Langevin algorithms for Markovian Neural Networks and Deep Stochastic control

Pierre Bras, Gilles Pagès

Langevin algorithms for Markovian Neural Networks and Deep Stochastic control

Pierre Bras*† and Gilles Pagès*

Abstract

Stochastic Gradient Descent Langevin Dynamics (SGLD) algorithms, which add noise to the classic gradient descent, are known to improve the training of neural networks in some cases where the neural network is very deep. In this paper we study the possibilities of training acceleration for the numerical resolution of stochastic control problems through gradient descent, where the control is parametrized by a neural network. If the control is applied at many discretization times then solving the stochastic control problem reduces to minimizing the loss of a very deep neural network. We numerically show that Langevin algorithms improve the training on various stochastic control problems like hedging and resource management, and for different choices of gradient descent methods.

Keywords– Langevin algorithm, SGLD, Markovian neural network, Stochastic control, Deep neural network, Stochastic optimization

1 Introduction

Stochastic Optimal Control (SOC), which consists in optimizing a functional of a trajectory of a controlled Stochastic Differential Equation (SDE) has applications in a wide range of problems: management of resources, queuing systems, epidemic and population processes, pricing of financial derivatives, portfolio allocation... In comparison with classic optimal control, SOC models include a random noise with known probability distribution that affects the evolution or the observation of the system. SOC also aims at managing the risk induced by this noise.

SOC problems are usually solved using specific strategies, such as Forward-Backward SDEs (FBSDEs) [PW99], or by solving Hamilton-Jacobi-Bellman (HJB) optimality conditions [Bel57] through partial differential equations methods using appropriate numerical schemes or by stochastic dynamic programming [KD01]. Such problems can also be solved using Neural Networks calibrated by SGD techniques [GM05, HE16, WLP+19, CL21, BHLP22].

More specifically, in this article we consider the numerical resolution of a SOC problem where the control is parametrized by a neural network calibrated by gradient descent. This method implies to compute the pathwise derivatives along the trajectory of the SDE of the objective function with respect to the parameters of the neural network, as introduced in [GG05, Gil07]. Stochastic gradient descent is a very general approach that can be applied to a wide range of problems and which does not need to be specifically adapted to each problem under consideration. Moreover, SGD scales very efficiently to high-dimensional problems, in contrast with HJB-based methods, and has proved its efficiency on highly non-convex problems [DdVB15].

However, if the neural control is applied at many time steps as it is the case for the Euler-Maruyama scheme where the (discrete) control is taken as an approximation of a continuous control, then the SOC problem reads as the optimization of a very deep neural network, which is roughly as deep as the number of instants at which the control can be applied (see Figures 1 and 2). Very deep neural networks are known to be considerably more difficult to train [GB10, DPG+14] and may run into vanishing gradient problems [Hoc91, Han18]. Indeed, the deeper the neural network is, the more non-linear it is, thus increasing the number of local traps for the gradient descent such as local (but not global) and saddle points. In image analysis where very deep convolutional neural networks are commonly used, residual [HZRS16] and convolutional dense [HLVDMW17] networks were introduced to deal with this issue. These networks are based on architectures with residual connections to propagate the gradient information through the numerous successive layers.

As it comes to deep SOC, we cannot freely change the structure of the neural network since it is fixed by the equations defining the SOC problem and therefore we cannot directly use residual connections. We can only

*Sorbonne Université, Laboratoire de Probabilités, Statistique et Modélisation, UMR 8001, case 158, 4 pl. Jussieu, F-75252 Paris Cedex 5, France. E-mail: pierre.bras@sorbonne-universite.fr and gilles.pages@sorbonne-universite.fr.
†Corresponding author.

freely choose the structure of the neural network returning the control, for which a few layers is often enough (see for example [BGTW19, BHLP22]).

A way to improve the learning is to replace SGD algorithms by Stochastic Gradient Langevin Dynamics (SGLD) algorithms. Such optimizers add an exogenous white noise to the gradient descent, providing regularization and allowing to escape from traps. It has indeed been observed that adding noise improves the learning for very deep neural networks [NVL+15, Ani19, GMDB16, SLH+19]. Moreover, [Bra22] compares side-by-side Langevin with non-Langevin algorithms on networks with increasing depth and shows that for shallow neural networks, Langevin algorithms do nothing else than adding noise to the gradient descent, however the deeper the network is, the greater the gains provided by Langevin algorithms are.

In the present article we study the performances of Langevin optimizers on SOC problems where the number of discretization times where the control is applied is large enough. We use the preconditioned versions of SGD and SGLD [LCCC16] for various choices of preconditioners. We compare side-by-side Langevin and non-Langevin algorithms and we show that Langevin optimizers can significantly improve the training procedure on various problems: fishing quotas [LPP21], deep hedging [BGTW19], oil drilling and resource management [GGKL21]. We mainly consider two different approaches for numerical resolution of SOCs. In the first approach, the control is a single neural network which is applied to every time step and which may depend on the running time t (see Figure 1). This approach leads to a model with fewer trainable parameters, which is critical in some data-driven financial applications where the amount of data is limited, and which is more able to capture the specific Markovian features of the problem. In the second approach, a different neural network is used for each control time (see Figure 2). This last setting is also suitable for applying the Layer Langevin algorithm, which is a variant of the Langevin algorithm introduced in [Bra22] and which proved to be more adapted to the training of very deep neural networks than the Langevin algorithm itself.

We observe that the gains of Langevin algorithms depend on the preconditioner however. While the Adam [KB15] and the Adadelta [Zei12] algorithms can be substantially accelerated by Langevin training, the gains are more limited or sometimes null for RMSprop [TH12].

The code for the numerical experiments is available at `https://github.com/Bras-P/langevin-for-stochastic-contro` It includes in particular ready-to-use Langevin optimizers and Layer Langevin optimizers as instances of the TensorFlow `Optimizer` base class, a framework for algorithm comparison in a SOC setting with GPU support and a demonstration notebook.

Notations: For x and y two vectors we define the Schur product $x*y$ as the vector $(x_i y_i)_i$. We also sometimes use the notation $*$ for scalar-vector multiplication. For $a, b \in \mathbb{N}$ we denote $\mathcal{M}_{a,b}(\mathbb{R})$ the set of $a \times b$ matrices with real-valued coefficients. For $x \in \mathbb{R}$ we define the positive part of x denoted x_+ as $\max(x, 0)$. We consider multivariate $(\mathcal{F}_t)$-Brownian motions W and B defined on some filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\in[0,T]}), \mathbb{P})$.

2 Stochastic control through gradient descent

2.1 Stochastic optimal control

We consider the following SOC problem in continuous time:

$$\min_u J(u) := \mathbb{E}\left[\int_0^T G(t, X_t)dt + F(X_T)\right], \tag{2.1}$$

$$dX_t = b(X_t, u_t)dt + \sigma(X_t, u_t)dW_t, \ t \in [0, T] \tag{2.2}$$

where $b : \mathbb{R}^{d_1} \times \mathbb{R}^{d_3} \to \mathbb{R}^{d_1}$, $\sigma : \mathbb{R}^{d_1} \times \mathbb{R}^{d_3} \to \mathcal{M}_{d_1,d_2}(\mathbb{R})$, W is a $\mathbb{R}^{d_2}$-valued Brownian motion and u is a $\mathbb{R}^{d_3}$-valued continuous adapted process, $T > 0$, $G : [0, T] \times \mathbb{R}^{d_1} \to \mathbb{R}$ and $F : \mathbb{R}^{d_1} \to \mathbb{R}$.

We first approximate the continuous SDE $(X_t)_{t\in[0,T]}$ with its Euler-Maruyama scheme and the control u_t with a discrete-time control. For $N \in \mathbb{N}$ being the number of discretization times, we consider the regular subdivision of $[0, T]$:

$$t_k := kT/N, \ k \in \{0, \dots, N\}, \quad h := T/N \tag{2.3}$$

and we approximate the control applied at times $t_0, \dots, t_{N-1}$ as the output of a single neural network depending on t, or as the output of N neural networks, one for each discretization instant t_k:

$$u_{t_k} = \bar{u}_\theta(t_k, X_{t_k}) \quad \text{or} \quad u_{t_k} = \bar{u}_{\theta^k}(X_{t_k}) \tag{2.4}$$

where $\bar{u}_\theta$ is a neural function with finite-dimensional parameter $\theta \in \mathbb{R}^d$. Indeed, since (2.2) defines a Markovian process, we can assume that u_t depends only on t and on X_t instead of t and $(X_s)_{s\in[0,t]}$.

The SOC problem (2.1) is numerically approximated by:

$$\min_{\theta} \bar{J}(\bar{u}_\theta) := \sum_{k=0}^{N-1} (t_{k+1} - t_k) G(t_{k+1}, \bar{X}^\theta_{t_{k+1}}) + F(\bar{X}^\theta_{t_N}), \tag{2.5}$$

$$\bar{X}^\theta_{t_{k+1}} = \bar{X}^\theta_{t_k} + (t_{k+1} - t_k) b\big(\bar{X}^\theta_{t_k}, \bar{u}_{k,\theta}(\bar{X}^\theta_{t_k})\big) + \sqrt{t_{k+1} - t_k}\, \sigma\big(\bar{X}^\theta_{t_k}, \bar{u}_{k,\theta}(\bar{X}^\theta_{t_k})\big)\xi_{k+1}, \tag{2.6}$$

$$\xi_k \underset{\text{i.i.d.}}{\sim} \mathcal{N}(0, I_{d_2}) \tag{2.7}$$

where $\theta \in \mathbb{R}^d$ and $\bar{u}_{k,\theta} = \bar{u}_\theta(t_k, \cdot)$ in the first case of (2.4) and $\theta = (\theta^0, \dots, \theta^{N-1}) \in (\mathbb{R}^d)^N$ and $\bar{u}_{k,\theta} = \bar{u}_{\theta^k}$ in the second case of (2.4).

For every θ, $\nabla_\theta \bar{J}$ can be computed by automatic differentiation as the gradient w.r.t. to θ is tracked all along the trajectory through the recursive relation (2.6) [GG05, Gil07]. Then the SGD algorithm reads

$$\theta_{n+1} = \theta_n - \gamma_{n+1} \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \nabla_\theta \bar{J}(\bar{u}_{\theta_n}, (\xi_k^{i,n+1})_{1 \le k \le N}) =: \theta_n - \gamma_{n+1} g_{n+1} \tag{2.8}$$

where $(\xi_k^{i,n})_{1 \le k \le N, 1 \le i \le n_{\text{batch}}, n \in \mathbb{N}}$ is an array of i.i.d. random vectors $\mathcal{N}(0, I_{d_2})$-distributed, $(\gamma_n)_{n \in \mathbb{N}}$ is a non-increasing positive step sequence and where the dependence of $\bar{J}$ in $(\xi_k^{i,n})$ is made explicit.

If the number of Euler-Maruyama steps N is large, then the optimization problem in (2.5) consists in the training of a very deep neural network that can be difficult to train directly (see the Introduction). Both cases are illustrated in Figures 1 and 2.

2.2 Preconditioned stochastic gradient Langevin dynamics

We consider preconditioned stochastic gradient algorithms i.e. for (P_n) a preconditioner rule the update reads

$$\theta_{n+1} = \theta_n - \gamma_{n+1} P_{n+1} \cdot g_{n+1} \tag{2.9}$$

where g_{n+1} is defined in (2.8). We use the Adam [KB15], the RMSprop [TH12] and the Adadelta [Zei12] preconditioners, which are detailed in Algorithms 1, 2 and 3 respectively. For some algorithm *name*, the corresponding Langevin algorithm denoted L-*name* reads

$$\theta_{n+1} = \theta_n - \gamma_{n+1} P_{n+1} \cdot g_{n+1} + \sigma_{n+1} \sqrt{\gamma_{n+1}} \mathcal{N}(0, P_{n+1}) \tag{2.10}$$

where (σ_n) is a constant or non-decreasing sequence controlling the amount of injected noise.

Algorithm 1 Adam update

Parameters: $\beta_1, \beta_2, \lambda > 0$

$M_{n+1} = \beta_1 M_n + (1 - \beta_1) g_{n+1}$

$\mathrm{MS}_{n+1} = \beta_2 \mathrm{MS}_n + (1 - \beta_2) g_{n+1} \odot g_{n+1}$

$\widehat{M}_{n+1} = M_{n+1} / (1 - \beta_1^{n+1})$

$\widehat{\mathrm{MS}}_{n+1} = \mathrm{MS}_{n+1} / (1 - \beta_2^{n+1})$

$P_{n+1} = \mathrm{diag}\left(\mathbb{1} \oslash \left(\lambda \mathbb{1} + \sqrt{\widehat{\mathrm{MS}}_{n+1}}\right)\right)$

$\theta_{n+1} = \theta_n - \gamma_{n+1} P_{n+1} \cdot \widehat{M}_{n+1}$.

Algorithm 2 RMSprop update

Parameters: $\alpha, \lambda > 0$

$\mathrm{MS}_{n+1} = \alpha \mathrm{MS}_n + (1 - \alpha) g_{n+1} \odot g_{n+1}$

$P_{n+1} = \mathrm{diag}\left(\mathbb{1} \oslash \left(\lambda \mathbb{1} + \sqrt{\mathrm{MS}_{n+1}}\right)\right)$

$\theta_{n+1} = \theta_n - \gamma_{n+1} P_{n+1} \cdot g_{n+1}$

Algorithm 3 Adadelta update

Parameters: $\beta_1, \beta_2, \lambda > 0$

$\mathrm{MS}_{n+1} = \beta_1 \mathrm{MS}_n + (1 - \beta_1) g_{n+1} \odot g_{n+1}$

$P_{n+1} = \mathrm{diag}\left((\lambda \mathbb{1} + \widehat{\mathrm{MS}}_n) \oslash \left(\lambda \mathbb{1} + \sqrt{\widehat{\mathrm{MS}}_n}\right)\right)$

$\theta_{n+1} = \theta_n - \gamma_{n+1} P_{n+1} \cdot g_{n+1}$.

$\widehat{\mathrm{MS}}_{n+1} = \beta_2 \mathrm{MS}_n + (1 - \beta_2)(\theta_{n+1} - \theta_n) \odot (\theta_{n+1} - \theta_n)$.

The Layer Langevin algorithm, introduced in [Bra22] consists in updating with Langevin noise only some layers of the network. It relies on the heuristic that for a deep neural network, the non-linearities of the network are mostly contained in the deepest layers and adds Langevin noise to these layers only.

Choosing a preconditioner rule P called *name*, the Layer Langevin algorithm denoted LL-*name* reads

$$\theta_{n+1}^{(i)} = \theta_n^{(i)} - \gamma_{n+1} [P_{n+1} \cdot g_{n+1}]^{(i)} + \mathbb{1}_{i \in \mathcal{J}} \sigma_{n+1} \sqrt{\gamma_{n+1}} \big[\mathcal{N}(0, P_{n+1})\big]^{(i)}, \tag{2.11}$$

where $\mathcal{J}$ is a subset of weight indices. In particular, we denote LL-*name* p% the Layer Langevin *name* algorithm where the Langevin layers are chosen to be the first p% layers.

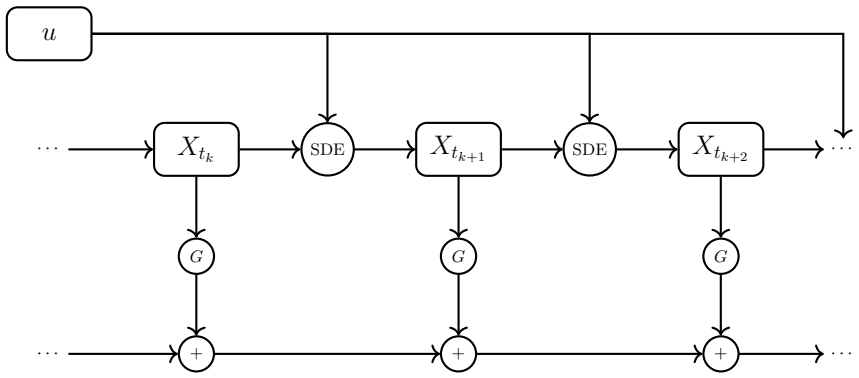

Figure 1: Depth of Markovian neural networks. The control u acts on X_{t_k}, which itself acts on $X_{t_{k+1}}$, $X_{t_{k+2}}$, ..., X_{t_N}, hence the depth of the network.

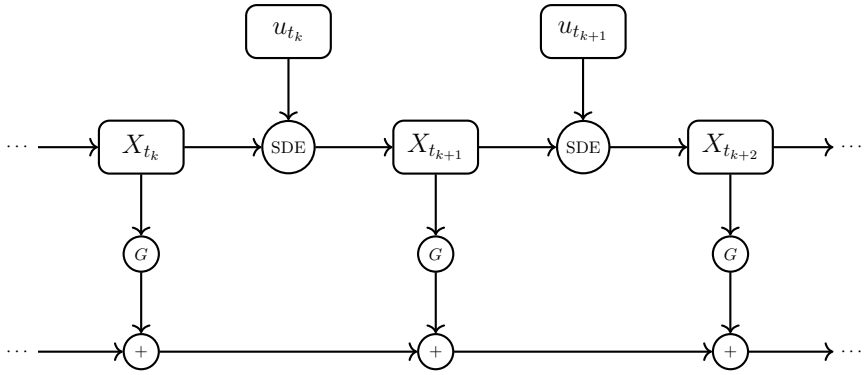

Figure 2: Markovian neural network with one control for every time step.

2.3 Experimental setting

We proceed to side-by-side comparison of Langevin algorithms (2.10) with their non-Langevin counterparts (2.9) on various SOC problems.

We consider a first case where the control is given by only one neural network depending on t and a second case where a different neural network is used for each control time (2.4). In this second case, since we can expect two consecutive control networks to have close parameters, one usual way of performing the training procedure is to first train networks for a small amount of control times, then to perform the whole training through transfer learning. We do not expect Langevin algorithms to be suitable for the fine tuning, but since the first step of the training still consists in training a deep neural network, we analyse the benefits of Langevin algorithms for this first step.

Unless stated otherwise, the batch size is set to $n_{\text{batch}} = 512$ i.e. stochastic gradient iterations are performed by averaging the gradient over 512 random trajectories. In the plot, each epoch consists in 5 batches i.e. the average loss $J(u_\theta)$ is plotted every 5 iterations of the stochastic gradient. After each epoch, $J(u_\theta)$ is estimated over 25×512 trajectories and 95% confidence intervals are indicated, although for some plots the intervals are too small to be visible. While comparing some algorithm with its Langevin or Layer Langevin counterpart, we ensure that both training procedures start with the same initial weights.

3 Fishing quotas

We consider the fishing quota problem introduced in [LPP21]. Let $X_t \in \mathbb{R}^{d_1}$ be the fish biomass for every fish species; we wish to keep it close to an ideal state $\mathcal{X}_t \in \mathbb{R}^{d_1}$. The dynamics of X are given by

$$dX_t = X_t * ((r - u_t - \kappa X_t)dt + \eta dW_t)\,, \ t \in [0, T], \tag{3.1}$$

where $r \in \mathbb{R}^{d_1}$ is the growth rate for each species, $u_t \in \mathbb{R}^{d_1}$ is the controlled fishing (with $d_3 = d_1$), $\kappa \in \mathcal{M}_{d_1,d_1}(\mathbb{R})$ is the interaction matrix between the fish species, $\eta \in \mathcal{M}_{d_1,d_2}(\mathbb{R})$, W is a $\mathbb{R}^{d_2}$-valued Brownian motion. The

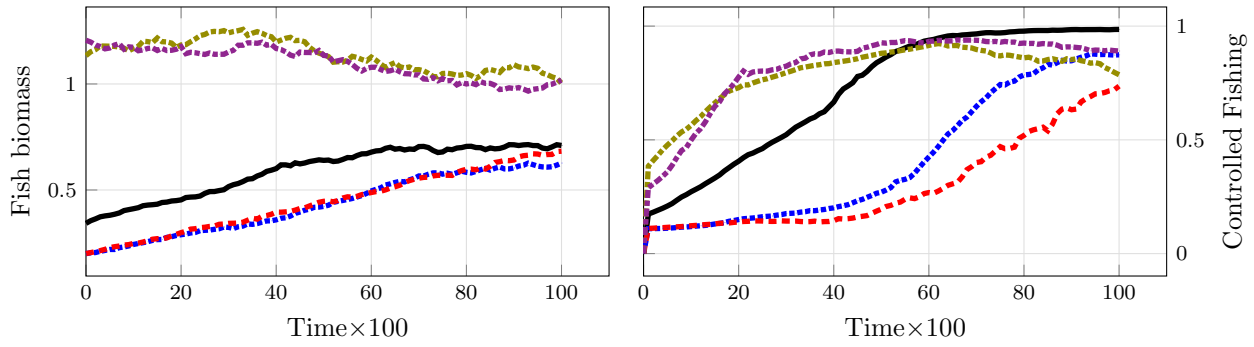

Figure 3: Example of a trajectory of $X_t \in \mathbb{R}^5$ along with the controlled fishing u_t with $N = 100$. We recall that the objective biomass is $\mathcal{X}_t \equiv \mathbf{1}$.

control u is constrained to take its values in $[u_m, u_M]^{d_1}$. The objective is

$$J(u) = \mathbb{E}\left[\int_0^T (|X_t - \mathcal{X}_t|^2 - \langle \alpha, u_t \rangle)dt + \beta[u]^{0,T}\right], \tag{3.2}$$

where $\alpha \in \mathbb{R}^{d_1}$, $\beta \in \mathbb{R}^+$, $[u]^{0,T}$ denotes the quadratic variation of u on $[0,T]$. The term $\langle \alpha, u \rangle$ penalizes small fishing quotas while the term $\beta[u]^{0,T}$ penalizes too many daily changes.

In the experiments, following [LPP21] we choose

$$d_1 = d_2 = 5,\ T = 1,\ \mathcal{X} \equiv \mathbf{1},\ r = 2 * \mathbf{1},\ \eta = 0.1 * I_{d_1},\ \alpha = 0.01 * \mathbf{1},\ \beta = 0.1,\ u_m = 0.1,\ u_M = 1. \tag{3.3}$$

and

$$\kappa = \begin{pmatrix} 1.2 & -0.1 & 0 & 0 & -0.1 \\ 0.2 & 1.2 & 0 & 0 & -0.1 \\ 0 & 0.2 & 1.2 & -0.1 & 0 \\ 0 & 0 & 0.1 & 1.2 & 0 \\ 0.1 & 0.1 & 0 & 0 & 1.2 \end{pmatrix}. \tag{3.4}$$

The initial state X_0 is randomly generated following $\mathcal{N}\big(\mathbf{1}, (1/2)I_{d_1}\big)$ clipped to $[0.2, 2]^{d_1}$. The quadratic variation $[u]^{0,T}$ is approximated in the discretized setting by

$$[u]^{0,T} \simeq \sum_{k=0}^{N-1} |u_{t_{k+1}} - u_{t_k}|^2. \tag{3.5}$$

Each control u_θ is given by a feedforward neural network with two hidden layers with 32 units each and with ReLU activation while the output layer has sigmoid activation in order to fulfil the constraint on u. An example of controlled trajectory is plotted in Figure 3.

The results are given in Figure 4 for the Adam optimizer with an increasing number of Euler-Maruyama steps and with one single control, in Figure 5 for the RMSprop and L-Adadelta optimizers and with one single control and in Figure 6 for the training with multiple neural networks.

4 Deep hedging

We consider the problem of hedging portfolio of derivatives as a SOC problem as in [BGTW19]. We aim to replicate a $\mathcal{F}_T$-measurable payoff Z defined on some portfolio $S_t \in \mathbb{R}^{d_1}$ by trading (at least some of) the assets contained in S_t at times (t_k). The control is given by $u_t \in \mathbb{R}^{d_1}$ representing the amount held for each asset. The objective is

$$J(u) = \nu\left(-Z + \sum_{k=0}^{N-1} \langle u_{t_k}, S_{t_{k+1}} - S_{t_k} \rangle - \sum_{k=0}^{N} \langle c_{tr}, S_{t_k} * |u_{t_k} - u_{t_{k-1}}| \rangle\right) \tag{4.1}$$

where $\nu : L^1(\Omega) \to \mathbb{R}$ is a convex risk measure (see [BGTW19, Definition 3.1]), $c_{tr} \in \mathbb{R}^{d_1}$ represents proportional transaction costs and we fix $u_{t_{-1}} = u_{t_N} = 0$, implying full liquidation in T. We furthermore assume that ν can be written as

$$\nu(X) = \inf_{w \in \mathbb{R}} (w + \mathbb{E}[\ell(-X - w)]) \tag{4.2}$$

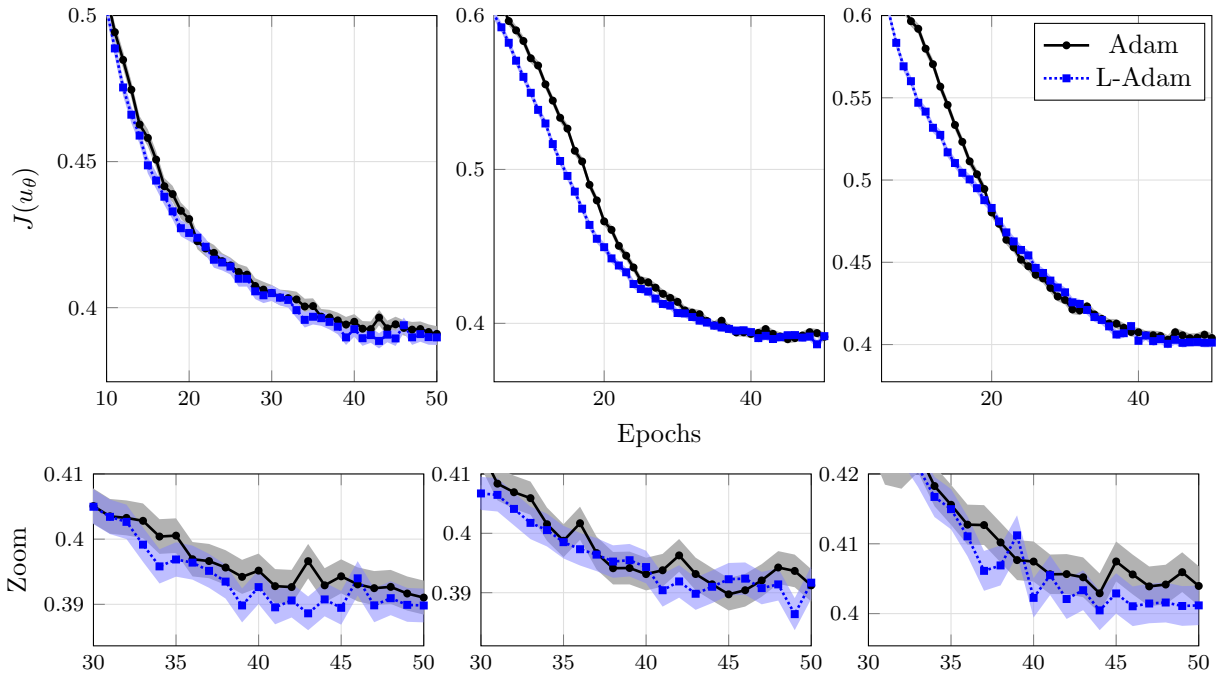

Figure 4: Comparison of Adam et L-Adam algorithms during the training for the fishing control problem with $N = 20, 50, 100$ respectively. The schedules are $\gamma_n = 2\,\mathrm{e}{-3}$ and $\sigma_n = 1\,\mathrm{e}{-3}$ ($5\,\mathrm{e}{-3}$ for $N = 100$) for epochs 0 to 40 and $\gamma_n = 2\,\mathrm{e}{-4}$ and $\sigma_n = 0$ beyond. At the end of each epoch, J is estimated over 50×512 trajectories. A zoom on the last epochs is given.

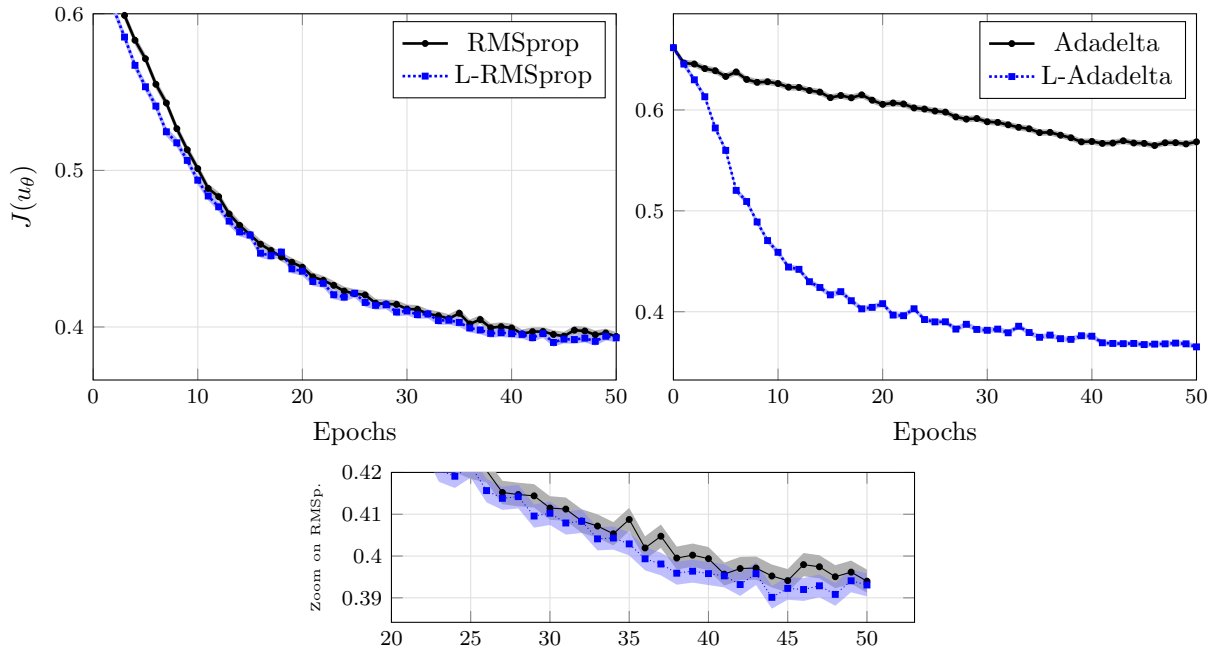

Figure 5: Comparison of Langevin algorithms with their non-Langevin counterparts during the training for the fishing control problem with $N = 50$ respectively. The schedules are $\gamma_n = 2\,\mathrm{e}{-3}$ ($5\,\mathrm{e}{-1}$) and $\sigma_n = 5\,\mathrm{e}{-3}$ ($1\,\mathrm{e}{-2}$) for RMSprop (Adadelta resp.) for epochs 0 to 40 and γ_n is divided by 10 and σ_n is set to 0 beyond. At the end of each epoch, J is estimated over 50×512 trajectories. A zoom on the last epochs for RMSprop is given.

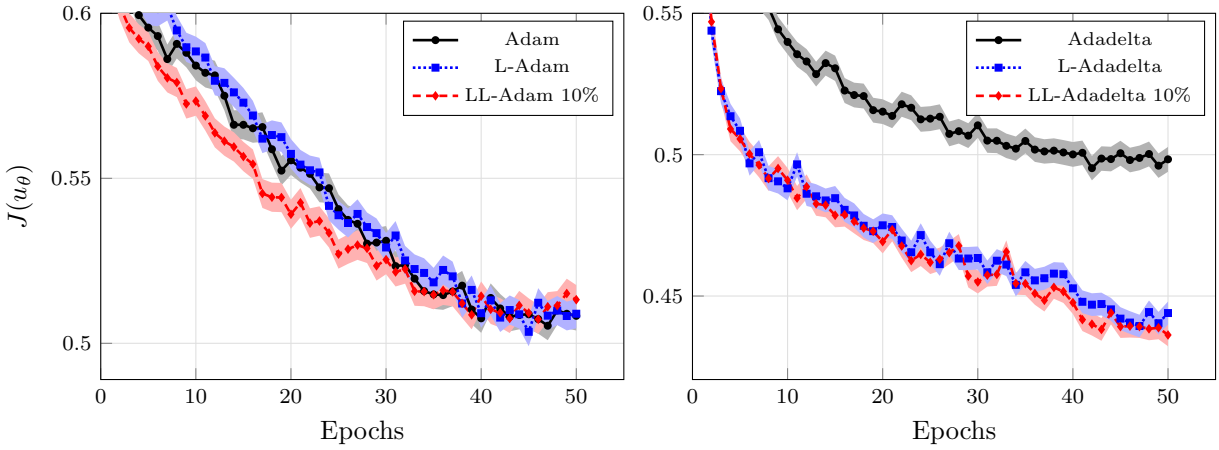

Figure 6: Training of the fishing problem with multiple controls with $N = 10$. The schedules are $\gamma_n = 2\,\mathrm{e}{-3}$ and $\sigma_n = 2\,\mathrm{e}{-3}$ for Adam and $\gamma_n = 5\,\mathrm{e}{-1}$ and $\sigma_n = 5\,\mathrm{e}{-3}$ for Adadelta, for epochs 0 to 40 γ_n is divided by 10 and σ_n is set to 0 beyond.

where the loss function $\ell : \mathbb{R} \to \mathbb{R}$ is continuous, non-decreasing and convex. This is the case in particular for the entropic risk measure where $\ell(x) = -\exp(-\lambda x)$ and the conditional value at risk measure where $\ell(x) = (1-\alpha)^{-1}\max(x, 0)$. Then (4.1) can be rewritten as

$$\inf_{u,w} J(u, w) := \mathbb{E}\left[w + \ell\left(Z - \sum_{k=0}^{N-1}\langle u_{t_k}, S_{t_{k+1}} - S_{t_k}\rangle + \sum_{k=0}^{N}\langle c_{tr}, S_{t_k} * |u_{t_k} - u_{t_{k-1}}|\rangle - w\right)\right]. \tag{4.3}$$

In the numerical experiments, we analyse the problem of hedging in a Heston model as described in [BGTW19, Section 5]. For even d_1, we consider $d_1' := d_1/2$ independent Heston models where the price and volatility processes are described by the following SDEs for $1 \le i \le d_1'$:

$$dS_t^{1,i} = \sqrt{V_t^i}\, S_t^{1,i} dB_t^i, \quad S_0^{1,i} = s_0^i, \tag{4.4}$$

$$dV_t^i = a^i(b^i - V_t^i)dt + \eta^i\sqrt{V_t^i}dW_t^i, \quad V_0^i = v_0^i, \tag{4.5}$$

where $a, b, \eta, s_0, v_i \in (\mathbb{R}^+)^{d_1'}$ and for each $1 \le i \le d_1'$, B^i and W^i are standard Brownian motions with correlation $\rho^i \in [-1, 1]$. The volatility V itself is not tradable directly but only through options on variance modelled by the following variance swap:

$$S_t^{2,i} := \mathbb{E}\left[\int_0^T V_s^i ds \,\middle|\, \mathcal{F}_t\right] = \int_0^t V_s^i ds + L^i(t, V_t^i), \tag{4.6}$$

$$L^i(t, v) := \frac{v - b^i}{a^i}\left(1 - e^{a^i(T-t)}\right) + b^i(T - t). \tag{4.7}$$

The payoff is given by

$$Z = \sum_{i=1}^{d_1'}\left(S_T^{1,i} - K^i\right)_+$$

where $K \in (\mathbb{R}^+)^{d_1'}$. We consider the convex risk measure associated to the value-at-risk i.e. associated to the loss function

$$\ell(x) = \frac{1}{1-\alpha}\max(x, 0).$$

In the experiments we choose

$$d_1' = 5,\ T = 1,\ a = \mathbf{1},\ b = 0.04 * \mathbf{1},\ \eta = 2 * \mathbf{1},\ \rho = -0.7 * \mathbf{1},\ \alpha = 0.9, \tag{4.8}$$

$$s_0 = K = \mathbf{1},\ v_0 = 0.1 * \mathbf{1},\ c_{\mathrm{tr}} = 5\,\mathrm{e}{-4} * \mathbf{1}. \tag{4.9}$$

Each control u_θ is given by a feedforward neural network with two hidden layers with 32 units each and with ReLU activation while the output layer has ReLU activation too in order to forbid short-selling. As recommended in [BGTW19], since transaction costs are involved the control u_θ at time t_k is a function of $\log(S_{t_k}^1)$, V_{t_k} and $u_{t_{k-1}}$. An example of controlled trajectory showing only one of the five Heston models is plotted in Figure 7.

The results are given in Figure 8 for the comparison of Langevin and non-Langevin algorithms with a single control and in Figure 9 for the training with multiple controls.

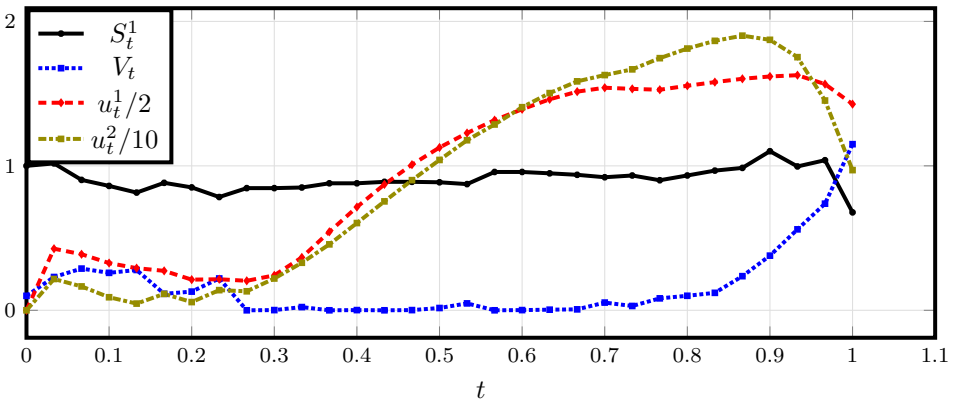

Figure 7: Example of trajectory for the deep hedging problem with $N = 30$.

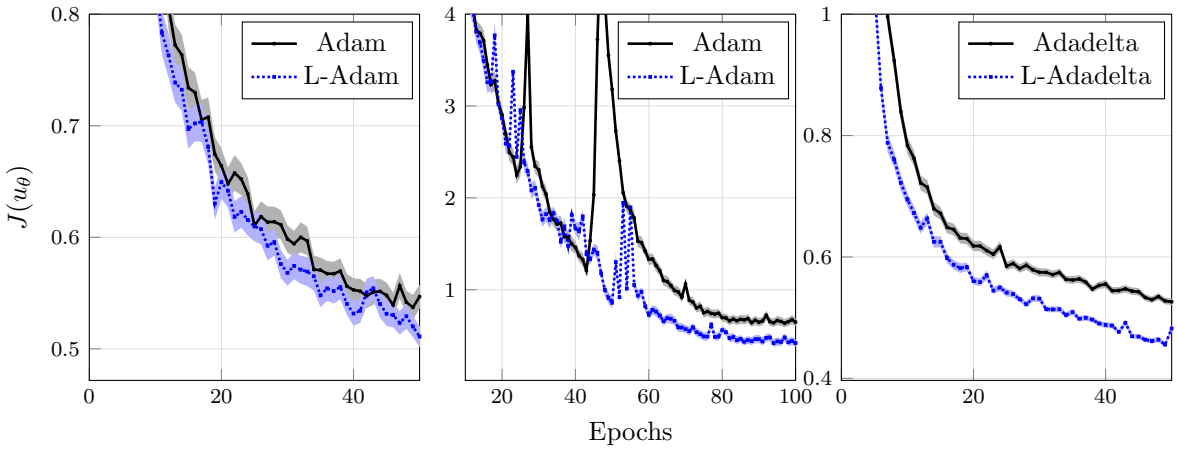

Figure 8: Comparison of algorithms during the training for the deep hedging control problem with $N = 30, 50, 50$ respectively. The schedules are $\gamma_n = 2\,\mathrm{e}{-3}$ $(5\,\mathrm{e}{-1})$ and $\sigma_n = 2\,\mathrm{e}{-3}$ $(5\,\mathrm{e}{-3})$ for Adam (resp. Adadelta) for epochs 0 to 80 and γ_n is divided by 10 and σ_n is set to 0 beyond.

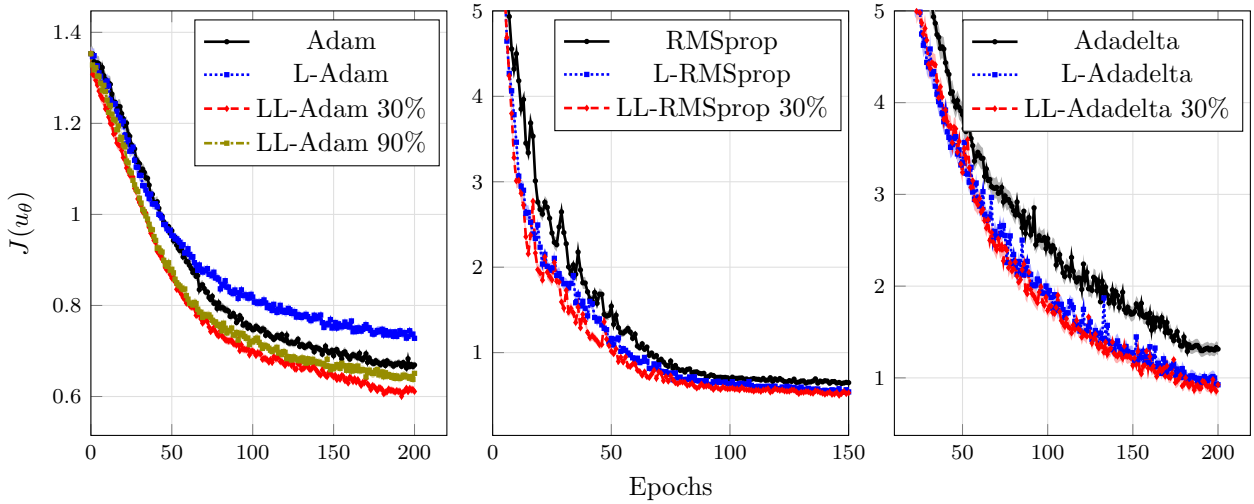

Figure 9: Training of the deep hedging problem with multiple controls with $N = 10$. The schedules are $\gamma_n = 2\,\mathrm{e}{-3}$ $(5\,\mathrm{e}{-1})$ and $\sigma_n = 2\,\mathrm{e}{-3}$ $(5\,\mathrm{e}{-3})$ for Adam and RMSprop (resp. Adadelta) for epochs 0 to 180 and γ_n is divided by 10 and σ_n is set to 0 beyond.

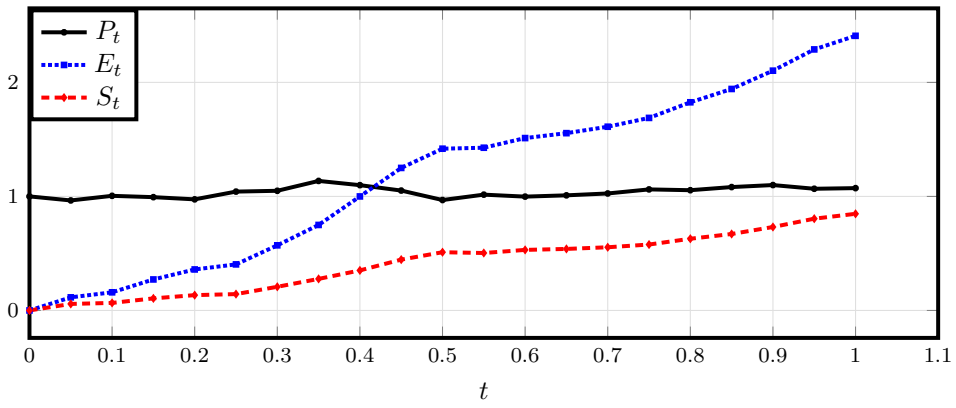

Figure 10: Example of trajectory for the oil drilling problem with $N = 20$.

5 Oil drilling

We consider the control problem in the management of natural resources applied to oil drilling introduced in [GKL18] and extended in [GGKL21]. The objective is for an oil driller, to balance the costs of extraction, storage in a volatile energy market. The oil price $P_t \in \mathbb{R}$ is assumed to be a Black-Scholes process:

$$dP_t = \mu P_t dt + \eta P_t dW_t. \tag{5.1}$$

The control is given by $q_t = (q_t^v, q_t^s, q_t^{v,s}) \in \mathbb{R}^3$ where q_t^v is the quantity of extracted oil immediately sold on the market per time unit, q_t^s is the quantity of extracted oil that is stored per time unit, $q_t^{v,s}$ is the quantity of stored oil that is sold per time unit. The cumulated quantities of extracted and stored oil at time t are respectively given by

$$E_t = \int_0^t (q_r^v + q_r^s)dr, \quad S_t = \int_0^t (q_r^s - q_r^{v,s})dr. \tag{5.2}$$

The extraction and storage prices are respectively given by

$$c_e(E_t) = \exp(\xi_e E_t), \quad c_s(S_t) = \exp(\xi_s S_t) - 1. \tag{5.3}$$

The constraints on the control are the following:

$$q_t^v,\ q_t^s,\ q_t^{v,s} \geq 0, \quad q_t^{v,s} \leq q^S, \quad q_t^v + q_t^s \leq K_0, \quad 0 \leq S_t \leq Q^S, \tag{5.4}$$

where q^S, K_0 and Q^S are operational bounds. The objective is

$$J(q) = -\mathbb{E}\left[\int_0^T e^{-\rho r} U\Big(q_r^v P_r + q_r^{v,s}(1-\varepsilon)P_r - (q_r^v + q_r^s)c_e(E_r) - c_s(S_r)\Big)dr\right], \tag{5.5}$$

where $U : \mathbb{R} \to \mathbb{R}$ is the utility function.

In the experiments we take

$$\begin{aligned} &T = 1,\ \mu = 0.01,\ \eta = 0.2,\ \rho = 0.01,\ \varepsilon = 0,\ K_0 = 5, \\ &\xi_e = 1\,\mathrm{e}{-2},\ \xi_s = 5\,\mathrm{e}{-3},\ q^S = 10,\ P_0 = 1,\ U(x) = x. \end{aligned} \tag{5.6}$$

The control q_t is given by a feedforward neural network with two hidden layers with 32 units and with ReLU activation while the output layer has several ReLU activations such that the constraints on q (5.4) are fulfilled[1]. An example of controlled trajectory is given in Figure 10.

The results are given in Figure 11 for the comparison of Langevin and non-Langevin algorithms with a single control. We do not display the results for the training with multiple controls however as we could not obtain satisfying results neither with Langevin nor-with non-Langevin methods.

6 Comments on the numerical experiments

We observe that in many cases and in various SOC problems, Langevin and Layer Langevin algorithms show improvement when compared with their respective non-Langevin counterparts, provided that N is large enough,

[1] We remark that $\max(q, K) = K - \mathrm{ReLU}(-q + K)$.

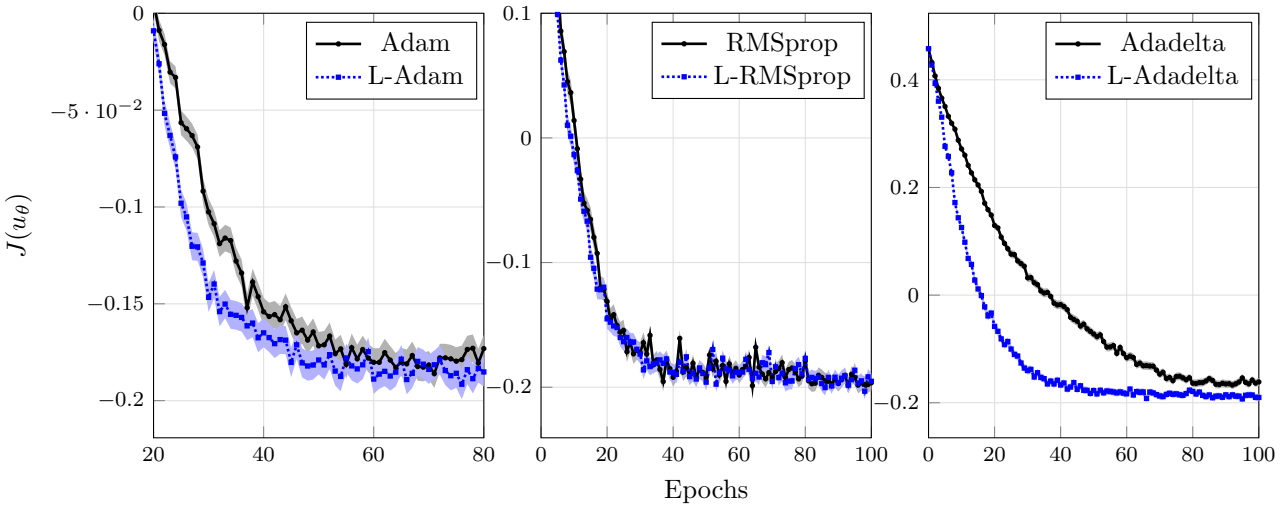

Figure 11: Comparison of algorithms during the training for the deep hedging control problem with $N = 50$. The schedules are $\gamma = 2\,\mathrm{e}{-3}$ ($2\,\mathrm{e}{-3}$, $5\,\mathrm{e}{-1}$) and $\sigma = 1\,\mathrm{e}{-3}$ ($2\,\mathrm{e}{-3}$, $5\,\mathrm{e}{-3}$) for Adam (resp. RMSprop, Adadelta) for epochs 0 to 60 (resp. 80, 80) and γ is divided by 10 and σ is set to 0 beyond.

which is remarkable for randomized algorithms. Langevin algorithms converge faster and/or toward a lower loss value. This is particularly visible for the Adadelta method. The gains are limited in some cases (see Figures 5 and 11 for RMSprop) but still the optimization procedure is improved.

The gains for the L-RMSprop algorithm remain limited however. In particular, we did not observe any significant improvement for fishing SOC with multiple controls, for deep hedging SOC with a single control and for oil drilling SOC. We do not have explanation for this fact.

The gains brought by Langevin algorithm increase with the depth of the network as shown in Figures 4 and 8. However, contrary to [Bra22], we did not observe overwhelming gains as N becomes very large. We believe that this is due (in part) to the particular structure of the deep SOC problem where the same control is repeated all along the trajectory.

As for SOC with multiple controls, we observe that Layer Langevin algorithms with a small number of Langevin layers (10%-30%) generally outperforms Vanilla Langevin methods while Vanilla Langevin may bring limited gains or be less efficient than the standard non-Langevin methods, see Figures 6 and 9 for Adam.

Acknowledgements

The authors thank Idris Kharroubi for helpful discussions.

References

[Ani19] Chandrasekaran Anirudh Bhardwaj. Adaptively Preconditioned Stochastic Gradient Langevin Dynamics. *arXiv e-prints*, page arXiv:1906.04324, June 2019.

[Bel57] Richard Bellman. *Dynamic programming*. Princeton University Press, Princeton, N. J., 1957.

[BGTW19] H. Buehler, L. Gonon, J. Teichmann, and B. Wood. Deep hedging. *Quant. Finance*, 19(8):1271–1291, 2019.

[BHLP22] Achref Bachouch, Côme Huré, Nicolas Langrené, and Huyên Pham. Deep neural networks algorithms for stochastic control problems on finite horizon: numerical applications. *Methodol. Comput. Appl. Probab.*, 24(1):143–178, 2022.

[Bra22] Pierre Bras. Langevin algorithms for very deep Neural Networks with application to image classification. *arXiv e-prints*, page arXiv:2212.14718, December 2022.

[CL21] René Carmona and Mathieu Laurière. Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games I: The ergodic case. *SIAM J. Numer. Anal.*, 59(3):1455–1485, 2021.

[DdVB15] Yann Dauphin, Harm de Vries, and Yoshua Bengio. Equilibrated adaptive learning rates for non-convex optimization. In *Neural Information Processing Systems*, 2015.

[DPG+14] Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and Attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2933–2941, Cambridge, MA, USA, 2014. MIT Press.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[GG05] Michael B. Giles and Paul Glasserman. Smoking adjoints: fast evaluation of greeks in monte carlo calculations. Technical Report NA05/15, Oxford University Computing Laboratory, 2005.

[GGKL21] M'hamed Gaïgi, Stéphane Goutte, Idris Kharroubi, and Thomas Lim. Optimal risk management problem of natural resources: application to oil drilling. *Ann. Oper. Res.*, 297(1-2):147–166, 2021.

[Gil07] Michael B. Giles. Monte Carlo evaluation of sensitivities in computational finance. Technical Report NA07/12, Oxford University Computing Laboratory, 2007.

[GKL18] Stéphane Goutte, Idris Kharroubi, and Thomas Lim. Optimal management of an oil exploitation. *International Journal of Global Energy Issues*, 41(1/2/3/4):69–85, 2018.

[GM05] Emmanuel Gobet and Rémi Munos. Sensitivity analysis using Itô-Malliavin calculus and martingales, and application to stochastic optimal control. *SIAM J. Control Optim.*, 43(5):1676–1713, 2005.

[GMDB16] Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 3059–3068. JMLR.org, 2016.

[Han18] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? In *NeurIPS*, pages 580–589, 2018.

[HE16] Jiequn Han and Weinan E. Deep Learning Approximation for Stochastic Control Problems. *Deep Reinforcement Learning Workshop, NIPS (2016)*, November 2016.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[Hoc91] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. diploma thesis, institut für informatik, lehrstuhl prof. brauer, technische universität münchen, 04 1991.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KD01] Harold J. Kushner and Paul Dupuis. *Numerical methods for stochastic control problems in continuous time*, volume 24 of *Applications of Mathematics (New York)*. Springer-Verlag, New York, second edition, 2001. Stochastic Modelling and Applied Probability.

[LCCC16] Chunyuan Li, Changyou Chen, David Carlson, and Lawrence Carin. Preconditioned stochastic gradient langevin dynamics for deep neural networks. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 1788–1794. AAAI Press, 2016.

[LPP21] Mathieu Laurière, Gilles Pagès, and Olivier Pironneau. Performance of a Markovian neural network versus dynamic programming on a fishing control problem. *arXiv e-prints, to appear in Probability, Uncertainty and Quantitative Risk*, page arXiv:2109.06856, September 2021.

[NVL+15] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding Gradient Noise Improves Learning for Very Deep Networks. *arXiv e-prints*, page arXiv:1511.06807, November 2015.

[PW99] Shige Peng and Zhen Wu. Fully coupled forward-backward stochastic differential equations and applications to optimal control. *SIAM J. Control Optim.*, 37(3):825–843, 1999.

[SLH+19] Kumar Shridhar, Joonho Lee, Hideaki Hayashi, Purvanshi Mehta, Brian Kenji Iwana, Seokjun Kang, Seiichi Uchida, Sheraz Ahmed, and Andreas Dengel. ProbAct: A Probabilistic Activation Function for Deep Neural Networks. *arXiv e-prints*, page arXiv:1905.10761, May 2019.

[TH12] T. Tieleman and G. E. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. Coursera: Neural Networks for Machine Learning, 2012.

[WLP+19] Ziyi Wang, Keuntaek Lee, Marcus A. Pereira, Ioannis Exarchos, and Evangelos A. Theodorou. Deep forward-backward sdes for min-max control. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 6807–6814, 2019.

[Zei12] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv e-prints*, page arXiv:1212.5701, December 2012.