Deep differentiable reinforcement learning and optimal trading

Thibault Jaisson

To cite this article: Thibault Jaisson (2022) Deep differentiable reinforcement learning and optimal trading, Quantitative Finance, 22:8, 1429-1443, DOI: 10.1080/14697688.2022.2062431

To link to this article: https://doi.org/10.1080/14697688.2022.2062431

Quantitative Finance, 2022
Vol. 22, No. 8, 1429–1443, https://doi.org/10.1080/14697688.2022.2062431

Deep differentiable reinforcement learning and optimal trading

THIBAULT JAISSON*

Pictet Asset Management Ltd, Geneva, Switzerland

(*Received 6 December 2021; accepted 25 March 2022; published online 8 June 2022*)

In many reinforcement learning applications, the underlying environment reward and transition functions are explicitly known differentiable functions. This enables us to use recent research which applies machine learning tools to stochastic control to find optimal action functions. In this paper, we define differentiable reinforcement learning as a particular case of this research. We find that incorporating deep learning in this framework leads to more accurate and stable solutions than those obtained from more generic actor critic algorithms. We apply this deep differentiable reinforcement learning (DDRL) algorithm to the problem of one asset optimal trading strategies in various environments where the market dynamics are known. Thanks to the stability of this method, we are able to efficiently find optimal strategies for complex multi-scale market models. We also extend these methods to simultaneously find optimal action functions for a wide range of environment parameters. This makes it applicable to real life financial signals and portfolio optimization where the expected return has multiple time scales. In the case of a slow and a fast alpha signal, we find that the optimal trading strategy consists in using the fast signal to time the trades associated to the slow signal.

Keywords: Reinforcement learning; Optimal trading; Differentiable; Stochastic control; Dynamic programming; Deep learning; Multi-scale signals

1. Introduction

Reinforcement learning is a field of machine learning which aims at finding optimal sequences of actions in order to achieve a multi-step task or maximize a cumulative reward. Its combination with deep learning, see LeCun *et al.* (2015), called deep reinforcement learning (DRL), see François-Lavet *et al.* (2018), has recently had a huge success in addressing sequential decision problems. For example, it outperforms the best humans at many games, see Mnih *et al.* (2015), and has state of the art applications in many real life applications such as robotics, see Lillicrap *et al.* (2015), self-driving cars, see Tai *et al.* (2017) and finance, see Cartea *et al.* (2021).

Quantitative portfolio management is the use of mathematical and statistical tools to devise ‘optimal’ portfolios of financial assets. Historically, this field mostly focused on getting the best expected return (or alpha in the following) with the smallest possible risk as in the mean variance framework of Markowitz, see Markowitz (1968). However, this does not take into account the practically paramount role of trading costs: An investor needs to control his turnover if he does not want to spend his alpha in trading costs, see Frazzini *et al.* (2012). This implies that the current choice of portfolio weights does not only impact the portfolio returns over the next short period but also the future investor positioning. This naturally puts us in the setting of reinforcement learning where the current actions (trades) have an impact on the next reward as well as on the future state of the environment which itself has an impact on future rewards, see Sutton and Barto (2018).

Over the last decade, the use in finance of modern machine learning techniques such as deep learning has seen a huge increase to: price options, see Horvath *et al.* (2021), predict returns, see Gu *et al.* (2020) and model risks, see Gu *et al.* (2021). In spite of the challenges inherent to financial data such as the structural low signal to noise ratio and non stationarity, adding non linearity and interactions between usual features seems to add value compared to more classical linear models, see Daul *et al.* (2021). Deep reinforcement learning has also been extensively applied to directly build optimal strategies from real financial data, see Deng *et al.* (2016) and Cong *et al.* (2020), by minimizing some loss of the portfolio performance. These approaches have the particularity that they do not split the portfolio construction

*Corresponding author. Email: tjaisson@pictet.com process in two steps: First modeling and estimating the dynamics of the market (which is typically a data science issue with no mathematically perfect solution). Second finding the optimal strategy in this model (which is an engineering problem with a mathematically well defined solution). This however makes it harder for an investor to understand his portfolio and what criteria explain the asset weights.

An alternative approach taken in Chaouki *et al.* (2020), which we follow here, is to assume that the market is well modeled and its parameters are known. In this model, we can then use deep reinforcement learning to find the optimal strategy. The advantage of their approach is that one can always simulate as much data as one needs and evaluate strategies as precisely as necessary. They use the state of the art Deep Deterministic Policy Gradient (DDPG). This algorithm tends to be hard to train and requires much engineering, meta-parameters tweaking and to be trained on different starting points to be well fitted in practice, see Fujimoto *et al.* (2018). Even then, it is only applied to a very simple $AR(1)$ alpha model and when more complex models are given as input, the strategies obtained do not converge towards the optimal ones.

In particular, in this article, we are interested in devising optimal trading strategies in environments where the alpha process has a multi-time scale behavior, see Gârleanu and Pedersen (2013) and Boyd *et al.* (2017). Indeed, in practice, investors have access to return predictors with different reversion time scales. For example, Value, Momentum, see Asness *et al.* (2013), and Quality, see Asness *et al.* (2019), can be considered as slow signals, indeed if a stock has a good Quality signal it will most likely remain good in the coming months. On the other hand, price reversal, see Bremer and Sweeney (1991), and analyst recommendations, see Jaisson *et al.* (2021), are much faster signals with shorter memory. We tried to apply the DDPG algorithm to such multi-scale trading environments however, even after many attempts, we did not manage to make it converge to optimal strategies.

To solve this issue, we follow recent research which applies machine learning tools to stochastic control problems, see Han (2016), Fécamp *et al.* (2019), and Germain *et al.* (2021). It leverages on the fact that many reinforcement learning problems can be cast as optimal control where the underlying dynamics of the environment are known. This enables to precisely anticipate the future impact of actions and to directly maximize the cumulative expected reward. Looking for action functions which take the form of a feed forward neural network (FFNN) gives an algorithm which derives optimal trading strategies in the mono-asset case. We call this framework deep differentiable reinforcement learning. Furthermore we thoroughly analyze and describe the interactions between the slow and fast signals in the optimal action. We find that the slow signal is more correlated to the portfolio weights and the fast signal is more correlated to the trades. In other words, the fast signal is used to time the trades associated to the slow signal.

This paper is organized as follows: In Section 2, we introduce the differentiable reinforcement learning framework and a generic algorithm to find optimal action functions approximated by a deep neural network. In Section 3, we show how generic (multi-scale) optimal trading problems fit into the context of differentiable reinforcement learning. In Section 4, we exhibit and describe the properties of optimal trading strategies. We conclude on how to apply this in practice in Section 5.

2. Differentiable reinforcement learning

2.1. *The reinforcement learning framework*

In the standard reinforcement learning setup, see Sutton and Barto (2018), a learning agent is interacting with his environment in discrete time in the following way: At each time t, he sees the current sate of the environment s_t (in some settings the agent only sees a noised version of the environment but this is irrelevant in what follows) and chooses an action a_t. The environment then samples a reward r_t to the agent and shows it the next state s_{t+1}. The aim of the agent is to find a sequence of actions which maximize the expectation of the cumulative reward up to time T which is defined as

$$\mathrm{CR}_T = \sum_{i=0}^{T-1} r_i. \tag{1}$$

What makes the problem challenging is that both the reward and the next state (and thus future rewards) depend on the actions of the agent which thus have short-term and long-term impact.

Note that in some reinforcement learning settings, the objective of the agent is to maximize the infinite sum of discounted future rewards, see Lillicrap *et al.* (2015). If the discounting factor is close to one and T is large, the optimal action functions for these two objectives should be very similar. However, as we will see, having a large but finite sum enables us to minimize it efficiently.

A noteworthy point is that as opposed to the more classical optimal control setting, see Bertsekas (2012), in the reinforcement learning setting, the underlying dynamics of the environment are not necessarily known by the agent. From his point of view, the environment is simply a black box from which we can sample state transitions and rewards by giving it actions.

Until recently, the choice of an optimal action function was only solvable either in very specific settings where a closed form solution exists or where both the action and the state spaces are finite and rather small, see Watkins and Dayan (1992). In the last decade, the combination of ideas from optimal control with the flexibility of deep learning has enabled astonishing progress in the context of self-driving cars, see Tai *et al.* (2017), games, see Mnih *et al.* (2015) and robotics, see Lillicrap *et al.* (2015). First for discrete action spaces, see Mnih *et al.* (2015), and even more recently for continuous and potentially high dimensional action spaces. For example Lillicrap *et al.* (2015) introduced the Deep Deterministic Policy Gradient (DDPG) algorithm which jointly learns the optimal action and the value function by approximating them with deep neural networks and using the Bellman equation.

However, even if progress is made every year in this direction, see Fujimoto *et al.* (2018), this kind of methods called actor critic are still hard to train. This is especially the case when the expected cumulative reward is flat around the optimal action where they require much tweaking of the meta-parameters and engineering of the algorithm, see Ioffe and Szegedy (2015) and Henderson *et al.* (2018). Even then, the results are far from perfect and need to be trained on many learning paths with different starting points to get closer to the optimal solution, see Chaouki *et al.* (2020) and Appendix 3.

In the next paragraph, we introduce the differentiable reinforcement learning setting. It is very close to that of recent works which apply neural networks to solve stochastic control problems, see Han (2016), Fécamp *et al.* (2019), and Germain *et al.* (2021) . These approaches use the fact that in many applications such as the one of Chaouki *et al.* (2020), the underlying dynamics of the environment are known. Moreover, in a sense that we will make precise, these dynamics are differentiable. In this setting, we follow these works to devise a conceptually simple algorithm which finds optimal action functions that are far more accurate and stable than the ones obtained from generic reinforcement learning methods, see Appendix 3.

Applied to the problem of optimal trading strategies, it enables us to optimize more complex market models with a large number of meta-parameters.

2.2. *The differentiable reinforcement learning framework*

We define the differentiable reinforcement learning framework as a particular case of discrete time stochastic control:

- The next state s_{t+1} is a known differentiable transition function Φ of the current state s_t, the chosen action a_t and a random variable U_{t+1} that we can sample from
$$s_{t+1} = \Phi(s_t, a_t, U_{t+1}). \tag{2}$$
- The reward r_t is a known differentiable function Ψ of the current state s_t, the chosen action a_t and a random variable V_t that we can sample from
$$r_t = \Psi(s_t, a_t, V_t). \tag{3}$$

In this framework, the variables U and V fully capture the randomness of the environment.

The aim of optimal control problems is to find an action function A such that

$$a_t = A(s_t)$$

which maximizes the expectation of the long-term cumulative reward. Note that in Han (2016) and Fécamp *et al.* (2019), the authors either fit one function per time to maturity of take the time to maturity as an input of the action function. In the following, we consider ‘stationary’ environments where the horizon T is not a real objective but a technical parameter that just needs to be large enough compared to the environment time scales. To reduce the input dimension of the action function, we do not take time to maturity as input.

Note that applications that take as input real world data instead of simulated ones, do not make it into this framework. However, when working with known environments that we can simulate we are mostly in this setting.

2.3. *The cumulative reward function*

An important property of this framework is that given an action function A the cumulative reward CR_T of the strategy is only a function of A, s_0, $(V_t)_{0\le t\le T-1}$ and $(U_t)_{0\le t<T-1}$:

$$\mathrm{CR}_T = \mathrm{CR}_T(A, s_0, (V_t)_{0\le t\le T-1}, (U_t)_{0\le t<T-1}). \tag{4}$$

For example, the rewards for $T = 1$, 2 and 3 are shown below.

- $\mathrm{CR}_1 = \Psi(s_0, A(s_0), V_0)$.
- $\mathrm{CR}_2 = \mathrm{CR}_1 + \Psi(\Phi(s_0, A(s_0), U_0), A(\Phi(s_0, A(s_0), U_0)), V_1)$.
- $\mathrm{CR}_3 = \mathrm{CR}_2 + \Psi(\Phi(\Phi(s_0, A(s_0), U_0), A(\Phi(s_0, A(s_0), U_0)), U_1), A(\Phi(\Phi(s_0, A(s_0), U_0), A(\Phi(s_0, A(s_0), U_0)), U_1)), V_2)$.

Of course for T large, this function is quite deep in the sense that it is a composition of hundreds of base functions. However, we will see that we are able to efficiently minimize its expectation with respect to the action function for large T (up to at least 100 which is more than enough for our applications).

2.4. *Deep learning application*

In order to numerically find the action function which maximizes the expectation of the cumulative reward up to a given time T, we need to restrict it to a family of parametrized action functions, with parameters denoted θ:

$$A(s) = F(s; \theta). \tag{5}$$

Assuming that F is differentiable with respect to these parameters θ, the cumulative reward samples are also differentiable with respect to θ. To numerically compute the associated derivatives of these deep composition of functions, we use automatic differentiation (AD) and backpropagation through the Pytorch package, see Paszke *et al.* (2017). A stochastic gradient descent (SGD) like algorithm can then be used to find the action parameters which minimize the empirical sample average of the cumulative reward.

A flexible extensively studied and numerically convenient parametrization of the action function used is a similar manner in Fécamp *et al.* (2019) is the dense feed forward neural network or multi-layer perceptron, see Schmidhuber (2015) for the definition. In what follows, unless stated otherwise, we take a neural network with two hidden layers with 300 hidden neurons each and *ReLU* activation functions so that the model has around 10’000 parameters.

2.5. *Sampling and empirical average*

We fit A (or equivalently θ) by minimizing a loss function defined as the opposite of the empirical average of the cumulative reward on N independent samples of initial states s_0^i and randomness $(U_t^i)_{t\le T}$ and $(V_t^i)_{t\le T}$:

$$\min_\theta -\sum_{i=1}^{N} \mathrm{CR}_T(F(\cdot,\theta), s_0^i, (V_t^i)_{0\le t\le T}, (U_t^i)_{0\le t\le T-1}). \quad (6)$$

Note that the sampling distribution of s_0^i does not matter too much. What is important is that the sample covers the states that we are interested in.

In what follows, unless stated otherwise, we minimize the above loss function with $T = 50$ taking $N = 10$ million samples with the Adam algorithm, see Kingma and Ba (2014), with 1024 mini-batch sizes and the default parameters of the Pytorch library: lr = 0.001 and betas = (0.9, 0.999), see Paszke *et al.* (2017). We perform 50 epochs on the entire sample size. At each epoch, we reduce the learning rate by 10% to improve convergence. See Appendix 1 for the pseudo code.

The complexity of the model is

$$\#\{\text{model parameters}\} \times T \times N \times \text{epochs}.$$

On a a server using 16 *2.9 GHz Xeon Platinum 8268* CPUs it takes around 10 hours to train the model (this time is weakly dependent on the environment complexity). Note however that for the simplest environments, we get a very good solution after the first epoch.

2.6. *Making environment parameters variable*

To make it usable in practice in a quantitative strategy, we need to be able to apply this to potentially thousands of stocks with different environment parameters (costs, spread, max weights,...). It would not be efficient to fit a model for each set of parameters. One easy way to solve this issue is to consider these environment parameters as static (that is non time varying) parts of the state. We therefore only need to train one model which takes as input the state and the environment parameters to simultaneously find the optimal action for a continuum of models.

More formally denote ζ an environment parameter:

$$s_{t+1} = \Phi(s_t, a_t, U_{t+1}; \zeta) \quad \text{and} \quad r_t = \Psi(s_t, a_t, V_t; \zeta).$$

Then defining the new state as $s'_t = (s_t, \zeta)$, we are in the same context as before without environment parameters:

$$s'_{t+1} = \Phi'(s'_t, a_t, U_{t+1}) \quad \text{and} \quad r_t = \Psi'(s'_t, a_t, V_t).$$

We can thus find the optimal action as a function of s_t and ζ.

As for initial states, the environment parameter sampling does not matter too much and must simply cover the range of environments that one is interested in.

3. Conception of optimal trading strategies

3.1. *Optimal investment and reinforcement learning*

As discussed in the introduction, when choosing his weights the investor must not only consider his immediate reward but also how he will be positioned in the future. As stated in Gârleanu and Pedersen (2013): ‘A good hockey player plays where the puck is. A great hockey player plays where the puck is going to be’.

Three papers which effectively tackle this issue are: Gârleanu and Pedersen (2013), where the authors find a simple intuitive solution in the simple case where the transaction costs are quadratic and related to the covariance matrix in a strong way. Boyd *et al.* (2017), where the authors use convex optimization in a multi-period setting to find a heuristic solution by assuming that the future alpha trajectory is deterministic. As we will see, this assumption yields suboptimal strategies when the alpha is stochastic. De Lataillade *et al.* (2012), where the authors use optimal control theory to find the optimal weight trajectory when costs are linear.

Another branch of application of deep reinforcement learning in finance is illustrated in Cong *et al.* (2020). Instead of splitting the model estimation and the conception of an optimal strategy in this model in two steps, it consists in giving as input to a large carefully engineered neural network the raw features and expect as output optimal weights. See Kolm and Ritter (2020) for a recent review of the applications on reinforcement learning in quantitative finance.

In this article, we follow the more classical road which consists in working in two steps:

- First modeling (and estimating) the expected return (for example using supervised machine learning as in Gu *et al.* (2020)), risk and costs related to our assets.
- Then find the optimal strategy under in this model.

Although the two approaches have merits, some advantages of splitting modelization and optimization is that it makes it easier to: decompose our performance, test our assumptions, have margin of errors on our alpha estimations and make active decisions. Moreover, in this setting, the strategy conception step is not anymore a data estimation problem (since we can always simulate more data from the model) but an optimization problem whose tentative solutions can be precisely evaluated.

We are thus in the footsteps of Chaouki *et al.* (2020). However we want to work with more complex alpha models. In such models, we did not manage to make our action functions converge with the DDPG algorithm. In particular, we want to consider models where our alpha process has (at least) two time scales: a slow and a fast. To illustrate these alphas, assume that the slow alpha has a half-life of one year (think Value, Momentum, Quality, see Asness *et al.* (2013) and Asness *et al.* (2019)) and the fast alpha has a half-life of one month (think short-term price reversal, analyst revisions, sector momentum, see Bremer and Sweeney (1991) and Jaisson *et al.* (2021)).

The kind of non-trivial properties that we hope to find is that for reasonable model parameters, the fast alpha times the execution of the slow alpha. Indeed, assume that we expect a stock to perform well in the next year but not in the next month. If we do not have the stock in our portfolio, we will wait a month before buying it. If we have it in our portfolio, it might not be worth to sell it and to buy it again in a month because of transaction costs.

3.2. Reward and objective

As in Chaouki *et al.* (2020), we will consider environments where the reward writes as a combination of a return term equal to the return times the position, a risk term and a cost term:

$$r_t = w_t R_{t+1} - \text{Risk}(w_t) - \text{Cost}(w_t, lw_t) \tag{7}$$

where R_{t+1} is the asset return over the next period, w is the weight to be chosen and lw is the last weight before the choice of the new weight. Assuming that the return writes as:

$$R_{t+1} = \alpha_t + \sigma N^R_{t+1}$$

where the expected return α_t is observable at time t, σ is the stock volatility and N^R_{t+1} are independent standard normal variables, an equivalent problem is to maximize the expectation of the cumulated expected reward defined as:

$$r_t = w_t \alpha_t - \text{Risk}(w_t) - \text{Cost}(w_t, lw_t). \tag{8}$$

In Appendix 2, we study the impact of considering the 'noised' reward on the convergence speed but in what follows, we consider this expected reward.

In this case, the state is the alpha and the last weight:

$$s_t = (\alpha_t, lw_t)$$

and the action is the next weight w_t.

In what follows, the action has a trivial effect on the next state: The last dimension of the next state is equal to the action. However, we can imagine trading environments where the action has a more complex effect on the alphas at different scales which is equivalent to the problem introduced in Curato *et al.* (2017) where market impact is transient and the kernel is approximated by a combination of exponential functions.

3.3. Trading environments description

In this paragraph, we describe the different trading environments that we will test. We give examples of alpha processes, risk functions and cost functions.

Note that all the combinations of models defined below enter the differentiable reinforcement learning framework.

3.3.1. Alpha modeling. Let us begin by defining the two alpha dynamics that we will consider: mono-scale and multi-scale.

Mono-scale alpha

In this model, as in Chaouki *et al.* (2020), the alpha of the asset is an observable $AR(1)$ process (with $\rho_\alpha = 0.9$ and $\eta_\alpha = 1$ for example):

$$\alpha_t = \rho_\alpha \alpha_{t-1} + \eta_\alpha N^\alpha_t \tag{9}$$

where N^α_t are independent standard normal variables.

The parameter η_α should be seen as a alpha scaling parameter. Indeed, the distribution of the α_t is a Gaussian of mean zero and standard deviation $\eta_\alpha/\sqrt{1-\rho^2_\alpha}$.

Two-scale alpha In this model, the alpha of the asset is a combination of two (independent) observable $AR(1)$ processes, a fast (F) and a slow (S) (with $\rho_{\alpha^S} = 0.9$, $\rho_{\alpha^F} = 0$, $\eta_{\alpha^S} = 1$ and $\eta_{\alpha^F} = 3$ for example):

$$\begin{aligned}\alpha^S_t &= \rho_{\alpha^S}\alpha^S_{t-1} + \eta_{\alpha^S} N^{\alpha^S}_t \\ \alpha^F_t &= \rho_{\alpha^F}\alpha^F_{t-1} + \eta_{\alpha^F} N^{\alpha^F}_t \\ \alpha_t &= \alpha^S_t + \alpha^F_t\end{aligned} \tag{10}$$

where $N^{\alpha^S}_t$ and $N^{\alpha^F}_t$ are independent standard normal variables.

This model will enable us to take into account the interaction between fast and slow return predictors and the last weight in the optimal action.

3.3.2. Risk modeling. Let us now state the two risk terms that we will consider: L_2 and max weight

L_2 risk This risk term is equal to a risk aversion times the square of the position:

$$\text{Risk}(w_t) = \frac{\lambda}{2} w^2_t. \tag{11}$$

Note that this risk term corresponds to penalizing the variance of a portfolio holding w_t in the stock.

Max weight as a risk As explained in Chaouki *et al.* (2020), we can impose a max weight constraint as a penalization of weights above a threshold M (or below $-M$). Here we choose a *ReLU* penalization:

$$\text{Risk}(w_t) = K(|w_t| - M)_+ \tag{12}$$

where K is large so that the optimal weight is never above M.

3.3.3. Cost modeling. Let us now state the two cost terms that we will consider: L_2 and L_1.

L_2 cost As in Gârleanu and Pedersen (2013), the cost term can be modeled as the square of the turnover. This enables to have closed form solutions but is not very realistic:

$$\text{Cost}(w_t, lw_t) = C \times |w_t - lw_t|^2. \tag{13}$$

L_1 cost To take into account spread costs, we can model the cost as proportional to the turnover:

$$\text{Cost}(w_t, lw_t) = S \times |w_t - lw_t|. \tag{14}$$

Note that this function is not strictly speaking differentiable. However, it is differentiable almost everywhere. We can thus apply SGD like methods to minimize the corresponding loss (in the same way that we can apply SGD to minimize the parameters of neural networks with *ReLU* activation functions).

4. Numerical results

In this section, we fit and describe the optimal action function in five trading environments:

- An environment with a mono-scale alpha, a L_1 cost and a L_2 risk to show that we recover the result of Chaouki *et al.* (2020) and study the behavior of our algorithm.
- An environment with a mono-scale alpha, a L_1 cost and a max weight risk to show that we recover the result of Chaouki *et al.* (2020) in a more numerically difficult case where the optimal action is not a continuous function of alpha.
- An environment with a mono-scale alpha with a variable alpha scaling, a L_1 cost and a L_2 risk. With this environment we want to prove that our algorithm can find the optimal strategy for a continuum of environment parameters with only one fit and study the impact of the alpha variability on the optimal strategy.
- An environment with a multi-scale alpha, a L_1 cost and a L_2 risk to study the impact of the fast alpha on the optimal strategy.
- An environment with a multi-scale alpha, a L_1 cost with a variable spread and a L_2 risk to study the impact of the spread on the effect of the fast alpha on the optimal strategy.

4.1. Mono-scale alpha

To study the convergence of the algorithm towards the optimal action we begin by a simple environment where:

- The alpha is mono-scale with persistence parameter $\rho_\alpha = 0.9$ and scaling $\eta_\alpha = 1$.
- There is a L_1 cost term with spread $S = 4$.
- There is a L_2 risk term with risk aversion $\lambda = 1$.

There are different behaviors depending on the model parameters: If the spread is too high compared to the amplitude of the expected return, then the optimal strategy is to (almost) never trade. If the spread is too low, the optimal strategy is to (almost) always trade. With the parameters chosen above, we are between the two regimes: we trade about once every 5 periods (see figure 3).

4.1.1. Properties of the optimal action.

The optimal weight is a function of the last weight and the alpha:

$$a_t = A(\alpha_t, lw_t).$$

We represent it by plotting for a few last weight values the optimal weight as a function of alpha in figure 1.

Note that in this model, if there is no spread costs, the optimal weight is the one which maximizes the current reward r_t which is $w_t^{\mathrm{tgt}} = \alpha_t/\lambda$. In the current case where $\lambda = 1$, α can thus be seen as a target weight. We find as in De Lataillade *et al.* (2012) that the optimal strategy has the same functional form as in the single period case, see Dybvig and Pezzo (2020), and consists in not trading if the alpha is within a threshold around the last weight and to get to this threshold if the alpha is outside of it.

These thresholds are analyzed theoretically in De Lataillade *et al.* (2012). We plot them in figure 2. The upper (resp. lower) bound is defined as the smallest (resp. largest) alpha such that the optimal weight is strictly above (resp. below) the last weight to which we subtract the last weight. We find that except for a zero last weight, this threshold is asymmetric: for positive (resp. negative) last weight, the upper bound in larger (resp. smaller) than minus the lower bound. For high (resp. low) last weights, the lower (resp.upper) bound tends to zero. This means that for a high (resp. low) last weight, the optimal strategy is to sell (resp. buy) as soon as the alpha/target weight is below (resp. above) the last weight.

We plot in figure 3 a sample trajectory of the alpha and of the associated optimal weight. We see that when the alpha gets too far from the weight, trading happens.

4.1.2. Convergence speed.

To showcase the stability of the convergence of the algorithm we plot in figure 4 the evolution of the average reward (normalized to one for the optimal strategy) through learning paths over the first epoch of the optimization.

We find that the randomness of the starting point and of the sampling does not affect the obtained strategy as it is sometimes the case when fitting deep neural networks. We thus do not need to train multiple agents and take the best.

Note also that after 1000 1024 minibatches, corresponding to one epoch over 1 million samples, the validation cumulated reward is above 99% of the optimal cumulated reward for 4 of the 5 training paths. Therefore, taking 50 epochs on 10 million samples is very conservative.

4.1.3. Impact of the horizon T.

In this paragraph, we study the impact of the horizon T on the obtained strategy. Intuitively, this horizon should be longer than the ‘memory’ of the model. For example, here the autocorrelation of the alpha process behaves as $(\rho_\alpha)^t$ and thus, the half-life of the model is $-\log(2)/\log(\rho_\alpha) \sim 6.5$. Hence, one should take a horizon of at least 10. Plotting in figure 5 the long-term reward as a function of the horizon, this is roughly what we find: The reward (and the associated strategy) converges around $T = 5$. Recall that to be conservative and allow for longer memory models, we take $T = 50$.

4.2. Max weight risk

An environment which is slightly harder to solve numerically, because of the discontinuity of the optimal action as a function of alpha, is the following:

- As before, the alpha is mono-scale with persistence parameter $\rho_\alpha = 0.9$ and scaling $\eta_\alpha = 1$.
- There is a L_1 cost term with spread $S = 4$.
- There is a max weight risk term with max weight $M = 3$ and penalization $K = 10$ (but no L_2 risk aversion).

We plot in figure 6 the optimal weight as a function of alpha for a few last weights as above. We find as in Chaouki *et al.* (2020) that the optimal solution is to get to the max weight (resp. minus the max weight) if the alpha is greater (resp. lower) than a limit and not to trade between. The

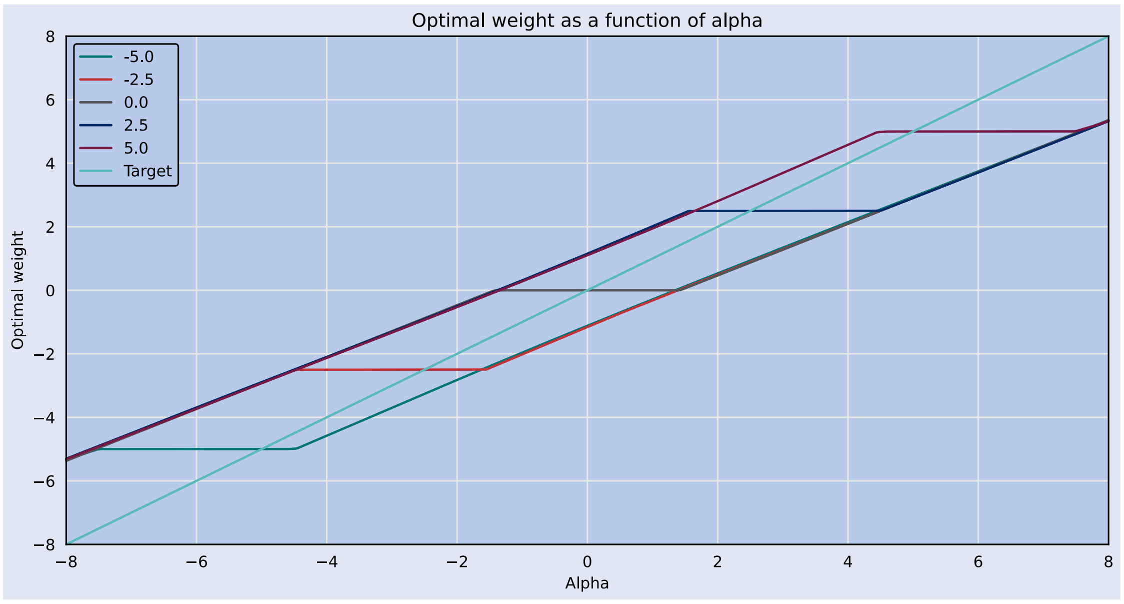

Figure 1. Optimal weight as a function of the current alpha for different last weights and the target weight which in this case is the alpha.

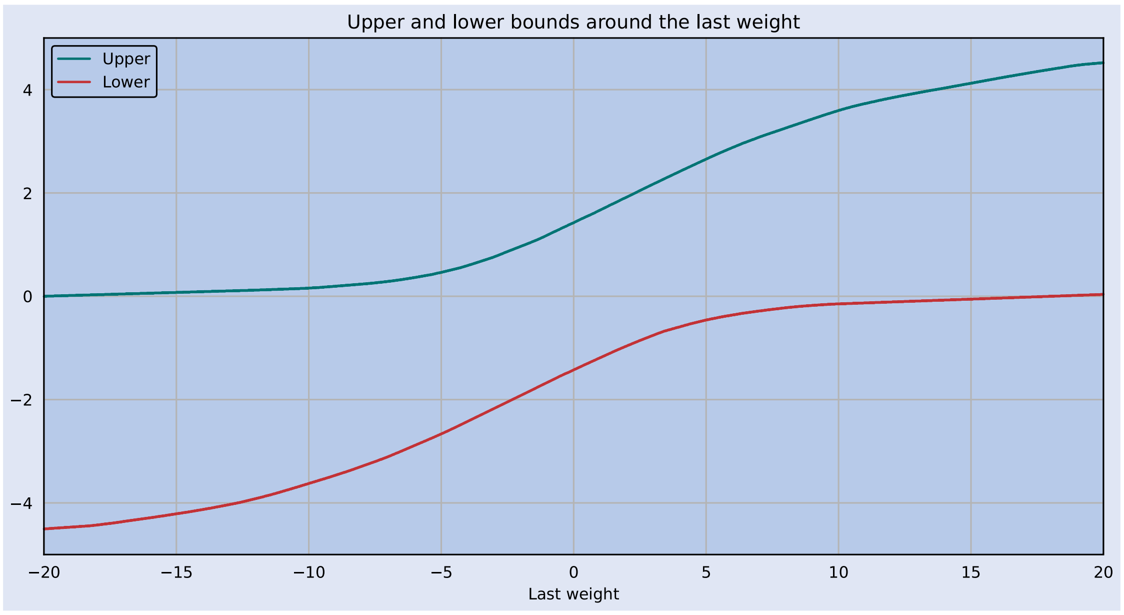

Figure 2. Upper and lower trading bounds as a function of the last weight.

optimal action as a function of alpha is thus discontinuous. Although the fit is less good than before (because a neural network needs large parameters to get large derivatives) it still captures the global shape of the optimal strategy. Note that if we increase the number of epochs, we get closer and closer to the perfect discontinuous real solution.

4.3. *Alpha scaling impact*

To show the ability of our model to treat environment parameters as static state dimensions, we train a meta-model which finds the optimal action as a function of the last weight, the alpha signal and the alpha scaling:

$$a_t = A(\alpha_t, lw_t, \eta_\alpha).$$

To do this, we take the same model as in the first environment except that we sample the alpha scaling uniformly over $[0, 4]$ and treat is as a static (that is non time varying) state dimension.

In figure 7 we plot the optimal action function as a function of the alpha for a last weight equals to zero and for different values of alpha scaling.

Interestingly, we see that the optimal strategy strongly depends on the alpha variability. It makes sense that the non trading region increases with the alpha scaling. Indeed, if the alpha variability is large, it is more likely that the alpha will change sign shortly and one will need to revert his trades. The heuristic solution of Boyd *et al.* (2017) which assumes that the future alpha trajectory is deterministic equal to its expectation corresponds to a alpha variability $\eta_\alpha = 0$ (an $AR(1)$ process with no noise is equal to its expectation). In this framework,

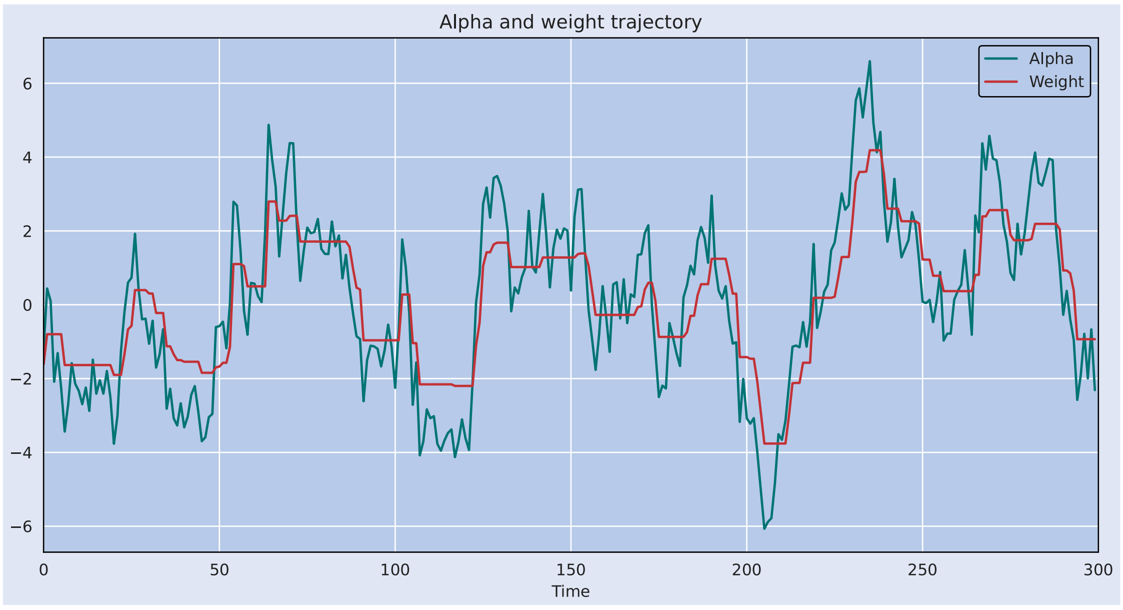

Figure 3. Sample of the trajectory of the alpha signal and optimal weight.

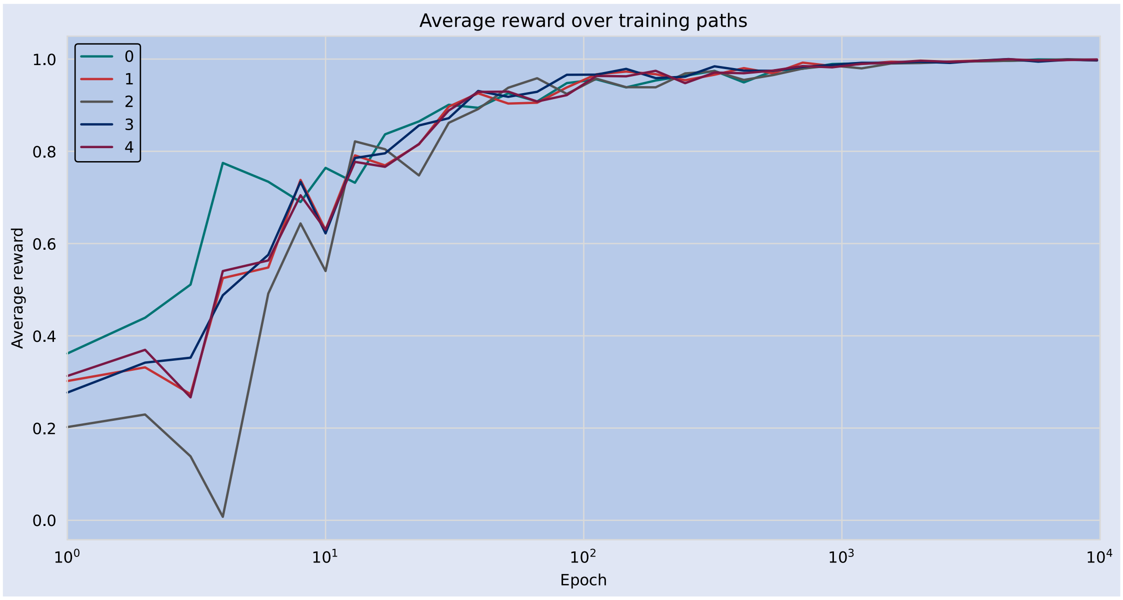

Figure 4. Evolution of the validated (on 1 million samples) reward over 5 training paths. The x axis corresponds to the number of mini-batches (each having 1024 samples).

the obtained strategy is thus the curve 0 in figure 7. Since in practise, the future alpha trajectory is uncertain ($\eta_\alpha > 0$), we see that this heuristic can be significantly suboptimal. In particular, the no trading regions are smaller than they should be.

4.4. Two-scale alpha

In this paragraph, we introduce the first multi-scale alpha environment. To our knowledge, outside of the case of L_2 transaction costs, the study of optimal strategies for a multi-scale alpha signals has never been done.

- The alpha is two-scale with persistence parameters $\rho_{\alpha^S} = 0.9$ and $\rho_{\alpha^F} = 0$ and volatilities $\eta_{\alpha^S} = 1$ and $\eta_{\alpha^F} = 3$.
- There is a L_1 cost term with spread $S = 4$.
- There is a L_2 risk term with risk aversion $\lambda = 1$.

The parameters are chosen so that we are in a regime where the optimal weight has the same form as the alpha trajectory without trading at every period. The fast alpha magnitude is chosen such that, as it is the case in practice for large portfolios, it is rarely worth doing a round trip to exploit the fast alpha. There is however room to use the fast alpha to time the execution of the slow alpha.

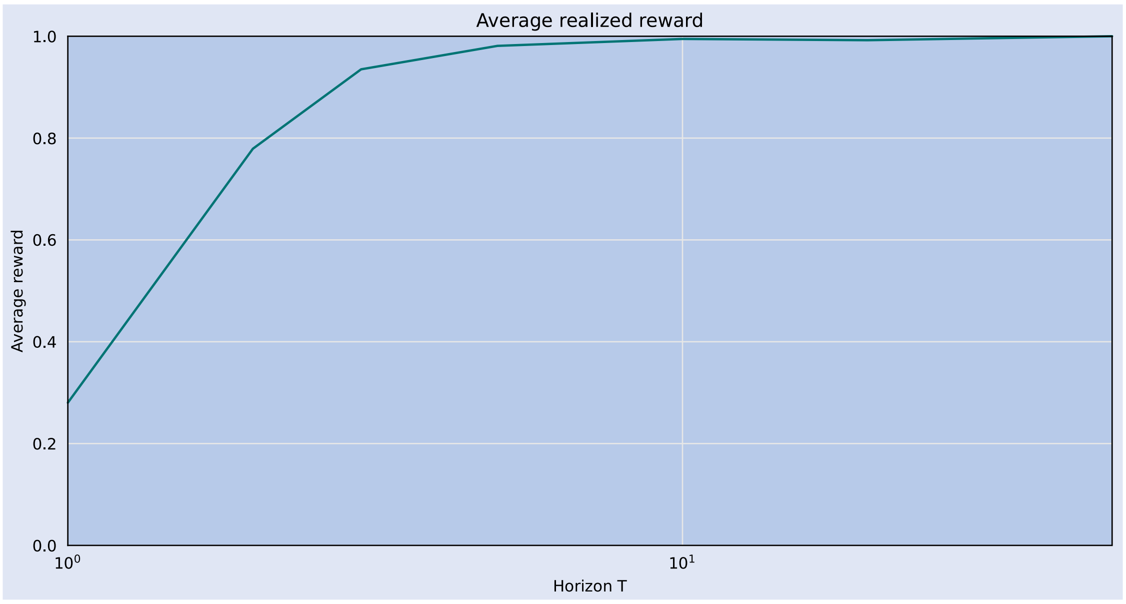

Figure 5. Average long-term reward as a function of the horizon T.

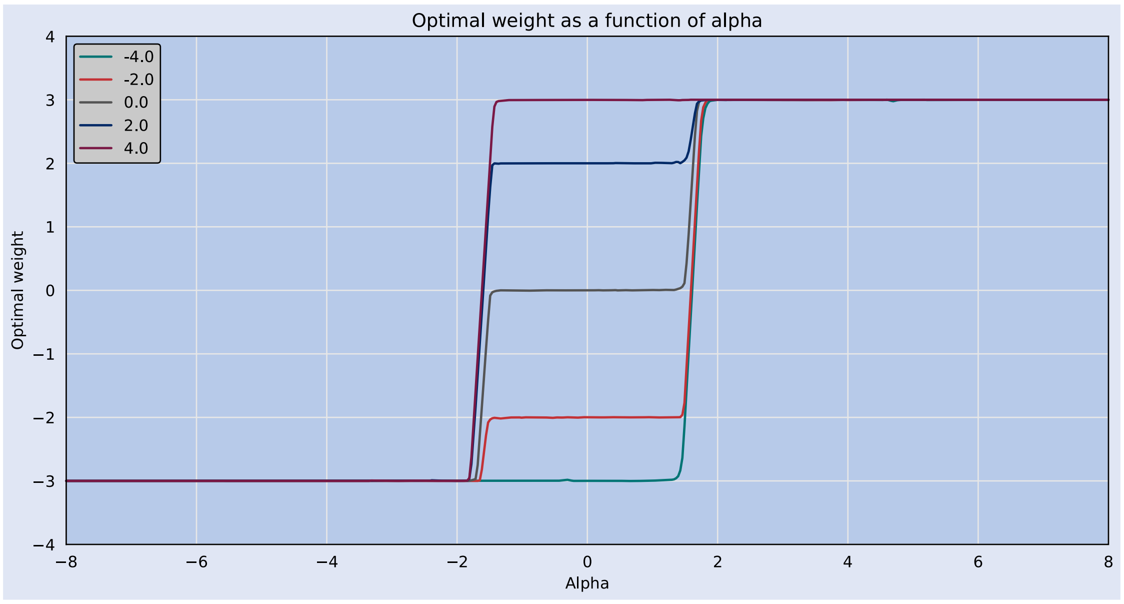

Figure 6. Optimal weight as a function of the current alpha for different last weights.

The optimal action is a function of the last weight, the slow alpha and the fast alpha:

$$a_t = A(\alpha_t^F, \alpha_t^S, lw_t).$$

To show the properties of the optimal strategy, we begin by fixing the fast alpha to zero and as before to plot in figure 8 the optimal weight as a function of the slow alpha for different starting weight.

We find the same behavior as in the first environment without the fast alpha. Let us now study the impact of the fast alpha on the optimal strategy. To do this, in figure 9, we fix the last weight equals to zero and plot as a heat-map the optimal weight as a function of the slow alpha and the short-term (or total) alpha defined as the sum of the slow and fast alphas.

We see that this heat-map is split into three areas:

- On the bottom left, the optimal action is to sell (more and more as you get lower and lefter).
- On the top right, the optimal action is to buy.
- In the middle, there is a no trading region.

If the short-term alpha is higher (resp. lower) than 2 spreads (resp. minus 2 spreads) we always buy (even if we expect to revert the trade next month). If the the short-term alpha is higher (resp. lower) than zero we never sell (resp. buy) but prefer to wait next month.

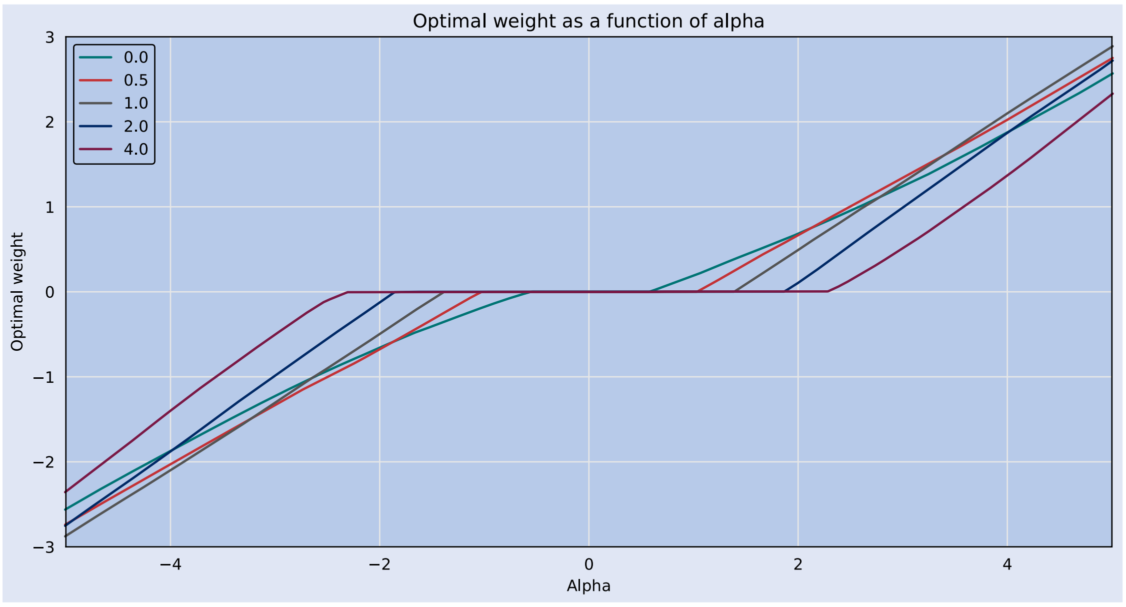

Figure 7. Optimal weight as a function of the current alpha for different alpha scaling and last weight equals zero.

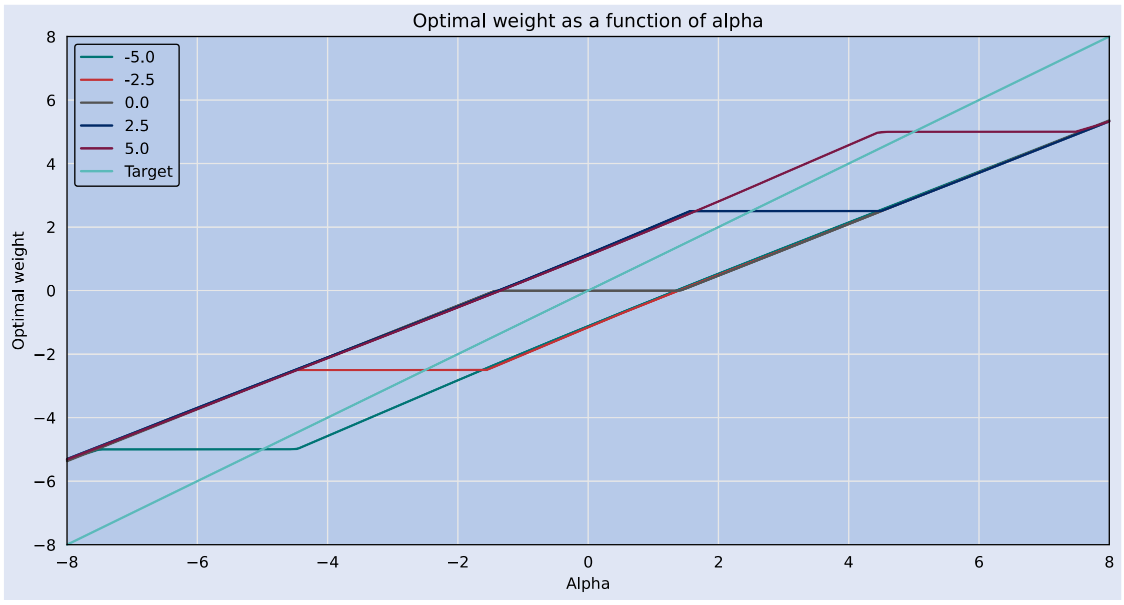

Figure 8. Optimal weight as a function of the slow alpha for different starting weight and fast alpha equals zero.

Looking at sample trajectories of the slow and fast alphas and of the optimal weight in figure 10, we see that the main long-term driver of the weight is the slow alpha and the fast alpha does not seem to matter much. However, zooming in, we see that the fast alpha is used to time the trades corresponding to the slow alpha. Indeed, looking at the two blue ovals on figure 11 below, we see that at some points the slow alpha goes up but the optimal weight does not follow before the fast alpha confirms that it is worth trading now and not to wait. This is a quite intuitive result: If you know that a stock is going to perform well next year but not next month, you wait before buying it. However, if you already have the stock then you do not necessarily sell it.

We confirm this by computing the empirical correlations between alphas, optimal weights and trades (defined as $w_t - lw_t$). We find that although the slow alpha is strongly correlated to the optimal weight (88%), trades are more correlated to the fast alpha (50%) than to the slow alpha (26%) (Table 1).

Table 1. Correlation matrix between weights, trades and slow and fast alphas.

	Slow alpha	Fast alpha	Optimal weight	Trade
Slow alpha	1.00			
Fast alpha	0.00	1.00		
Optimal weight	**0.88**	0.17	1.00	
Trade	0.26	**0.50**	0.17	1.00

Note: We see that although the weights are driven by the slow alpha, the trades are driven by the fast alpha.

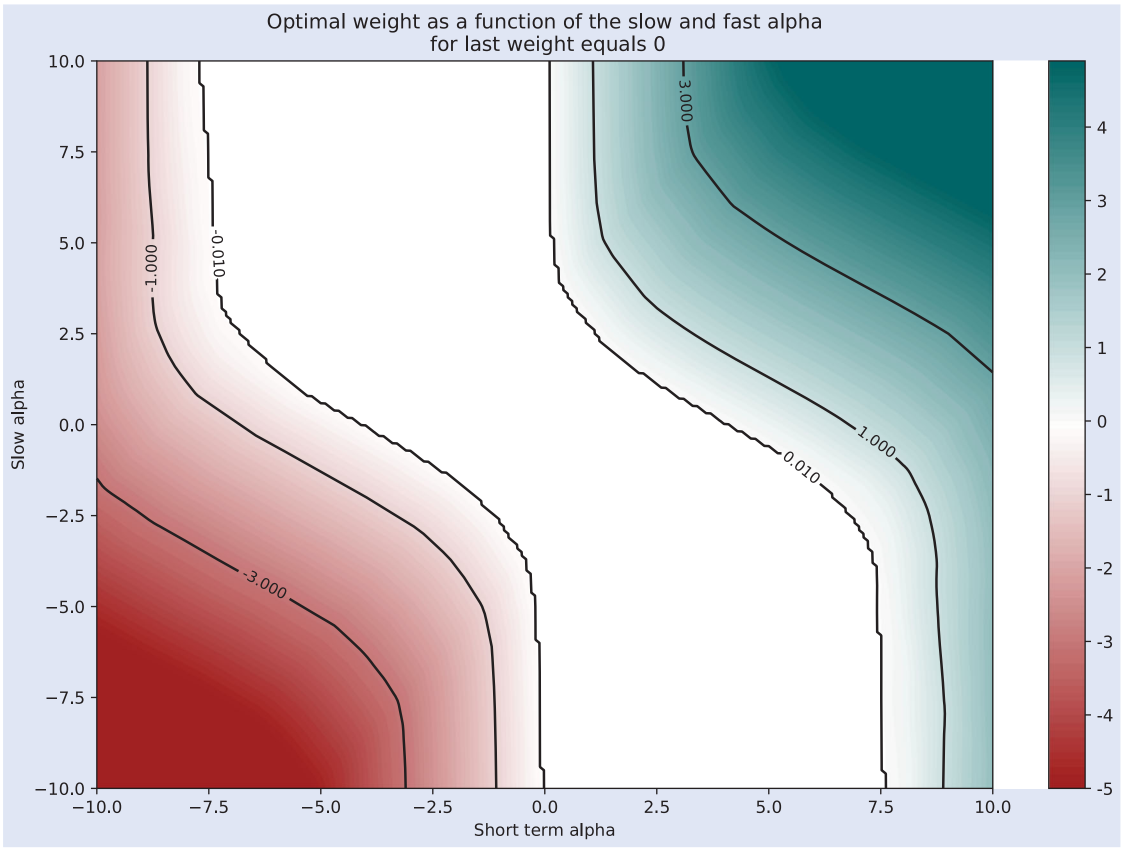

Figure 9. Heat-map of the optimal weight as a function of the slow and fast alphas for the last weight equals zero.

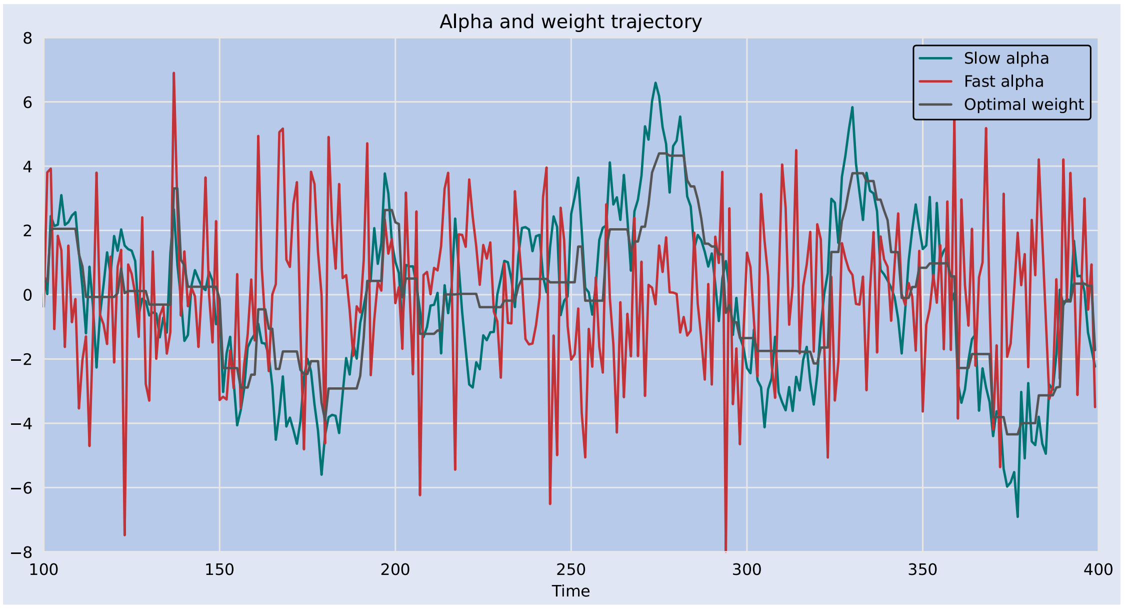

Figure 10. Sample trajectories of the slow and fast alphas and of the optimal weight.

4.5. Spread impact

To study the impact of the spread value on the effect of the fast signal in the optimal strategy, we consider the same environment as in the previous paragraph except that we treat the spread as a static state variable. We sample the spread uniformly over [0, 6].

In figure 12 we plot the no trading zones for 4 spread values.

As expected, the higher the spread the larger the no trading zone and when the spread is zero the no trading zone is empty. We also find that as the spread decreases, conditionally on the short-term alpha, the slow alpha matters less and less and only the short-term alpha is important. When the spread is zero,

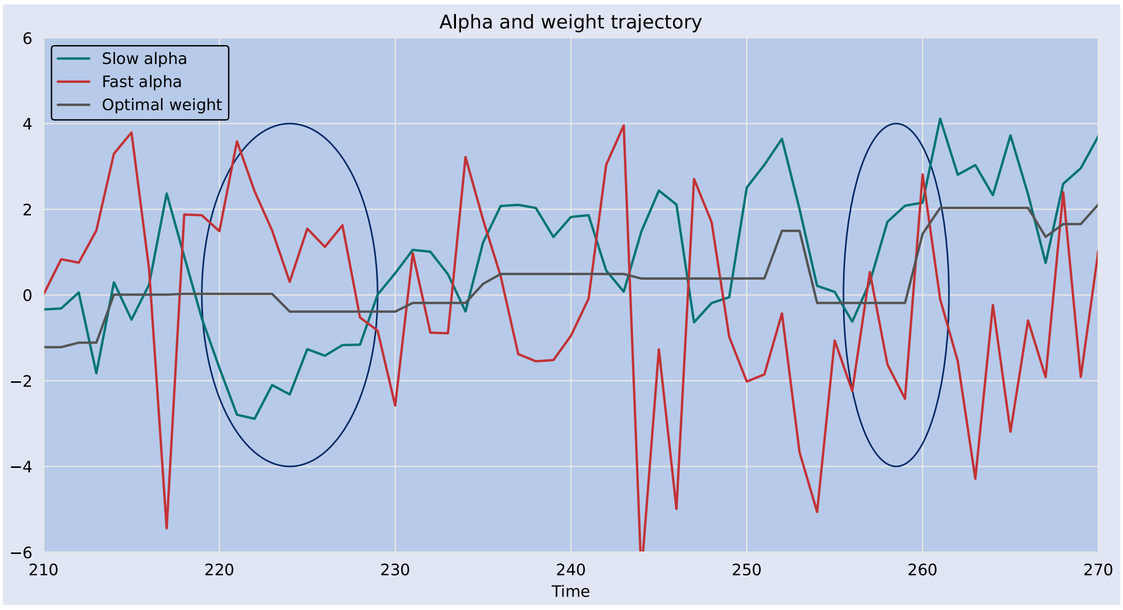

Figure 11. Zoom on sample trajectories of the slow and fast alphas and of the optimal weight.

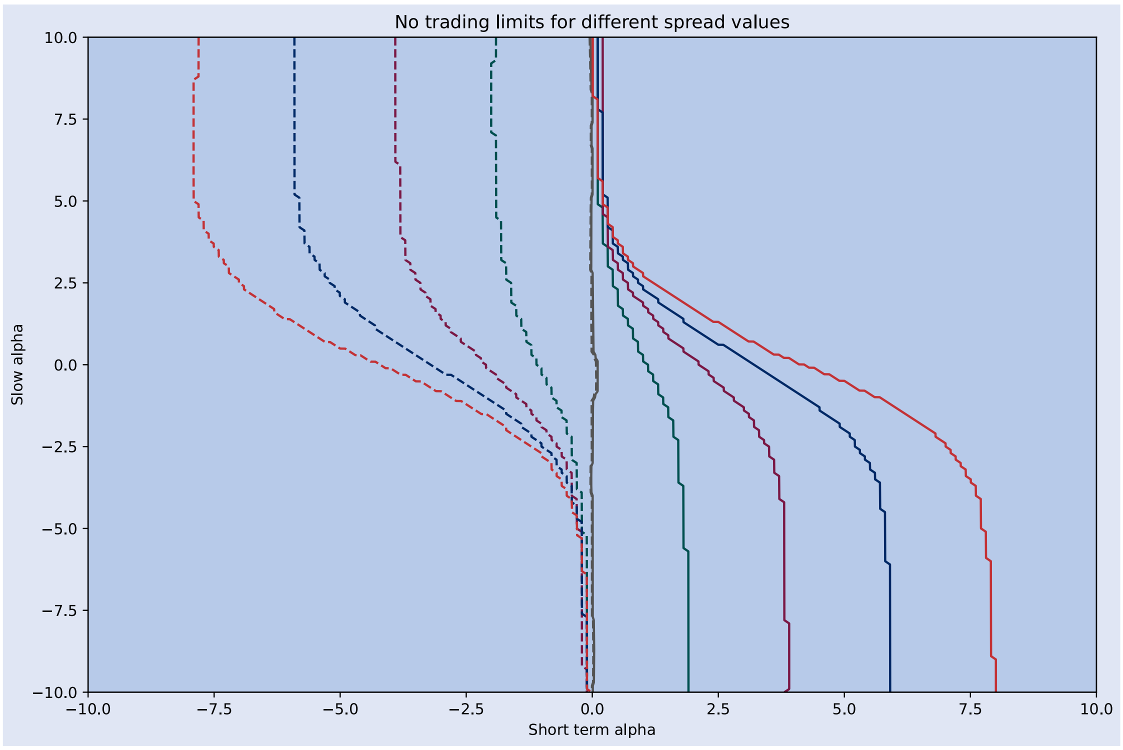

Figure 12. No trading regions as a function of the slow and short-term alpha for different spread values: 0 (gray), 1 (green), 2 (purple), 3 (blue) and 4 (red). On the left of dashed lines, we sell, on the right of full lines we buy and between the two we do not trade.

the contour lines are fully vertical and the slow alpha does not matter.

5. Conclusion

In this article, we introduced the differentiable reinforcement learning framework where the environment is not a black box but its transition and reward functions are differentiable functions of the state, the action and some randomness that can be sampled. We noted that many well studied reinforcement learning and optimal control problems fall into this framework. Using this environment knowledge, we proposed a simple algorithm to devise optimal action functions by writing them as deep neural networks. The stability and speed of this algorithm allows us to work in complex continuous multidimensional state and action environments. We can also simultaneously find a solution for many environments with one training by considering model parameters as a static state variables.

We applied this framework to devising sequential optimal trading strategies. We started by reproducing some well known strategies when the alpha process has only one time scale. We then defined and tackled environments where the expected return is a combination of a slow and fast component and studied the interactions of these two components with the current weight. We found that for reasonable parameters, the fast alpha is used to time the trades corresponding to the slow alpha. We also studied the impact of the model parameters such as the alpha scaling and the spread on the optimal strategy.

Going further, being able to have access to the optimal strategy for any model parameters allows us to extend this by combining these strategies over thousands of stocks. One way to do this is to proceed iteratively by solving the problems for all stocks independently and then to include risk and constraint terms through penalization of the alphas of stocks. Another research direction is to consider more complex models with even more alpha time scales and transient market impact to apply deep learning to get numerical solutions to the extensively studied optimal execution problems.

Acknowledgments

The author thanks Reda Messikh, Stéphane Daul and Rémy Cottet for useful discussions on the topic and feedbacks on the manuscript and two anonymous referees for their remarks and recommendations. The opinions expressed in this article are solely those of the author.

Disclosure statement

No potential conflict of interest was reported by the author(s).

References

Asness, C.S., Frazzini, A. and Pedersen, L.H., Quality minus junk. *Rev. Accounting Stud.*, 2019, **24**, 34–112.

Asness, C.S., Moskowitz, T.J. and Pedersen, L.H., Value and momentum everywhere. *J. Finance.*, 2013, **68**, 929–985.

Bertsekas, D., *Dynamic Programming and Optimal Control: Volume I*, Vol. 2012 (Athena Scientific: Belmont, MA).

Boyd, S., Busseti, E., Diamond, S., Kahn, R.N., Koh, K., Nystrup, P. and Speth, J., Multi-period trading via convex optimization. *Found. Trends Optim.*, 2017, **3**(1), 1–76.

Bremer, M. and Sweeney, R.J., The reversal of large stock-price decreases. *J. Finance.*, 1991, **46**, 747–754.

Cartea, Á., Jaimungal, S. and Sánchez-Betancourt, L., Deep reinforcement learning for algorithmic trading, 2021. Available at SSRN.

Chaouki, A., Hardiman, S., Schmidt, C., Sérié, E. and De Lataillade, J., Deep deterministic portfolio optimization. *J. Finance Data Sci.*, 2020, **6**, 16–30.

Cong, L.W., Tang, K., Wang, J. and Zhang, Y., AlphaPortfolio for investment and economically interpretable AI, 2020. SSRN, https://papers.ssrn.com/sol3/papers.cfm.

Curato, G., Gatheral, J. and Lillo, F., Optimal execution with non-linear transient market impact. *Quant. Finance*, 2017, **17**, 41–54.

Daul, S., Jaisson, T. and Nagy, A., Performance attribution of machine learning methods for stock returns prediction, 2021. Available at SSRN 4015231.

De Lataillade, J., Deremble, C., Potters, M. and Bouchaud, J.P., Optimal trading with linear costs. arXiv preprint arXiv:1203.5957, 2012.

Deng, Y., Bao, F., Kong, Y., Ren, Z. and Dai, Q., Deep direct reinforcement learning for financial signal representation and trading. *IEEE. Trans. Neural. Netw. Learn. Syst.*, 2016, **28**, 653–664.

Dybvig, P.H. and Pezzo, L., Mean-variance portfolio rebalancing with transaction costs, 2020. Available at SSRN 3373329.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G. and Pineau, J., An introduction to deep reinforcement learning. *Found. Trends Mach. Learn.*, 2018, **11**(3-4), 219–354.

Frazzini, A., Israel, R. and Moskowitz, T.J., Trading costs of asset pricing anomalies. Fama-Miller Working Paper, Chicago Booth Research Paper, 2012.

Fujimoto, S., Hoof, H. and Meger, D., Addressing function approximation error in actor-critic methods. In *Proceedings of the International Conference on Machine Learning*, pp. 1587–1596, 2018 (PMLR web site: Cambridge, MA).

Fécamp, S., Mikael, J. and Warin, X., Risk management with machine-learning-based algorithms. arXiv preprint arXiv:1902.05287, 2019.

Gârleanu, N. and Pedersen, L.H., Dynamic trading with predictable returns and transaction costs. *J. Finance*, 2013, **68**, 2309–2340.

Germain, M., Pham, H. and Warin, X., Neural networks-based algorithms for stochastic control and PDEs in finance. arXiv preprint arXiv:2101.08068, 2021.

Gu, S., Kelly, B. and Xiu, D., Empirical asset pricing via machine learning. *Rev. Financ. Stud.*, 2020, **33**(5), 2223–2273.

Gu, S., Kelly, B.T. and Xiu, D., Autoencoder asset pricing models. *J. Econometrics*, 2021, **222**(1), 429–450.

Han, J., Deep learning approximation for stochastic control problems. arXiv preprint arXiv:1611.07422, 2016.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. and Meger, D., Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32, 2018 (AAAI Press: Palo Alto, CA).

Horvath, B., Muguruza, A. and Tomas, M., Deep learning volatility: A deep neural network perspective on pricing and calibration in (rough) volatility models. *Quant. Finance*, 2021, **21**(1), 11–27.

Ioffe, S. and Szegedy, C., Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*, pp. 448–456, 2015 (PMLR: Cambridge, MA).

Jaisson, T., Nguyen, L., Messikh, R. and Susinno, G., The predictive power of analysts and their impact on prices, 2021. Available at SSRN 3973080.

Kingma, D.P. and Ba, J., Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Kolm, P.N. and Ritter, G., Modern perspectives on reinforcement learning in finance. *J. Machine Learning Finance*, 2020, **1**.

LeCun, Y., Bengio, Y. and Hinton, G., Deep learning. *Nature*, 2015, **521**, 436–444.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

Markowitz, H.M., *Portfolio Selection*, 1968 (Yale University Press: New Haven, CT).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D., Human-level control through deep reinforcement learning. *Nature*, 2015, **518**, 529–533.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A., Automatic

differentiation in PyTorch. In *Proceedings of the NIPS-W*, 2017 (NIPS: Long Beach, CA).
Schmidhuber, J., Deep learning in neural networks: An overview. *Neural Networks*, 2015, **61**, 85–117.
Sutton, R.S. and Barto, A.G., *Reinforcement Learning: An Introduction*, 2018 (MIT Press: Cambridge, MA).
Tai, L., Paolo, G. and Liu, M., Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 31–36, 2017 (IEEE).
Watkins, C.J. and Dayan, P., Q-Learning. *Mach. Learn.*, 1992, **8**, 279–292.

Appendices

Appendix 1. Pseudo code of the algorithm

In addition to the stability and precision improvement, another advantage of the deep differentiable reinforcement learning approach compared to more generic reinforcement learning ones is code simplicity. Using the Pytorch framework (or any other automatic differentiation frameworks such as Tensorflow) the code is very simple and has little parameters which need to be tweaked.

The parameters of the algorithm are:

- The horizon T of the cumulated reward.
- The number of samples N.
- The number of epochs.
- The decrease rate at each epoch of the learning rate γ.
- The Adam parameters (batch size, learning rate, learning rates, betas) that we let to their default values.

It takes as input an environment object which must have four methods:

- $e.transition(state, action, U) \mapsto state$.
- $e.reward(state, action, V) \mapsto reward$.
- $e.generate\_randomness(N, T) \mapsto (U, V)$.
- $e.initialize() \mapsto state$.

We then need to implement the cumulated reward function as

```
function $CR_T(s_0, (U_t)_{t\le T}, (V_t)_{t\le T}, A, e)$
    $R \leftarrow 0$
    $a_0 \leftarrow A(s_0)$
    for $t \leftarrow 0, T-1$ do
        $R+ = e.reward(s_t, a_t, V_t)$
        $s_{t+1} \leftarrow e.transition(s_t, a_t, U_t)$
        $a_{t+1} \leftarrow A(s_{t+1})$
    return R
```

Once we have these ingredients, we simply need follow the procedure:

(1) Sample N randomness samples U, V.
(2) Initialize function A as a dense feed forward neural network.
(3) Define the average of the cumulated reward on these samples.
(4) Minimize this average with respect to the parameters of A using Adam.

Appendix 2. Noised return reward

In this paragraph, we consider the environment of Section 4.1 except that as explained in Section 3.2 we take the noised return $R_{t+1} = \alpha_t + \sigma N^R_{t+1}$ instead of the alpha in the reward:

$$r_t = w_t R_{t+1} - \text{Risk}(w_t) - \text{Cost}(w_t, lw_t). \quad \text{(A1)}$$

Note that this is framework corresponds to the case where the investor does not know a priori that the expected return is equal to α_t and needs to learn it from samples. We take the parameter $\sigma = 50$ so that the scale of the signal to noise ratio is $\eta_\alpha / \sqrt{1-\rho_\alpha^2}/\sigma$ is around 5%.

In figure A1 we plot the average validation reward over 10 training paths of the algorithm applied to the environment of Section 4.1 versus the one defined above. We get that with noise, the algorithm takes longer to reach optimality: For example it takes 4000 steps instead of 1500 to reach 99% of the optimal reward. However, after 50 epochs over 10 million samples, the obtained strategy is indistinguishable from the one without noise.

Appendix 3. Application of the DDPG algorithm

To showcase the stability and precision of the DDRL algorithm compared to more generic approaches, in this paragraph, we apply the DDPG algorithm of Lillicrap *et al.* (2015) (with the same

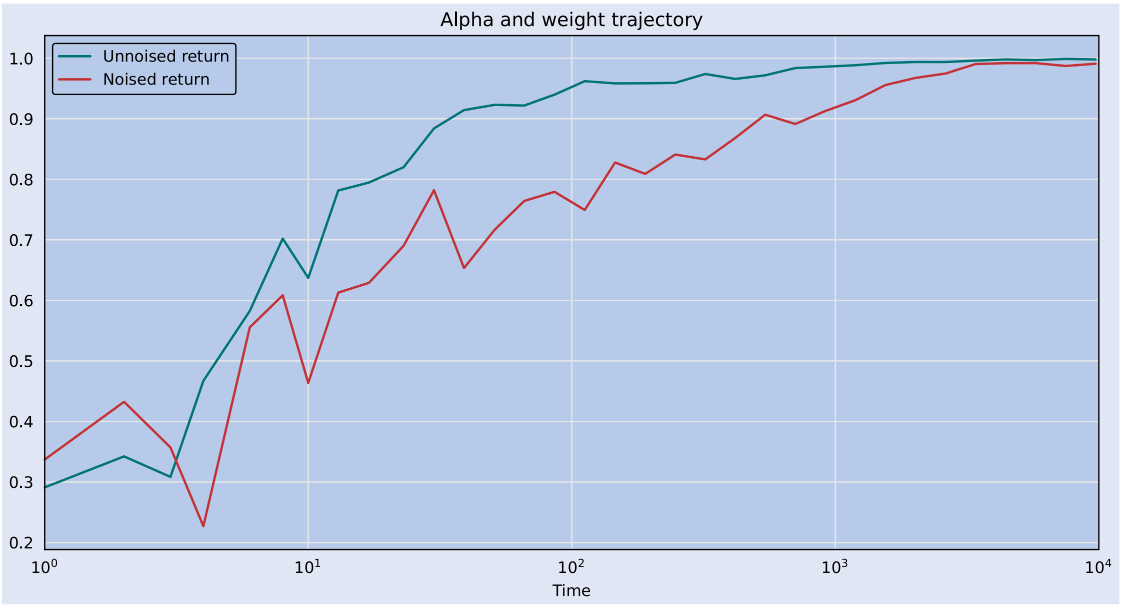

Figure A1. Average validation (over 1 million samples) reward over 10 training paths of the algorithm applied to the environment of Section 4.1 versus the same one with noised return.

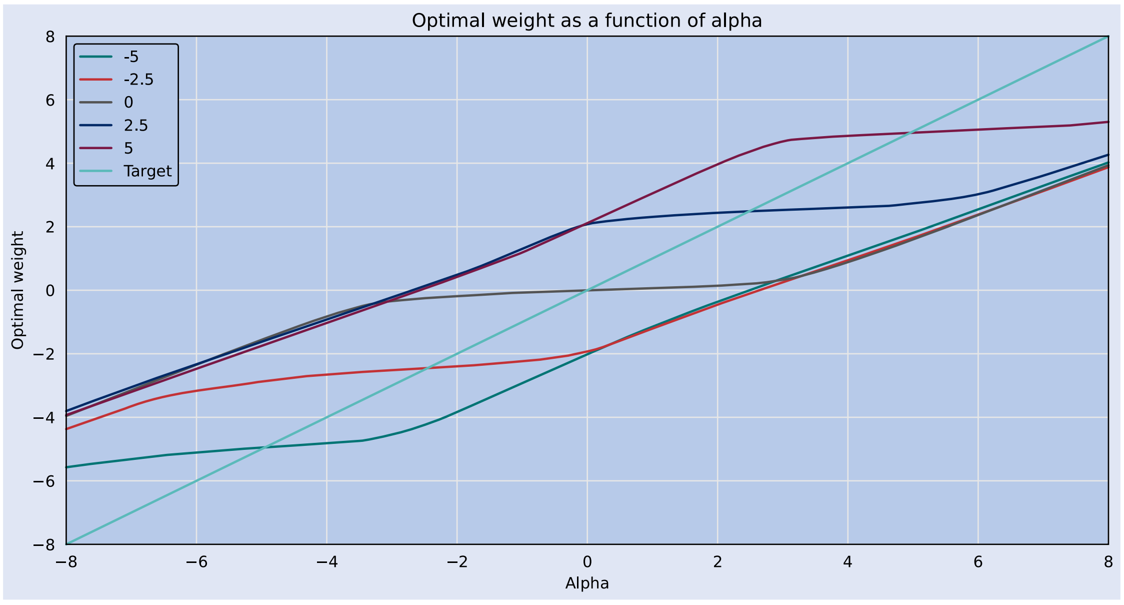

Figure A2. Weight obtained with DDPG as a function of the slow alpha for different starting weight and fast alpha equals zero.

parameters) to the two-scale alpha environment of Section 4.4. We find that through the learning path, the validation reward begins as expected by increasing and after about 15000 mini batches starts to decrease going further from the optimal strategy.

In figure A2, we plot the action function as a function of the slow alpha for different starting weight and fast alpha equals zero for the best agent of the learning path (in terms of validation reward over 1 million samples). This solution is unstable depending on the training path and lacks the symmetries of Section 4.4. It is therefore not surprising that its validation reward is only 87% of that of Section 4.4. Tweaking the parameters and considering more complex variations of the algorithm such as twin delayed DDPG, see Fujimoto *et al.* (2018), does not significantly improve the validated reward of the obtained solution.