theorem states:

$$\nabla_\Theta J^\beta(\Theta) \approx \mathbb{E}_{s\sim\rho^\beta}[\nabla_\Theta \phi_\Theta(s) \nabla_a Q_\Theta(s,a)|_{a=\phi_\Theta(s)}].$$

This estimation is useful as it can be used to update Θ with gradient ascent, without having to take into account the dependency of Q_Θ on Θ.

In off-policy training the agent interacts with the environment to gather experiences (training samples in the form of tuples $(s_t, a_t, \text{rwd}_t, s_{t+1})$ of state, action, reward, next state) in a replay buffer, then samples training batches in order to update first the critic and then the actor (see Fig. B.9).

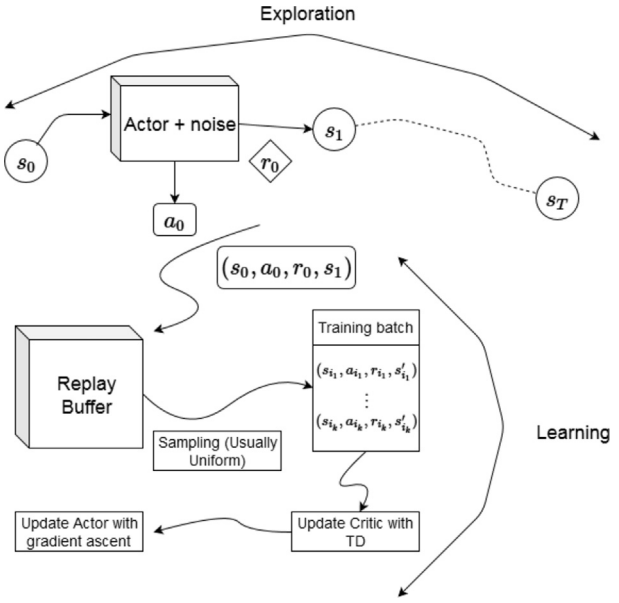

Figure B.9. Off-policy training summary.

Exploration

An episode represents a walk through the environment, the agent always starts with a position $\pi_0 = 0$. The agent interacts with the environment by taking actions $a_t = a_t^{pred} + \eta_t^{(a)}$, given the current position π_t and the predictor p_t:

$$\begin{cases} a_t^{pred} & = \phi^\Theta(p_t, \pi_t) \\ \eta_t^{(a)} & = (1-\rho^{\text{expl}})\eta_{t-1}^{(a)} + \sigma^{\text{expl}}\varepsilon_t; \ . \\ \eta_0^{(a)} & = 0 \end{cases}$$

The $\eta_t^{(a)}$ is the exploration noise, following an autoregressive process, with i.i.d. ε_t s.t. $\varepsilon_t \sim N(0,1), \sigma^{\text{expl}} > 0$.

Learning

Every τ steps we sample a training batch $(s_{t_j}, a_{t_j}, \text{rwd}_{t_j}, s_{t_j+1})_{1\le j\le b}$ of size b from the replay buffer. This decorrelates experiences within a batch so that gradient estimation is less biased.

Critic update. Let Q, $\tilde{Q}$, ϕ and $\tilde{\phi}$ denote respectively the current critic, target critic, current actor and target actor networks with their respective parameters $w, \tilde{w}, \Theta, \tilde{\Theta}$. For a given experience i in the sampled training batch, the target Q-value is computed with the target critic network where the action is chosen with the target actor network $\tilde{\phi}$

$$\tilde{Q}_i = \text{rwd}_{t_i} + \gamma\tilde{Q}(s_{t_i+1}, \tilde{\phi}(s_{t_i+1}))$$

And the Q-value is simply computed with the current critic network

$$Q_i = Q(s_{t_i}, a_{t_i})$$

Then the critic loss is calculated with temporal difference:

$$L_{critic}(w) = \frac{1}{2b}\sum_{i=1}^{b}(\tilde{Q}_i - Q_i)^2$$

where $b > 0$ is the batch size.

Actor update. Using the off-policy deterministic policy gradient theorem the current actor network is updated with gradient descent on the following loss function:

$$L_{actor}(\Theta) = -\frac{1}{b}\sum_{i=1}^{b} Q(s_i, \phi_\Theta(s_i)) \, .$$

Target networks update The parameters of target Critic and target Actor networks are updated with soft target updates

$$\tilde{w} \leftarrow \tau_{critic} w + (1 - \tau_{critic})\tilde{w}$$

$$\tilde{\Theta} \leftarrow \tau_{actor} \Theta + (1 - \tau_{actor})\tilde{\Theta}$$

where $0 < \tau_{critic} < 1$ and $0 < \tau_{actor} < 1$, with this method, the networks parameters get updated slowly, which makes training more stable.

The replay buffer. Different schemes may be applied when sampling from the replay buffer. We choose prioritized experience replay 39 because it can speed up training and improve convergence. In prioritized experience replay buffers, experiences are weighted according to their TD error

$$\delta(s, a, \mathrm{rwd}, s\prime) = \mathrm{rwd} + \gamma \max_{a\prime} Q(s\prime, a\prime) - Q(s, a)$$

and are sampled according to the distribution

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, 1 \le i \le N, 0 \le \alpha \le 1$$

where N is the replay buffer size, $p_i = |\delta_i| + \varepsilon$ the priority with ε is a small positive number ensuring non-zero sampling probability for all samples.

The intuition is that experiences with high magnitude TD errors are those poorly evaluated by the critic, so increasing their learning frequency makes sense.

Notice that non-uniform sampling introduces a bias in the gradient estimation that can be corrected with importance sampling weights 40 by using $\alpha_i \delta_i$ instead of δ_i in updating the critic, with

$$\alpha_i = \left(\frac{1}{N}\frac{1}{P(i)}\right)^\beta, 0 \le \beta \le 1 \, .$$

The exponent β represents the emphasis put into correcting the bias. In our experiments this parameter is linearly annealed from an initial value β_0 to unity as correcting the bias is not as important in the beginning as it is near convergence.

References

1. Henderson P, Islam R, Bachman P, Pineau J, Precup D, Meger D. Deep reinforcement learning that matters. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.
2. Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of go without human knowledge. *Nature*. 2017;550(7676):354.
3. Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. *Nature*. 2015;518(7540):529.
4. Moravčík M, Schmid M, Burch N, et al. Deepstack: expert-level artificial intelligence in heads-up no-limit poker. *Science*. 2017;356(6337):508−513.
5. Brown N, Sandholm T, Machine S. libratus: the superhuman AI for No-limit poker. In: *in: IJCAI*. 2017:5226−5228.

6. Deng Y, Bao F, Kong Y, Ren Z, Dai Q. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*. 2016;28(3):653−664.
7. Wu D, Wang L, Zhang P. Solving statistical mechanics using variational autoregressive networks. *Phys Rev Lett*. 2019;122(8), 080602.
8. Levine S, Finn C, Darrell T, Abbeel P. End-to-end training of deep visuomotor policies. *J Mach Learn Res*. 2016;17(1):1334−1373.
9. Pan X, You Y, Wang Z, Lu C. *Virtual to Real Reinforcement Learning for Autonomous Driving*. 2017.
10. François-Lavet V, Taralla D, Ernst D, Fonteneau R. Deep reinforcement learning solutions for energy microgrids management. In: *European Workshop on Reinforcement Learning*. EWRL; 2016:2016.
11. G. Ritter, Machine learning for trading, Soc Sci Res Network URL https://doi.org/10.2139/ssrn.3015609.
12. Dütting P, Feng Z, Narasimhan H, Parkes D, Ravindranath SS. *Optimal auctions through deep learning*. In: *Proceedings of the 36th International Conference on Machine Learning*. vol. 97. PMLR; 2019:1706−1715.
13. Moody J, Saffell M. Learning to trade via direct reinforcement. *IEEE Trans Neural Network*. 2001;12(4):875−889.
14. Yu P, Lee JS, Kulyatin I, Shi Z, Dasgupta S. *Model-based Deep Reinforcement Learning for Dynamic Portfolio Optimization*. 2019.
15. Neuneier R. Optimal asset allocation using adaptive dynamic programming. In: *Adv Neural Inf Process Syst*. 1996:952−958.
16. Recht B. A tour of reinforcement learning: the view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*. 2019;2:253−279.
17. Hutson M. Artificial intelligence faces reproducibility crisis. *Science*. 2018;359(6377):725−726. https://doi.org/10.1126/science.359.6377.725. ISSN 0036-8075, doi: ＼let＼@tempa＼bibinfo@X@.
18. Markowitz H. Portfolio selection. *J Finance*. 1952;7(1):77−91.
19. Chamberlain G. A characterization of the distributions that imply mean−−variance utility functions. *J Econ Theor.* 1983;29(1):185−201.
20. Watkins CJCH. *Learning from Delayed Rewards*. Ph.D. thesis. Cambridge: King's College; 1989.
21. Sutton RS, Barto AG. *Reinforcement Learning: An Introduction*. MIT press; 2018.
22. Merton RC. Lifetime portfolio selection under uncertainty: the continuous-time case. *Rev Econ Stat*. 1969:247−257.
23. de Lataillade J, Deremble C, Potters M, Bouchaud J-P. *Optimal Trading with Linear Costs*. 2012.
24. Abeille M, Serie E, Lazaric A, Brokmann X. *LQG for Portfolio Optimization*. 2016.
25. Bertsimas D, Lo AW. Optimal control of execution costs. *J Financ Mark*. 1998;1(1):1−50.
26. Gârleanu N, Pedersen LH. Dynamic trading with predictable returns and transaction costs. *J Finance*. 2013;68(6):2309−2340.
27. R. Martin, T. Schoeneborn, Mean Reversion Pays, but Costs, RISK .
28. Muhle-Karbe J, Reppen M, Soner HM. A primer on portfolio choice with small transaction costs. *Annual Review of Financial Economics*. 2017;9:301−331.
29. Bertsekas DP. *Dynamic Programming and Optimal Control*. vol. 1. MA: Athena scientific Belmont; 1995.
31. Moody JE, Saffell M. Reinforcement learning for trading. In: *Adv Neural Inf Process Syst*. 1999:917−923.
32. François-Lavet V, Henderson P, Islam R, Bellemare MG, Pineau J. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*. 2018;11(3−4):219−354.
33. Kong W, Liaw C, Mehta A, Sivakumar D. A new dog learns old tricks: RL finds classic optimization algorithms. In: *International Conference on Learning Representations*; 2019. URL https://openreview.net/forum?id=rkluJ2R9KQ.
34. Lillicrap TP, Hunt JJ, Pritzel A, et al. *Continuous Control with Deep Reinforcement Learning*. 2015.
35. Xiong Z, Liu X-Y, Zhong S, Yang H, Walid A. *Practical Deep Reinforcement Learning Approach for Stock Trading*. 2018.
36. Plappert M, Houthooft R, Dhariwal P, et al. Parameter Space Noise for Exploration; 2017.