# SOLVING STOCHASTIC CONTROL PROBLEMS WITH NEURAL NETWORKS
# (NEW CHAPTER) - REVISED 2ND EDITION

PIERRE HENRY-LABORDÈRE

Abstract. In this chapter, we introduce a primal-dual method for solving stochastic control problems based on the use of neural networks, stochastic gradient descent and a dual formulation of stochastic control problems as derived in [14]. This approach orginates from [12, 13, 8, 3]. Our algorithm is illustrated with various examples relevant in Mathematical Finance: the pricing in the case of uncertain volatility models, the pricing of buyback options, the optimal posting of CSA, the pricing of counterparty risk and finally the computation of initial margin. Parts of this research were published in [19, 20].

## 1. Neural networks in a nutshell

In the first section, we review key properties of neural networks and give some insights of their numerical implementations.

### 1.1. Universal approximations.

The relevance of neural networks for approximating a nonlinear $n$-dimensional function $f$ can be understood (although not fully justified) with the following result from Kolmogorov-Arnold, solving the 13th Hilbert problem:

**Theorem 1.1.** *There exists $n(2n+1)$ universal $C^0$ functions $\Phi_{ij} : [0,1] \mapsto [0,1]$ such that for all $C^0$ functions $f : [0,1]^n \mapsto [0,1]$, there exists $(2n+1)$ $C^0$ function $g_i : [0,1] \mapsto [0,1]$ for which*

$$f(x_1, \cdots, x_n) = \sum_{i=1}^{2n+1} g_i \left( \sum_{j=1}^{n} \Phi_{ij}(x_j) \right)$$

In short, a nonlinear $n$-dimensional function $f$ can be *exactly* recovered using one-dimensional $n(2n+1)$ universal $C^0$ functions, i.e., independent of the function we would like to approximate, and $(2n+1)$ one-dimensional specific functions. If we believe that the universal functions $\Phi_{ij}$ are affine transformations, depending on a matrix $W \in M_{n \times 2n+1}$ and a drift $b \in \mathbb{R}^{2n+1}$:

$$\Phi_{ij}(x_j) := W_{ij} x_j + (b_i/n)$$

and the functions $(g_i)_{1 \leq i \leq 2n+1}$ are proportional to the same function $\sigma : \mathbb{R} \mapsto [0,1]$: $g_i(x) = \omega_i \sigma(x)$, we get the definition of a neural network with *one* hidden layer of dimension $2n+1$.

In general, a neural network with *multiple* hidden layers and linear outputs, is a function $a_N : \mathbb{R}^p \to \mathbb{R}^q$ obtained by successive compositions of one-dimensional functions of two types: First affine transformations and then composition by a non-linear function $\sigma$ (which can be for example

Received by the editors April 2018.

tanh, the sigmoid function, the relu function $\max(x,0)$, ...). Finally, the output $y$ as a function of the input $x$ is given by:

$$y = a_N(x) \in \mathbb{R}^q, \quad x \in \mathbb{R}^p$$

with

$$\begin{aligned} a_l &= \sigma(W_l a_{l-1} + b_l), \quad l = 1, \cdots, N-1 \\ a_N &= W_N a_{N-1} + b_N \end{aligned}$$

and $a_0 = x \in \mathbb{R}^p$. The function $a_N$ can be represented by a network taking $(x_1, \cdots, x_p)$ as inputs and $(y_1, \cdots, y_p)$ as outputs (see Figure 1) and depending on some matrix weights $(W_l)_{1 \leq l \leq N}$ and some vector weights $(b_l)_{1 \leq l \leq N}$. In practise, we take $\sigma(x) := \tanh(x)$ or the sigmoid function $\sigma(x) = 1/(1+e^{-x})$. Here $N-1$ is called the number of hidden layers.

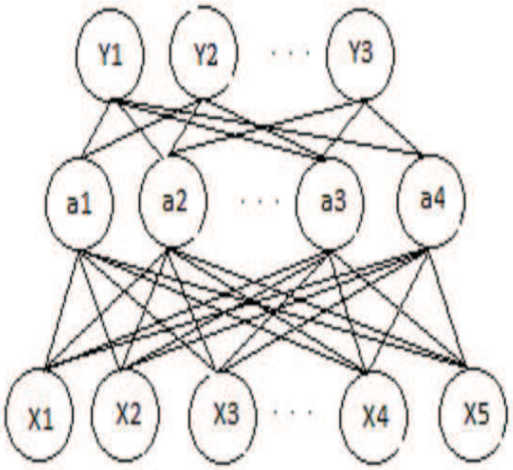


FIGURE 1. Representation of a neural network with 1 hidden layer ($p = 5, q = 3$).

The following theorem shows that a neural network can be seen as an "universal approximator", in particular can approximate any continuous function, providing the number of parameters of the neural network is large enough:

**Theorem 1.2** (Hornik [22])**.** *Suppose $\sigma$ is bounded and non-constant. The following statements hold:*

- *For any finite measure $\mu$ on $(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d))$ and $1 \leq p < \infty$, the set NN is dense in $\mathrm{L}^p(\mathbb{R}^d, \mu)$.*
- *If in addition $\sigma \in C(\mathbb{R})$, then the set NN is dense in $C(\mathbb{R})$ for the topology of uniform convergence on compact sets.*

1.2. **Forward and backward propagation.** Despite as being an universal approximator, the main and more important feature of a neural network is that the computation of the function $a_N(x)$ and its gradients with respect to the weight $(W_l, b_l)_{l=1,\cdots,N}$ can be achieved in $O(W)$ operations with the use of automatic differentiation. This is called the forward and backward propagation in the machine learning terminology. More precisely, from the differentiation chain rule, the gradients

for a function $j(a_N(x))$ are given by backward induction, hence its denomination, as

$$\begin{aligned}
\delta_i^N : &= [\nabla j(y)]_i, \quad y := a_N(x), \quad i = 1, \cdots, q \\
\left[\frac{\partial j}{\partial W_N}\right]_{ij} &= \delta_i^N a_{N-1}^j, \quad \left[\frac{\partial j}{\partial b_N}\right]_i = \delta_i^N, \quad \left[\frac{\partial j}{\partial a_{N-1}}\right]_i = \sum_k \left[W_N^\dagger\right]_{ik} \delta_k^N \\
\delta_i^{N-1} &:= \left[\frac{\partial j}{\partial a_{N-1}}\right]_i (1 - (a_{N-1}^i)^2) \qquad (1) \\
\left[\frac{\partial j}{\partial W_{N-1}}\right]_{ij} &= \delta_i^{N-1} a_{N-2}^j, \quad \left[\frac{\partial j}{\partial b_{N-1}}\right]_i = \delta_i^{N-1}, \quad \left[\frac{\partial j}{\partial a_{N-2}}\right]_i = \sum_k \left[W_{N-1}^\dagger\right]_{ik} \delta_k^{N-1}, \quad \cdots
\end{aligned}$$

Note that we have used here that $\tanh'(x) = 1 - \tanh(x)^2$. For the sigmoid function, we have instead $\sigma'(x) = \sigma(x)(1.0 - \sigma(x))$.

1.3. **Stochastic gradient descent.** By relying on the ease of the computation of the gradients with respect to the weights $\theta := (W_l, b_l)_{1 \le l \le N}$, the minimization of a functional

$$J(\theta) := \frac{1}{m} \sum_{i=1}^m j(a_N(x^i))$$

over $\theta$ can be obtained by using a stochastic gradient descent (in short SGD). Leaving apart efficient extensions (see for example [2] and Section 1.5), the basic method consists in drawing an uniform random variable $I_t$ in $[[1, m]]$ and then applying the (online) gradient descent on $j(a_N(x^{I_t}))$:

$$\theta_t \mapsto \theta_t - \eta_t \nabla_\theta j(a_N(x^{I_t})), \quad I_t \text{ Uniform} \in [[1, m]],$$

for a positive parameter $\eta_t \in \mathbb{R}_+^*$ at a step $t$, called learning rate, and then iterate until convergence. The convergence is stated by the following theorem:

**Theorem 1.3** (Robbins-Monro)**.** *If we impose that $\sum_{t=1}^\infty \eta_t = \infty$ and $\sum_{t=1}^\infty \eta_t^2 < \infty$, then $\lim_{t\to\infty} \theta_t = \theta^*$ almost surely with $\theta^*$ a local minimizer of $J$ (minimizer if $J$ is convex).*

Note that in our numerical experiments, we have observed than, surprisingly, choosing a constant learning rate gives better convergence results.

*Example* 1.4 (Learning $f(x) := x^2$). In Figure 2, as a simple example, we have plotted the one-dimensional function $y = a_N(x)$ obtained by minimizing the $\mathrm{L}^2$–norm with respect to $\theta$:

$$\frac{1}{m} \sum_{i=1}^m (a_N(x^i) - f(x^i))^2$$

over a train set $(x_i)_{1 \le i \le m} \in [-1, 1]$ with $f(x) := x^2$, and composed of $m = 100$ points.

1.4. **Learning option pricing formula.**

*Example* 1.5 (Learning Black-Scholes formula). We learn the Black-Scholes formula for call options with maturity $T = 1$ year using a training set of dimension 3400 consisting in Black-Scholes prices for different values $(K/X_0, \sigma) \in [0.1, 2] \times [0.1, 0.8]$ (red dots in Figure 3). Once the formula is trained, our NN approximation is checked on 3400 independent values $(K/X_0, \sigma)$. We have plotted the error in blue.

*Example* 1.6 (Learning Black-Scholes autocallable). Similar experiments (see Figure 4) for an autocall payoff with maturity $T = 1$ year and spot value $S_0 = 1$:

$$F_T := 4\% \sum_{i=1}^{n-1} 1_{X_i > 1} 1_{M_{i-1} < 1} + g(X_n) 1_{M_{n-1} < 1}, \quad g(s) = 1 - (1 - X_n) 1_{X_n < 0.6}$$

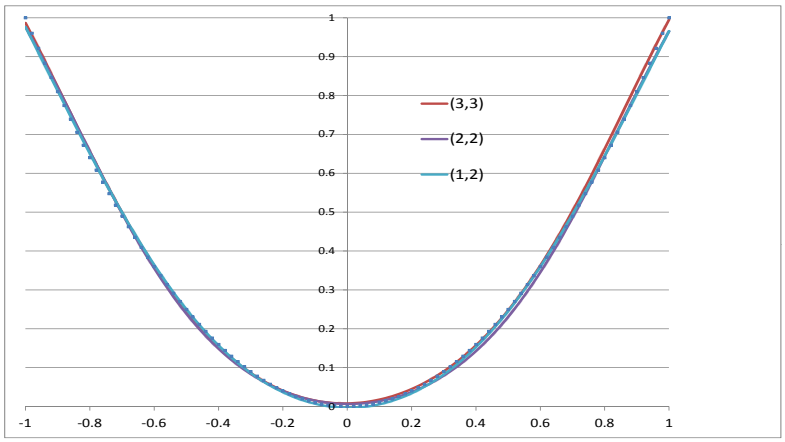


FIGURE 2. Fit over $f(x) = x^2$ with $N-1$ hidden layers with matrix weights $W$ of dimension $M$ (denoted $(M, N)$).

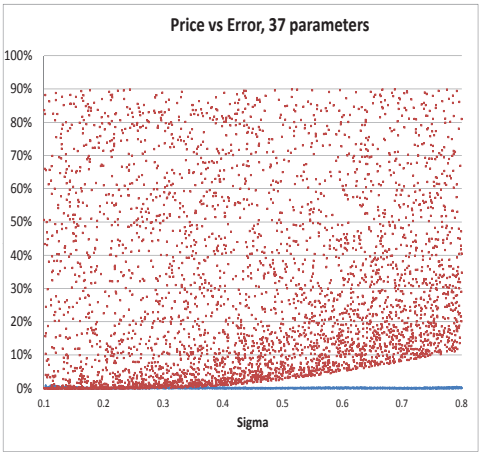


FIGURE 3. Approximation with a neural network with 2 hidden layers of dim. 4. Red dots: Black-Scholes prices. Blue dots: errors.

where $M_i := \max_{j \leq i} X_j$ denotes the running maximum computed monthly. Our NN is trained on 3400 prices, obtained by Monte-Carlo simulation, consisting in different volatilities $\sigma \in [0.1, 0.8]$. The model is then validated on 3400 independent additional volatilities.

As a more involved example in [21], neural networks are used to approximate a calibration function that takes as inputs market instruments, such as yield curves, swaption volatilities, and gives a model parameter (here Hull-White extended Vasicek model) as an output. Using this approximation, one does not need to (re)-calibrate the model using a numerical procedure.

### 1.5. **Accelerated stochastic gradient descent.**

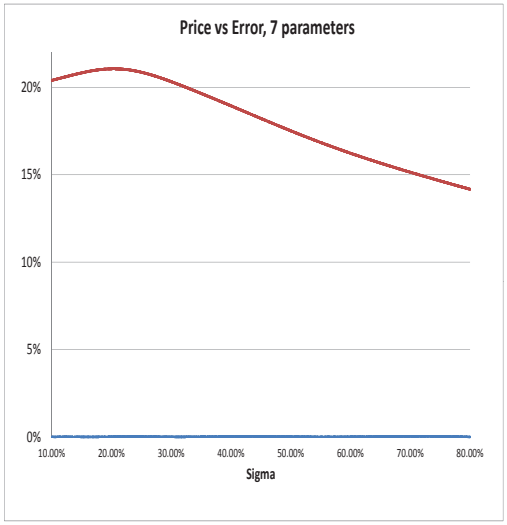


FIGURE 4. Approximation with a neural network with 1 hidden layer of dim. 3. Red dots: Monte-Carlo prices. Blue dots: errors.

1.5.1. *ADAM.* SGD could be speed-up by using a momentum method as ADAM [1]. It consists in having an adaptative learning rate and can be described by the following algorithm:

(1) Set $t := 0$ and $m_0 = v_0 := 0$. Fix 4 parameters $\beta_1$, $\beta_2$, $\epsilon$ and $\alpha$. In practice, we take the default values: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and $\alpha = 10^{-3}$.
(2) Compute the gradient $G_t := \frac{1}{m'} \sum_{i=1}^{m'} \nabla_\theta J_i(\theta_t)$ at step $t$ by back-propagation with $m'$ mini-batches as explained above.
(3) Set

$$\begin{aligned} m_{t+1} &= \beta_1 m_t + (1-\beta_1) G_t, \quad v_{t+1} = \beta_2 v_t + (1-\beta_2) G_t^2 \\ \hat{m}_{t+1} &:= \frac{m_{t+1}}{1-\beta_1^t}, \quad \hat{v}_{t+1} := \frac{v_{t+1}}{1-\beta_2^t} \end{aligned}$$

and update the weight by

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1} + \epsilon}}$$

(4) Set $t := t + 1$ and iterate steps (2) and (3) until convergence.

If not specified, this will be our algorithm of choice for SGD.

1.5.2. *SAG.* The SAG method has been shown to perform better for convex problems in [2]. It consists first in computing all gradients for all the training examples:

$$G_i^0 = \nabla_\theta j(a_N(x^i)), \quad \forall i = 1, \cdots, m$$

This can be done only when $m$ is not too large, otherwise the computer memory limit can be reached. Then at step $t+1$, the weights are updated by

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{m} \sum_{i=1} G_i^t$$

where at each iteration a random training example $I_t$ is selected and we set

$$\begin{aligned} G_i^t &= \nabla_\theta j(a_N(x^i)) \quad \text{if } i = I_t \\ &= G_i^{t-1} \quad \text{otherwise} \end{aligned}$$

Despite this algorithm shows better convergence than SGD (or ADAM) for convex problems, in our numerical experiments, it gives poor results (that we do not display) and therefore will not be considered.

1.6. **Some recipes for efficient training of NN.** Below, we have listed some traditional cooking recipes useful to speed-up the training of NN, that we have used in our numerical implementation in `C++` (see Chapter 5 in [6] and Part II in [5] for additional recipes). Note that these recipes miss sometimes proper mathematical justifications and are only validated by intensive experiments.

- Weight initialization: The weights of NN are initialized randomly using the rule:

$$W_{d,d'} := \sqrt{\frac{6}{d+d'}} U, \quad b := 0$$

  where $W_{d,d'}$ is a matrix connecting a $d$-dimensional layer with a $d'$-dimensional layer and $U$ is a uniform random number in $[-1, 1]$. Note that for the output layer (with a sigmoid activation in order to get outputs in $[0, 1]$), the weight initialization are multiplied additionally by a factor 4.
- Batch normalization: In order to have similar magnitude for the inputs of the NN, the inputs $(x_i)_{i=1,\cdots,N_{\rm MC}}$ should be normalized by $\hat{x}_i := \frac{x_i - m}{\sqrt{\rm var}}$ with $m := \frac{1}{N_{\rm MC}} \sum_{i=1}^{N_{\rm MC}} x_i$ and ${\rm var} := \frac{1}{N_{\rm MC}} \sum_{i=1}^{N_{\rm MC}} x_i^2 - m^2$.

1.7. **Enhancing back-propagation with genetic optimization.** The functional $J$ is a high-dimensional non-convex function and in practice SGD converges towards a local maximizer, in particular with a large number of hidden layers. Indeed, the number of local maximizers grows with the number of hidden layers. In order to circumvent this difficulty, we could complement the training of the NN using back-propagation with genetic optimizations that have the property of converging to the global maximizer. This was tested also in [29]. Note that these algorithms are generally quite difficult to fine-tune (for example the decrease of the temperature in the simulated annealing method) and could lead to poor results.

First, the NN will be trained with a genetic optimization in order to converge closely to the global maximizer. Then, we will use a (ADAM) SGD to converge to this maximizer. We have tested two different global optimizations: Particle swarm optimization (in short PSO), first developed in [24] and the simulated annealing algorithm (in short SA). They provide both similar results. PSO algorithm was recently used for optimizing MVA in [25].

Let us consider the *minimization* of a function $f : \mathbb{R}^d \mapsto \mathbb{R}$, not necessarily smooth as we don't use gradients.

1.7.1. *Particle swarm optimization.* PSO can be described by the following algorithm:

(1) Set $t := 0$. We choose randomly the positions $(x_i(0))_{i=1,\cdots,N}$ of $N$ particles and set $p_i(0) = x_i(0)$ and $p_f(0) = {\rm argmin}_{(x_i(0))_{i=1,\cdots,N}} f(x_i(0))$.
(2) Then, the positions of the particles evolve according to the dynamics

$$\begin{aligned} v_i(t+1) &= \omega v_i(t) + c_1 U_1 (p_i(t) - x_i(t)) + c_2 U_2 (p_f(t) - x_i(t)) \\ x_i(t+1) &= x_i(t) + v_i(t+1) \end{aligned}$$

with $p_i(t)$ is the best position for particle $i$ and $p_f(t)$ is the best global position at step $t$. $\omega$ is a parameter in $(0,1)$ and $c_1$ and $c_2$ are two positive coefficients, typically equal to 2. $U_1$ and $U_2$ are uniform random variables in $[-1,1]$.

(3) We stop after $n$ iterations.

Note that this algorithm does not depend on the gradient of the function $f$.

1.7.2. *Simulated annealing.* SA can be described by the following algorithm:

(1) Set $t=0$, $T_t := 1$ (initial temperature) and choose randomly $x_t \in \mathbb{R}^d$.
(2) Generate a new random position $x_{t+1}$. This is accepted, $x_t := x_{t+1}$, with a probability $e^{-\frac{(f(x_{t+1})-f(x_t))}{T_t}}$. In particular when $f(x_{t+1}) < f(x_t)$, the trial is always accepted. Do this $m$ times (i.e., number of epochs).
(3) Set $t := t+1$ and reduce the temperature by $T_{t+1} = \alpha T_t$, $\alpha \in (0,1)$. Goto step 2. In practice, we take $\alpha = 0.9$.

This algorithm converges to the global minimizer if $T_t \geq C/\ln t$.

*Example* 1.7. In order to illustrate PSO and SA, we consider a two-dimensional function $f$ (see Figure 5) defined by

$$f(x_1,x_2) := (4-2.1x_1^2+\frac{x_1^4}{3})x_1^2 + x_1x_2 + 4(x_2^2-1)x_2^2$$

for which the global minimizer is $-1.03163$ attained for $(0.09,-0.71)$ (or $(-0.09,0.71)$). For PSO, we have used 50 particles and 100 iterations. For SA, we have used 100 epochs and 100 iterations. The results are reported in Table 1. On this specific example, PSO seems to perform better than SA.

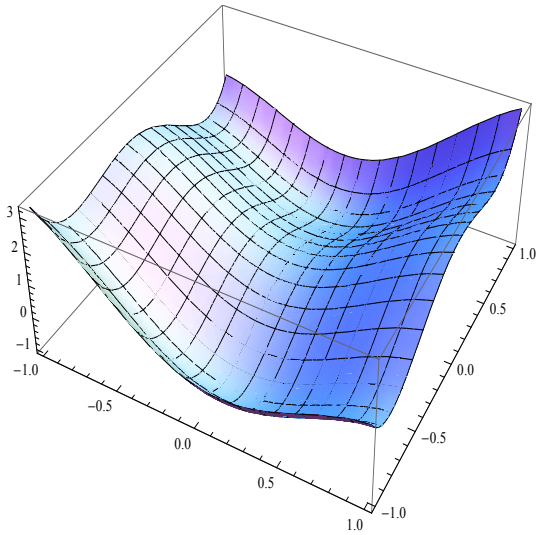


FIGURE 5. Function $f(x_1,x_2) := (4-2.1x_1^2+\frac{x_1^4}{3})x_1^2 + x_1x_2 + 4(x_2^2-1)x_2^2$.

In the next section, we deviate from the paradigm where neural networks are used to learn pricing or calibration formulas and explain their efficiency for solving also stochastic control problems and the associated non-linear Hamilton-Jacobi-Bellman PDEs.

| | Min | $x_1$ | $x_2$ |
|---|---|---|---|
| EXACT | $-1.03163$ | 0.09 | $-0.71$ |
| PSO(nb_par=50,iter=100) | $-1.03163$ | $-0.09$ | 0.71 |
| SA (epoch=100, iter=100) | $-1.03106$ | 0.08 | $-0.71$ |

TABLE 1. Numerical (global) minimization of $f(x_1, x_2)$.

## 2. Solving SCP with neural networks

We consider a generic stochastic control problem (in short SCP):

$$u(t,x) := \sup_{a_\cdot \in \mathcal{A}} \mathbb{E}_{t,x}[F(X_T^a)] \tag{2}$$

where the set $\mathcal{A}$ denotes the space of adapted processes valued in a compact domain $A$ and $X_t^a$ is an adapted process controlled by $a$. The numerical resolution of a generic SCP based on NN, as first suggested in [12], is quite easy to understand. In the case of our SCP (10), it consists in using a (flexible enough) parameterization of the control $(a_n)_{n=0,\cdots,N}$, discretized at times $(t_n)_{1\leq n\leq N}$ with $t_0 := 0$ and $t_N := T$, through the use of $N+1$ feedforward NN and then maximize the gain functional with respect to the weights of these NN, that we denote generically below by $\theta := (\theta_n)_{n=0,\cdots,N}$. Note that a similar approach was given in [11] for solving the SCP associated to the uncertain volatility model by parameterizing the frontier (without using NN here) where we switch from the lower to upper volatility.

### 2.1. The algorithm in a nutshell.

By taking the number $N_{\text{MC}}$ of Monte-Carlo paths to be large, the problem (10) can then be approximated by

$$\max_\theta \tilde{\mathbb{E}}[F] := \frac{1}{N_{\text{MC}}} \sum_{i=1}^{N_{\text{MC}}} F(X_{t_N}^{a(\theta),i}) \tag{3}$$

Once the optimization will be done, the algorithm will generate a sub-optimal control by construction, and therefore by performing a second independent Monte-Carlo algorithm, we will produce a lower bound. This is the procedure that we apply in the Longstaff-Schwarz algorithm for pricing American options. We will show in Section 2.2 that the optimal bound can be attained by our algorithm by increasing the complexity of the NN.

### 2.2. The bias-variance tradeoff.

In this section, we analyse the error in our algorithm and discuss the tradeoff between the number of Monte-Carlo paths needed to train our NN (with SGD) and the complexity of our NN (in terms of the number of hidden layers and their dimensions). As before, we denote $(a_n)_n^* \in \mathcal{A}$ the optimal control:

$$(a_n)_n^* := \text{argmax}_{(a_\cdot)\in\mathcal{A}} \mathbb{E}[F]$$

Then, we denote $(a_n)_n^{*,\text{NN}}$ the optimal control to our SCP obtained by restricting the class $\mathcal{A}$ to the subset $\mathcal{A}_{\text{NN}} \subset \mathcal{A}$ of adapted process approximated by neural network functions:

$$(a_n)_n^{*,\text{NN}} := \text{argmax}_{(a_\cdot)\in\mathcal{A}_{\text{NN}}} \mathbb{E}[F]$$

Similarly, we denote $(a_n)_n^{*,\text{NN},N_{\text{MC}}}$ the control obtained by training our NN on the functional (3) using $N_{\text{MC}}$ Monte-Carlo paths:

$$(a_n)_n^{*,\text{NN},N_{\text{MC}}} := \text{argmax}_{(a_\cdot)\in\mathcal{A}_{\text{NN}}} \tilde{\mathbb{E}}[F]$$

where the operator $\tilde{\mathbb{E}}[\cdot]$ is the empirical sum over the $N_{\mathrm{MC}}$ paths. As $a^{*,\mathrm{NN},\mathrm{N_{MC}}}_{\cdot}$ is a sub-optimal control by construction, we have the lower bound:

$$\mathbb{E}^{a^{*,\mathrm{NN},\mathrm{N_{MC}}}}[F] \leq \mathbb{E}^{a^*}[F]$$

The notation $\mathbb{E}^a[F]$ (resp. $\tilde{\mathbb{E}}^a[F]$) means that the (resp. empirical) expectation is computed using a control $a_\cdot$. Furthermore, we have the telescopic identity:

$$\begin{aligned}\mathbb{E}^{a^*}[F] &= \mathbb{E}^{a^{*,\mathrm{NN},\mathrm{N_{MC}}}}[F] + (\tilde{\mathbb{E}}^{a^{*,\mathrm{NN},\mathrm{N_{MC}}}}[F] - \mathbb{E}^{a^{*,\mathrm{NN},\mathrm{N_{MC}}}}[F]) + (\tilde{\mathbb{E}}^{a^{*,\mathrm{NN}}}[F] - \tilde{\mathbb{E}}^{a^{*,\mathrm{NN},\mathrm{N_{MC}}}}[F]) \\ &+ (\mathbb{E}^{a^{*,\mathrm{NN}}}[F] - \tilde{\mathbb{E}}^{a^{*,\mathrm{NN}}}[F]) + (\mathbb{E}^{a^*}[F] - \mathbb{E}^{a^{*,\mathrm{NN}}}[F])\end{aligned}$$

As $\tilde{\mathbb{E}}^{a^{*,\mathrm{NN},\mathrm{N_{MC}}}}[F] \geq \tilde{\mathbb{E}}^{a^{*,\mathrm{NN}}}[F]$ by definition of $(a_n)^{*,\mathrm{NN},N_{\mathrm{MC}}}_n$, this implies that

$$\mathbb{E}^{a^*}[F] \leq \mathbb{E}^{a^{*,\mathrm{NN},\mathrm{N_{MC}}}}[F] + 2\max_{a\in\mathcal{A}} |\mathbb{E}^a[F] - \tilde{\mathbb{E}}^a[F]| + |\mathbb{E}^{a*}[F] - \mathbb{E}^{a^{*,\mathrm{NN}}}[F]|$$

From the central limit theorem, we can control the error $\max_{a\in\mathcal{A}} |\mathbb{E}^a[F] - \tilde{\mathbb{E}}^A[F]|$ that scales as $1/\sqrt{\mathrm{N_{MC}}}$. Then, we use that a NN can approximate any continuous function on a compact domain with a single hidden layer with a large dimension $d$: From Theorem 1.2, for all $\epsilon > 0$,

$$|\mathbb{E}^{a^*}[F] - \mathbb{E}^{a^{*,\mathrm{NN}}}[F]| \leq \epsilon$$

for $d$ large enough. This implies the convergence of our algorithm as the number of Monte-Carlo paths $\mathrm{N_{MC}}$ and the dimension $d$ go to infinity:

**Proposition 2.1.**

$$\lim_{\mathrm{N_{MC}}\to\infty, d\to\infty} \mathbb{E}^{a^{*,\mathrm{NN},\mathrm{N_{MC}}}}[F] = \mathbb{E}^{a^*}[F]$$

As usual, the relevant number of MC paths can be easily determined by computing the variance of our MC estimator.

2.3. **Solving optimal stopping problem.** As explained in [8], we generalize the above approch to optimal stopping problem, corresponding to the pricing of a Bermudean option with payoff $g$ and discrete exercise dates in $\{0, 1, \cdots, N\}$:

$$u_0 := \sup_{\tau_0\in\{0,\cdots,N\}} \mathbb{E}[g(X_\tau)]$$

where $X$ is a Markov process. $\tau_0$ is a stopping time in $\{0, \cdots, N\}$, meaning that the event $\{\omega : \tau_0 \leq n\}$ will be a measurable fonction of the variables $(X_0, \cdots, X_n)$. We also introduce the sequence of auxiliary stopping problems:

$$u_n := \sup_{\tau_n\in\{n,\cdots,N\}} \mathbb{E}[g(X_\tau)]$$

where $\tau_n$ is a stopping time in $\{n, \cdots, N\}$. Following [8], we write the stopping times $(\tau_n)_{n=0,\cdots,N}$ as

$$\tau_N = N f_N(X_N) := N \tag{4}$$

$$\tau_n = n f_n(X_n) + \tau_{n+1}(1 - f_n(X_n)), \quad f_n \in \{0, 1\} \tag{5}$$

where $f_n$ is a measurable function of $X_n$ with $f_N(X_N) := 1$. From the recurrence (5), we obtain

$$\tau_n = \sum_{k=n}^{N} k f_k(X_k) \prod_{j=n}^{k-1}(1 - f_j(X_j)), \quad \prod_{j=n}^{n-1}(1 - f_j(X_j)) := 1$$

These equations are obvious: If $f_n := 1$, this means that we exercise and therefore $\tau_n = n$. If $f_n := 0$, we continue and therefore $\tau_n = \tau_{n+1}$. From the decomposition of the stopping times $(\tau_n)_{n=0,\cdots,N}$ in terms of the exercise policy functions $(f_n)_{n=0,\cdots,N} \in \{0, 1\}$, we can derived our

algorithm based on NN. It consists in approximating each $f_n(X_n)$ by a neural network $f_n^{\theta_n}(X_n)$ valued in [0,1], using for example a sigmoid activation for the outputs. The algorithm could then be described by the following steps [8]:

(1) Using a SGD with backpropagation, we solve:

$$\max_{\theta_n} \mathbb{E}[g(X_n)f^{\theta_n}(X_n) + g(X_{\tau_{n+1}})(1 - f^{\theta_n}(X_n))]$$

(2) Repeat step (1) backward in time up to $t = 0$.
(3) Then, we perform a second independent MC using our suboptimal stopping time. This leads to a lower bound. Note that by using the Rogers dual formula (and the Broadie-Andersen implementation for this stopping time $(\tau_n^{\theta_n})_{n=0,\cdots,N}$), we can compute an upper bound.

## 3. Application 1: Uncertain volatility model

As the first application, we consider the pricing a payoff $F_T$ in the (multi-dimensional) uncertain volatility model (in short UVM). The super-replication price consists in solving the following SCP:

$$u_0 := \sup_{\sigma \in [\underline{\sigma}, \overline{\sigma}]} \mathbb{E}[F_T], \quad \frac{dS_t}{S_t} := \sigma_t . dW_t$$

As a first parametrization, we take as inputs of the neural networks the asset values:

$$\sigma^*(t_n, S) = (\overline{\sigma} - \underline{\sigma})a_n^*(S) + \underline{\sigma}, \quad a_n^* \in [0, 1], \quad \text{Parameterization } \mathbf{I}$$

Here, we have used a sigmoid activation functions for the outputs (instead of a linear) to ensure that $a_n^* \in [0, 1]$ and therefore $\sigma^*(t_n, S) \in [\underline{\sigma}, \overline{\sigma}]$. In all our numerical experiments, we have taken $S_0^i = 1$ and $\underline{\sigma}^i = 0.1, \overline{\sigma}^i = 0.2$ for each asset $i = 1, \cdots, N$. Once the optimization over $\theta$ is done, we perform a second independent Monte-Carlo simulation leading to a robust lower bound, even if $\theta^*$ is a local maximizer or even an arbitrary point. We have used for the training and repricing $2^{16}$ Monte-Carlo paths, an identical discretization for the spot and the control $\Delta t_{\text{Euler}} = \Delta t_{\text{Control}} = 1/12$ and $10^6$ SGD iterations. For each payoff (in 1d or 2d) (see Table 3), we have computed the Black-Scholes price using mid-volatility $(\underline{\sigma} + \overline{\sigma}/2)$, the exact price obtained using a PDE solver and finally our lower bound using neural networks with 4 hidden layers of dimension 4 (see Column NN(4,4) I). Let us emphasize again that our choice of the number of hidden layers is only dictated by experiments.

| $F_T$ | BS$(\underline{\sigma} + \overline{\sigma}/2)$ | PDE | NN(4,4) I | NN(4,4) II |
|---|---|---|---|---|
| $(S_T - 1.0)^+$ | 5.98 | 7.97 | 7.97 | 7.97 |
| $(S_T - 0.9)^+ - (S_T - 1.1)^+$ | 9.52 | 11.20 | 10.81 | 11.03 |
| $(S_T^1 - S_T^2)^+$ $(\rho = 0)$ | 8.45 | 11.25 | 11.25 | 11.25 |
| $(S_T^1 - S_T^2)^+$ $(\rho = -0.5)$ | 10.34 | 13.75 | 13.75 | 13.75 |
| $(S_T^1 - 0.9S_T^2)^+ - (S_T^1 - 1.1S_T^2)^+$ $(\rho = -0.5)$ | 9.04 | 11.41 | 11.16 | 11.35 |

In the case of the call spread option $(S_T - 0.9)^+ - (S_T - 1.1)^+$ and $(S_T^1 - 0.9S_T^2)^+ - (S_T^1 - 1.1S_T^2)^+$, we do not get the exact result. From the HJB equation, we know that the optimal volatility is not only a function of the time $t$ and the spot values but is a function of the Gamma $\nabla_x^2 u(t, S)$ with $u$ the solution of the HJB PDE:

$$\sigma^*(t, S) = \Sigma(\nabla_x^2 u(t, S)), \quad \partial_t u + \frac{1}{2}\Sigma(\nabla_x^2 u(t, S))^2 \nabla_x^2 u(t, S) = 0$$

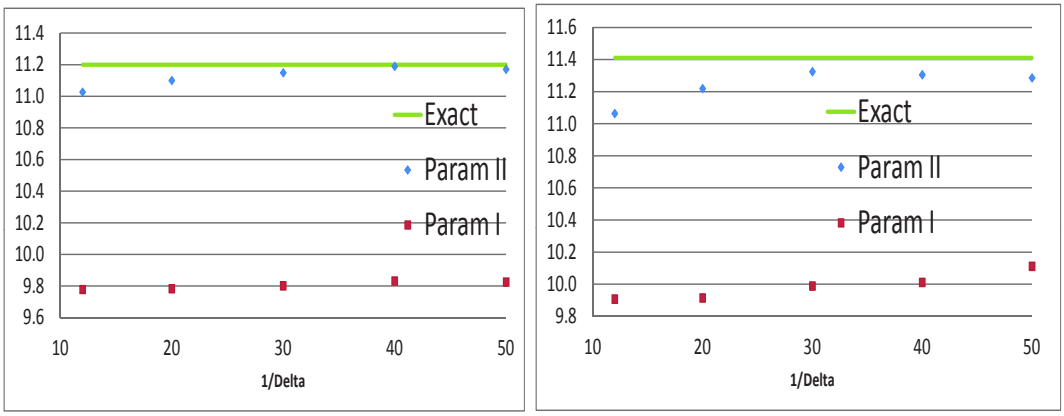


FIGURE 6. Left: $(S_T^1-0.9)^+-(S_T^1-1.1)^+$. Right: $(S_T^1-0.9S_T^2)^+-(S_T^1-1.1S_T^2)^+$. NN= (2,10).

Therefore as a better parameterization, we take as inputs of the neural networks the Black-Scholes Gamma, $\Gamma_{\text{BS}}$, computed with the mid-volatility, which can be computed analytically for our payoffs under consideration:

$$\sigma^*(t_n,S)=(\overline{\sigma}-\underline{\sigma})a_n^*(\Gamma_{\text{BS}}(t_n,S))+\underline{\sigma}, \quad \text{Parameterization } \mathbf{II}$$

As expected (see Table 3 and column NN(4,4) II), this parameterization greatly improves our lower bounds which are now close to the exact PDE results.

Note that our numerical implementation involves the choice of a time-discretization for the control. It should be noted that this induces an error which is bounded by:

**Proposition 3.1** (Krylov)**.**

$$|u_0-u_0^{\Delta t_{\text{Control}}}|\leq C\ (\Delta t_{\text{Control}})^{\frac{1}{3}}$$

Below, we have plotted our lower bound for different timesteps (see Figure 6).

**Computing the gradients.** The use of SGR requires the computation of $\partial_\theta\Phi(S_{T_1},\cdots,S_{T_n})$. We assume that the control $\sigma_\theta$ is piecewise constant over the intervals $[t_{i-1},t_i]$:

$$dS_t=S_t\sigma_\theta(S_{t_{i-1}})dW_t, \quad \forall t\in[t_{i-1},t_i]$$

This implies that

$$\partial_\theta\Phi(S_{T_1},\cdots,S_{T_n})=\partial_{\sigma_\theta(S_{t_{i-1}})}\Phi(S_{T_1},\cdots,S_{T_n})(\overline{\sigma}-\underline{\sigma})\partial_\theta a_{i-1}^*(S_{t_{i-1}}) \tag{6}$$

where $\partial_{\sigma_\theta(S_{t_{i-1}})}\Phi(S_{T_1},\cdots,S_{T_n})$ can be computed using a finite-difference scheme. Instead of equation (6), we could use

$$\partial_\theta\Phi(S_{T_1},\cdots,S_{T_n})=\sum_{i=1}^n\partial_{S_{T_i}}\Phi(S_{T_1},\cdots,S_{T_n})\partial_\theta S_{T_i})$$

This equation is more appropriate if there is less constat dates than discretization dates for the piecewise contant volatility.

## 4. Application 2: Buyback options

Let us define the processes, controlled by $\eta_\cdot \in [\underline{\eta}, \overline{\eta}]$:

$$\begin{aligned}\frac{dS_t}{S_t} &:= \sigma dW_t, \quad dI_t := S_t dt, \quad I_0 = 0\\ dQ_t &:= \eta_t dt, \quad Q_0 = 0, \quad dC_t := \eta_t S_t dt, \quad C_0 = 0\end{aligned}$$

A buyback option delivers at the maturity $T$:

$$F_T := g\left(Q_\tau\left(\frac{C_\tau}{Q_\tau} - \frac{I_\tau}{\tau}\right)\right)1_{\tau\leq T} + \Lambda(K - C_T)^+$$

with $\tau := \inf\{t \,:\, C_t \geq K\}$ and $g(x) := x1_{x>0} + \alpha x 1_{x\leq 0}$. The SCP reads then:

$$u_0 := \inf_{\eta_\cdot \in [\underline{\eta}, \overline{\eta}]} \mathbb{E}[F_T]$$

Here $\Lambda(K - C_T)^+$ with $\Lambda$ a large constant is a penalty term which ensures that $\tau < T$ almost surely when solving the SCP. The HJB equation is a $4d$-semilinear PDE depending on the four space variables $(s, i, q, c)$:

$$\partial_t u(t,s,i,q,c) + \frac{1}{2}\sigma^2 s^2 \partial_{ss} u + s\partial_i u + \inf_{\eta\in[\underline{\eta},\overline{\eta}]} \eta(\partial_q u + s\partial_c u) = 0, \quad \forall c \leq K$$

with the boundary condition:

$$u(t,s,i,q,c := K) = g(K - \frac{i}{t}q), \quad \forall t \leq T$$

The optimal control is:

$$\eta^* = (\eta_{\min} - \eta_{\max})(\partial_q u + s\partial_c u)^+ + \eta_{\max}$$

and this corresponds to a bang-bang solution. At order 0 in the volatility, $u = g(K - \frac{i}{t}q)$ and the optimal control $\eta_t^* = \eta(t, S_t/S_0, Q_t, C_t, I_t/t)$ depends on the sign of

$$\mathrm{sgn}(S_t - V_t)\mathrm{sgn}(C_t - V_t Q_t)$$

As a numerical example, we take $K = 335.10^6$, $S_0 = 44$, $\sigma = 0.2$, $\alpha = 0$ or $\alpha = 1$, and $T = 100$ days. $\eta_{\max} = 128.10^6$ and $\eta_{\min} = 11.10^6$ We have plotted of our upper bounds as a function of the number of iterations used in ADAM, eventually complemented with PSO. We have used neural networks with 2 hidden layers of dimension 10 and 4 hidden layers of dimension 4. For $\alpha = 0$, the lowest upper bound is 0.00 which must been optimal as the payoff $F_T$ is positive. For $\alpha = 1$, the lowest upper bound is $-0.68$.

## 5. Application 3: Optimal posting of CSA

For the sake of completeness, we recall briefly the formulation of optimal posting as a SCP, first considered in [27]. The collateral amount, denoted $C_n$ at a date $t_n$, can be posted in two assets $A$ and $B$, with (positive) quantities $A_n, B_n$ and instantaneous rates $r_n^A$ and $r_n^B$. The collateral amount $C_n$ (positive with our convention) at $t_n$ is therefore equal to

$$A_n + B_n := C_n \tag{7}$$

The functional $F$, corresponding to the CSA gain, that we want to maximize is

$$F := \sum_{n=0}^{N}(r_n^A A_n + r_n^B B_n)\delta t_n$$

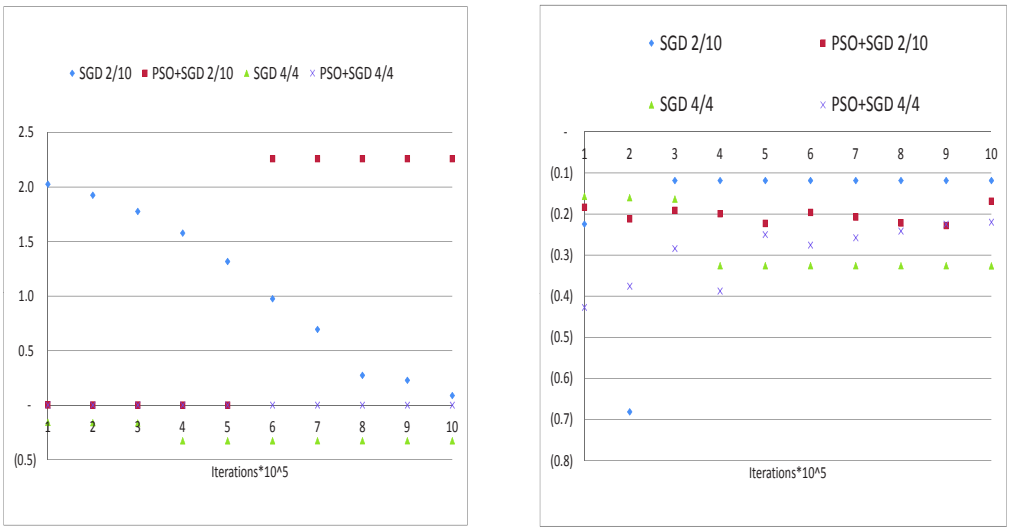


FIGURE 7. Upper bounds for different number of hidden layers and dimension as a function of number of iterations used in ADAM. Left: $\alpha = 0$. Right: $\alpha = 1$.

that we decompose using (7) as $F := F_+ + F_-$ with

$$F_+ = \sum_{n=0}^{N} (r_n^{AB} A_n + r_n^B C_n) \delta t_n \; 1(-\mathrm{MtM}_n > 0)$$
$$F_- = \sum_{n=0}^{N} (r_n^{AB} A_n + r_n^B C_n) \delta t_n \; 1(-\mathrm{MtM}_n < 0)$$

Here $\delta t_n := t_{n+1} - t_n$, $r_n^{AB} := r_n^A - r_n^B$ and $\mathrm{MtM}_n$ denotes the mark-to-market at $t_n$ (see below Equations (17) or (18) given the explicit relation between $\mathrm{MtM}_n$ and $C_n$ - without thresholds $C_n = |\mathrm{MtM}_n|$). $F_+$ (resp. $F_-$) corresponds to the gain that the seller (resp. buyer) wants to maximize.

Due to the so-called "rights of substitution", the set $\mathcal{A}$ of adapted process $(A_n)_{n=0,\cdots,N}$ needs to satisfy the following constraints in order to be an admissible strategy (see [27] for additional details):

- If $\Delta C_n := C_n - C_{n-1} \geq 0$ then the difference $\Delta C_n$ can be added to $A$ or $B$ without retrieving from $A$ or $B$:

$$0 \leq A_n - A_{n-1} \leq \Delta C_n \tag{8}$$

- If $\Delta C_n < 0$ then we can subtract up to $\Delta C_n$ from $A$ or $B$ without adding anything to either of them:

$$\max(-A_{n-1}, \Delta C_n) \leq A_n - A_{n-1} \leq \min(C_n - A_{n-1}, 0) \tag{9}$$

Let us mention additionally that if there is a change in the mark-to-market sign between $t_{n-1}$ and $t_n$, i.e., $\mathrm{MtM}_{n-1}.\mathrm{MtM}_n < 0$, then the above constraints apply with the collateral at $t_{n-1}$ and the position $A_{n-1}$ in $A$ resettled to zero (we switch for example from the seller to the buyer):

$$C_{n-1} := 0, \quad A_{n-1} := 0 \quad \text{if } \mathrm{MtM}_n.\mathrm{MtM}_{n-1} < 0$$

*Example* 5.1. As a simple numerical example, let us assume that at $t_n$, the collateral $C_{n-1} = 10$ ($\mathrm{MtM}_{n-1} = 10$) is uniformly distributed into $A$ and $B$: $C_{n-1} = 10 = 5 + 5$. Then, from (9), if $C_n = 8$ ($\mathrm{MtM}_n = 8$), we have only three possible configurations left for the posting: $C_n = 8 = 5 + 3 = 3 + 5 = 4 + 4$. Additionally, if $\mathrm{MtM}_n = -8$ (hence $C_n = 8$), then the seller can choose all configurations such that $8 = A_n + B_n$ with $A_n, B_n \geq 0$.

Our SCP reads finally (for the seller and buyer)

$$\mathcal{F}_\pm := \max_{(A.)\in\mathcal{A}} \mathbb{E}[F_\pm|\mathrm{MtM}_0, r_0^A, r_0^B, A_{-1}, C_{-1}, \mathrm{MtM}_{-1}] \tag{10}$$

where $C_{-1}$ denotes the previous collateral and $A_{-1}$ the previous posting in $A$ ($B_{-1} := C_{-1} - A_{-1}$).

5.1. **Optimal control.** In order to satisfy the constraints (8) and (9), we write the control $A_n \in \mathcal{A}$, without loss of generality, as

$$\begin{aligned} A_n - A_{n-1} &:= a_n \Delta C_n \, 1_{\Delta C_n \geq 0} \\ &+ \left(a_n \left(\min(C_n - A_{n-1}, 0) - \max(-A_{n-1}, \Delta C_n)\right) + \max(-A_{n-1}, \Delta C_n)\right) 1_{\Delta C_n < 0} \end{aligned} \tag{11}$$

where $(a_n)_{n=0,\cdots,N}$ is a discrete adapted process valued in $[0, 1]$. The (constrained) supremum in (10) over $A. \in \mathcal{A}$ is then replaced by an (unconstrained) supremum over $a.$. Let us remark that the constraints (8) and (9) could also be enforced by introducing Lagrange multipliers but it is more efficient to use our decomposition (11). The condition $a_n$ to be valued in $[0, 1]$ will be automatically satisfied by using below a neural network with a sigmoid activation function.

The optimal control $a_n^*$ at $t_n$, as being adapted, is a path-dependent function of the mark-to-market amounts $(\mathrm{MtM})_{k=0,\cdots,n}$, the rate $(r_k^{AB})_{k=0,\cdots,n}$ ($X_n := (\mathrm{MtM}_n, r_n^{AB})$), and possibly additional stochastic factors generating the path $(X_i)_{0\leq i\leq n}$:

$$a_n^* = a_n(X_0, \ldots, X_n, A_{-1}, C_{-1}, \mathrm{MtM}_{-1}) \tag{12}$$

In particular,

$$a_0^* = a_0(X_0, A_{-1}, C_{-1}, \mathrm{MtM}_{-1}) \tag{13}$$

Equivalently, from the Markov property (see (13)), the optimal control at $t_n$ can also be written as

$$a_n^* = a_n(X_n, A_{n-1}^*, C_{n-1}, \mathrm{MtM}_{n-1}) \tag{14}$$

and means that the control depends on the *optimal* posting $A_{n-1}^*$, the collateral $C_{n-1}$, the (sign of) mark-to-market amount $\mathrm{MtM}_{n-1}$ at $t_{n-1}$ and $X_n$. In all the above equations, the star index denotes the optimal strategy. By applying the recurrence after $n$ steps in Equation (14), we reproduce Equation (12), hence our original proposition.

5.2. **Full substitution.** In the case of full substitution, corresponding to relaxing the conditions (8-9), the optimal control can be analytically derived and corresponds to computing the maximum pathwise (i.e., inside the expectation). The solution, called Cheapest-To-Deliver, is given by

$$A_n^* = C_n 1_{r_n^{AB} \geq 0}$$

5.3. **Bang-bang strategy.** For use below, we define a sub-optimal control $\tilde{A}_n$ corresponding to a bang-bang strategy consisting in hitting the bounds in constraints (8-9) as a function of the sign of $r_n^{AB}$:

$$\begin{aligned} \tilde{A}_n &= (\tilde{A}_{n-1} + \Delta C_n)1_{\Delta C_n \geq 0} + (\tilde{A}_{n-1} + \min(C_n - \tilde{A}_{n-1}, 0))1_{\Delta C_n < 0}, \quad \text{if } r_n^{AB} \geq 0 \\ &= \tilde{A}_{n-1}1_{\Delta C_n \geq 0} + (\tilde{A}_{n-1} + \max(-\tilde{A}_{n-1}, \Delta C_n))1_{\Delta C_n < 0}, \quad \text{if } r_n^{AB} < 0 \end{aligned} \tag{15}$$

By construction, we have the trivial bounds

$$\mathcal{F}_{\pm}^{\text{bang-bang}} \leq \mathcal{F}_{\pm} \leq \mathcal{F}_{\pm}^{\text{CTD}} \tag{16}$$

5.4. **With or without threshold levels.** Without threshold levels, the collateral amount is equal to the absolute value of the mark-to-market value:

$$C_n = |\text{MtM}_n| \tag{17}$$

With threshold levels $K$ and MTA, we have instead

$$C_n = \left((|\text{MtM}_n| - K)^+ - C_{n-1}\right) 1_{|(|\text{MtM}_n| - K)^+ - C_{n-1}| \geq \text{MTA}} + C_{n-1} \tag{18}$$

where the strike $K$ corresponds to the amount that is not collateralized and MTA the minimum amount of margin. Note that in practice, we have different (seller and buyer) threshold levels for $\text{MtM}_n \geq 0$ and $\text{MtM}_n < 0$.

5.5. **Numerics.** Here $a_n^*$ is represented by a feedforward network $\mathbb{R}^4 \mapsto [0,1]$. The back-propagation applies, except that the gradient of $J$ with respect to $\theta_n$, as $a_n^*$ depends also on $\theta_{n-1}$ through $A_{n-1}^*$, is given by

$$\nabla_{\theta_n} J = \mathbb{E}[\frac{\partial a_n}{\partial \theta_n} \alpha_n \left( r_n^{AB} \delta t_n + r_{n+1}^{AB} \delta t_{n+1} \alpha_{n+1} \frac{\partial a_{n+1}}{\partial A_n} + r_{n+1}^{AB} \delta t_{n+1} \beta_{n+1} \right)]$$

with

$$\begin{aligned} \alpha_n &:= \Delta C_n \, 1_{\Delta C_n \geq 0} + (\min(C_n - A_{n-1}, 0) - \max(-A_{n-1}, \Delta C_n)) \; 1_{\Delta C_n < 0} \\ \beta_{n+1} &:= 1 + \left( a_{n+1}(-1_{C_{n+1} \leq A_n} + 1_{-A_n \geq \Delta C_{n+1}}) - 1_{-A_n \geq \Delta C_{n+1}} \right) \; 1_{\Delta C_{n+1} < 0} \end{aligned}$$

We have tested our algorithm on a synthetic market data which is explicitly given by the diffusion:

$$\text{MTM}_n = 10^3 (2e^{-\frac{\sigma_1^2}{2} t_n + \sigma_1 W_n} - e^{-\frac{\sigma_2^2}{2} t_n + \sigma_2 W_n^{\perp}})$$

with $\sigma_1 = 0.2$, $\sigma_2 = 0.4$ and $W$, $W^{\perp}$ two independent Brownian motions. This can be seen as modelling the basket of linear instruments - say assets or swaps. Similarly, $r^A$ and $r^B$ are simulated using a one-factor Ho-Lee model :

$$\begin{aligned} r_n^A &= 1\% + 2\% Z_n \\ r_n^B &= 2\% + 1\% Z_n^{\perp} \end{aligned}$$

with $Z$, $Z^{\perp}$ two independent Brownian motions. We do not include any threshold levels. Note that the MtM changes its sign in our data. We have not succeeded in using the approximation obtained in [27] for these examples. We have used here an explicit dynamics for the mark-to-market, this allows the reader to benchmark his own implementation of our algorithm. In all numerical examples, $t_N$ =1 year and $\text{MTM}_n$ are computed every 10 days. This requires to use 36 neural networks. One can try to use less neural networks, this corresponds to use a piece-wise constant approximation of the control - say every 20 days. This approximation is known to have a very slow convergence rate [**?**] and therefore needs to be check on a case-by case basis.

Note that, our numerical result $\mathcal{F}_{\pm}$ is, as expected (see Equation 16), bounded by the bang-bang and cheapest-to-deliver solutions. If the algorithm gives a result below the bang-bang strategy,

| | $\mathcal{F}_+$ | $\mathcal{F}_-$ |
|---|---|---|
| Full sub. | **0.1** | **22.7** |
| Bang-bang | **0.1** | **20.2** |

TABLE 2. $\mathcal{F}_+$ and $\mathcal{F}_-$.

| | $\mathcal{F}_+$ | $\mathcal{F}_-$ |
|---|---|---|
| SRU | **0.1** | **20.5** |
| SRU(+PSO) | 0.1 | 20.5 |
| SRU(+SA) | 0.1 | 19.8 |

TABLE 3. $\mathcal{F}_+$ and $\mathcal{F}_-$ using recurrent NNs.

this seems that our choice of hidden layers for our NN is not flexible enough and this needs to be changed.

Despite neural networks can approximate with arbitrary precision continuous functions, due to the presence of multiple local maximizers in our maximization problem, it could be useful to decrease the complexity of the neural network. This implies that choosing a good parameterization is a crucial issue. This was illustrated in our previous numerical examples (see Parameterizations **I** and **II**). In the next section, we explain another more involved parameterization, which consists in parameterizing the $Z$-process, which appears in the BSDE associated to the SCP, by a neural network.

## 6. Solving SCP with neural networks and BSDE

When $X^a_\cdot$ is a continuous Markov process controlled by $a$, the function $u$, as defined by (10), is a (viscosity) solution of a fully nonlinear PDEs of the form

$$\partial_t u + \mathcal{L}u + f(t,x,u,\nabla u,\nabla^2 u) = 0, \quad u(T,x) = g(x), \quad x \in \mathbb{R}^d \tag{19}$$

where $\mathcal{L} := b.\nabla + \frac{1}{2}\sigma\sigma^\dagger\nabla^2$ is the infinitesimal generator of the Itô process $X \in \mathbb{R}^d$:

$$dX_t = b(t,X_t)dt + \sigma(t,X_t).dW_t \tag{20}$$

### 6.1. Semilinear PDE and BSDE.

As a simple example, let us consider the semilinear PDE

$$\partial_t u + \mathcal{L}u + f(u) = 0, \quad u(T,x) = g(x), \quad x \in \mathbb{R}^d \tag{21}$$

where $f$ is a non-linear function, assumed to be Lipschitz. We focus on the class (21) for the sake of simplicity. Following closely the description of our algorithm, the extension to the general case is then straightforward. PDE (21) can be represented probabilistically through the use of a BSDE. This is defined by using two adapted processes $(Y_t, Z_t)$ (with respect to the filtration $\sigma((W_s)_{s\in[0,t]})$) satisfying

$$dY_t = -f(Y_t)dt + Z_t\sigma(t,X_t).dW_t \tag{22}$$

supplemented with the terminal condition $Y_T = g(X_T)$ (instead of an initial condition as for the SDE (20)). The link between the BSDE and the PDE is given by the relation

$$Y_t = u(t,X_t), \quad Z_t = \nabla_x u(t,X_t) \tag{23}$$

which can be easily checked by applying Itô's formula on $Y_t := u(t, X_t)$, assuming that the solution $u \in C^{1,2}(\mathbb{R}_+ \times \mathbb{R}^d)$ is smooth enough:

$$\begin{aligned} dY_t &= (\partial_t + \mathcal{L})u(t, X_t)dt + \nabla_x u(t, X_t)\sigma(t, X_t).dW_t \\ &= -f(Y_t)dt + Z_t\sigma(t, X_t).dW_t \end{aligned}$$

where we have used in the last equation that $u$ satisfies PDE (21). Note that by integration from $t = 0$ to $t = T$, the BSDE can be written as

$$g(X_T) = Y_0 - \int_0^T f(Y_t)dt + \int_0^T Z_t\sigma(t, X_t).dW_t \tag{24}$$

In the case $f = 0$, this means that the claim $g(X_T)$ can be replicated for price $Y_0$ with a delta-hedging strategy $Z_t$.

6.2. **Deep BSDE solver.** Because of the $1-1$ unique correspondence (23), solving our semilinear PDE (21) is therefore equivalent to solving BSDE (22). The adapted process $Z_t$, which can be interpreted as the delta strategy for a claim delivering $g(X_T)$ at a maturity $T$ when $f := 0$ (see Equation (24)), is needed in order to match the terminal condition $Y_T = g(X_T)$. In particular, $Y_0$ can be seen as the solution of the minimization problem:

**Proposition 6.1.**

$$Y_0^* := \text{argmin}_{\hat{Y}_0} \min_{z_t} \mathbb{E}[(g(X_T) - Y_T^z)^2] \tag{25}$$

*with*

$$dY_t^z = -f(Y_t^z)dt + z_t\sigma(t, X_t).dW_t, \quad Y_0^z = \hat{Y}_0$$

*and where the minimum over $z_t$ is performed over the set of adapted Markovian processes $z_t := z(t, X_t)$. Furthermore, this solution is unique if $f$ is assumed to be Lipschitz.*

*Proof.*

**(1)**: If we set $z_t = \nabla_x u(t, X_t)$ and $\hat{Y}_0 = u(0, X_0)$, the $\mathrm{L}^2$–error vanishes.

**(2)** - uniqueness: By assuming that we have 2 solutions $(Y_0^1, z_\cdot^1)$ and $(Y_0^2, z_\cdot^2)$, we have from Equation (24):

$$\begin{aligned} g(X_T) &= Y_0^1 - \int_0^T f(Y_t^1)dt + \int_0^T z_t^1\sigma(t, X_t).dW_t \\ g(X_T) &= Y_0^2 - \int_0^T f(Y_t^2)dt + \int_0^T z_t^2\sigma(t, X_t).dW_t \end{aligned}$$

By subtracting these two equations, we get

$$(Y_0^2 - Y_0^1) - \int_0^T (f(Y_t^2) - f(Y_t^1))dt + \int_0^T (z_t^2 - z_t^1)\sigma(t, X_t).dW_t = 0$$

which implies that $z_t^1 = z_t^2$ almost surely for all $t \in [0, T]$ and therefore

$$(Y_0^2 - Y_0^1) = \int_0^T (f(Y_t^2) - f(Y_t^1))dt$$

As $f$ is Lipschitz, this gives

$$|Y_0^2 - Y_0^1| \leq \int_0^T ||f||_{\text{Lip}}\mathbb{E}[|Y_t^2 - Y_t^1|]dt$$

from which we conclude from Gronwall's inequality. □

Problem (25) can then be discretised by an Euler-like scheme as

$$Y_0^{\Delta,*} := \text{argmin}_{\hat{Y}_0} \min_{(z_i(\cdot))_{0\le i\le n-1}} \mathbb{E}[(g(X_{t_n}^\Delta) - Y_{t_n}^\Delta)^2]$$

where $0 := t_0 < t_1 < \cdots < t_n := T$ and

$$Y_{t_{i+1}}^\Delta - Y_{t_i}^\Delta = -f(Y_{t_i}^\Delta)\Delta + z_i(X_{t_i}^\Delta)\sigma(t_i, X_{t_i}^\Delta).\Delta W_i, \quad \Delta W_i := W_{t_{i+1}} - W_{t_i}, \quad Y_0^\Delta = \hat{Y}_0$$

In particular, we have $\lim_{\Delta\to 0} Y^{\Delta,*} = Y_0^* = u(0, X_0)$. The minimization over the functions $(z_i(X_i^\Delta))_{i=1,\cdots,n-1}$ (and $z_0 \in \mathbb{R}$) is then achieved by choosing a "suitable" parametric form obtained with a neural network. The resulting algorithm is called deep BSDE solver in [30] (see also [3, 12, 13]) and can be described by the following steps.

**Deep BSDE solver: recipe** [30]**.**

(1) Simulate $M$ Monte-Carlo paths $(X_{t_1}^m, X_{t_2}^m, \cdots, X_{t_n}^m)_{1\le m\le M}$ at the discretization dates $0 := t_0 < t_1 < \cdots < t_n := T$.
(2) Parameterize the functions $(z_i)_{1\le i\le n-1}$ with $n-1$ neural networks depending on some weights $\theta_i := (W_l^i, b_l^i)_{1\le l\le N_i}$:
$$z_i = a_{\theta_i}(X_{t_i}), \quad i = 1, \cdots, n-1$$
(3) Minimize over $(\theta_i)_{1\le i\le n-1}$, $\hat{Y}_0$ and $z_0$ with a SGD:

$$J(\theta) := \frac{1}{2M}\sum_{m=1}^M (Y_{t_n}^m - g(X_{t_n}^m))^2 \tag{26}$$

where

$$Y_{t_{i+1}}^m - Y_{t_i}^m = -f(Y_{t_i}^m)\Delta + a_{\theta_i}(X_{t_i}^m)\sigma(t_i, X_{t_i}^m).\Delta W_i^m, \quad Y_0^m = \hat{Y}_0$$

Note that the pathwise gradient's flow with respect to $(\theta_i)_{1\le i\le n-1}$, $\hat{Y}_0$ and $z_0$ can be obtained by automatic differentiation through the formulas:

$$\begin{aligned}
\frac{\partial Y_{t_n}^m}{\partial \theta_{n-1}} &= \frac{\partial z_{n-1}}{\partial \theta_{n-1}}\sigma(t_{n-1}, X_{t_{n-1}}^m).\Delta W_{n-1}^m \\
\frac{\partial Y_{t_n}^m}{\partial \theta_{n-2}} &= (1 - f'(Y_{t_{n-1}}^m)\Delta)\frac{\partial z_{n-2}}{\partial \theta_{n-2}}\sigma(t_{n-2}, X_{t_{n-2}}^m).\Delta W_{n-2}^m \\
\frac{\partial Y_{t_n}^m}{\partial \theta_{n-3}} &= (1 - f'(Y_{t_{n-1}}^m)\Delta)(1 - f'(Y_{t_{n-2}}^m)\Delta)\frac{\partial z_{n-3}}{\partial \theta_{n-3}}\sigma(t_{n-3}, X_{t_{n-3}}^m).\Delta W_{n-3}^m, \quad \cdots \\
\frac{\partial Y_{t_n}^m}{\partial z_0} &= \sigma(t_0, X_0)\Delta W_0 \prod_{i=2}^n (1 - f'(Y_{t_{i-1}}^m)\Delta), \quad \frac{\partial Y_{t_n}^m}{\partial \hat{Y}_0} = \prod_{i=1}^n (1 - f'(Y_{t_{i-1}}^m)\Delta)
\end{aligned}$$

where the gradients $\frac{\partial z_i}{\partial \theta_i}$ are given by the backward propagation of the $(i)$ neural network (see Equations (1)). The (online) SGD of the functional $J(\theta)$ (26) with respect to the weights denoted generically $(\theta_i)_{1\le i\le n+1}$ (here $\theta_n := \hat{Y}_0$ and $\theta_{n+1} := z_0$) is then given by

$$\theta_i \mapsto \theta_i - \eta_t(Y_{t_n}^I - g(X_{t_n}^I))\frac{\partial Y_{t_n}^I}{\partial \theta_i}$$

at a step $t$ with $I$ a uniform random variable in $[[1, M]]$. Note that in practice, we introduce two discretization schedules with timesteps $\Delta_{\text{Euler}} \le \Delta_{\text{Z}}$: $Y_{t_i}^m$ is simulated on a timestep $\Delta_{\text{Euler}}$ and the process $z_t$ is assumed to be piecewise constant on a timestep $\Delta_{\text{Z}}$. This allows to reduce the dimensionality of the optimization problem.

The deep BSDE solver has been implemented in [30] using `Python` machine learning package `tensorflow`. A quick inspection of the above algorithm shows that this is not necessary and everything could be quickly coded from scratch with a proper compiled language as `C++`.

*Some important remarks.*

**(i)** Machine learning miracle: There is nothing fancy in using a neural network. This is nothing else than a standard parameterization. One interesting aspect is the ability of computing quickly the function and its gradients.

**(ii)** The online SGD can be improved using a more efficient method: let us cite Polyak-Ruppert averaging which consists in using $\theta^* := \frac{1}{T}\sum_{t=0}^{T}\theta_t$ as the minimizer after $T$ iterations, Adam optimizer, mini-batches, etc... (see [2]). Additionally, in some cases when the BSDE can be solved exactly with $f := 0$, one can use this closed-form formula as an efficient guess in the optimization (see [9]) for the choice of $\hat{Y}_0$ and $z_0$ and also decompose $z_{t_i} := z_{t_i}^{f=0} + a_{\theta_i}(X_{t_i})$.

**(iii)** Despite these various acceleration techniques for the optimization, since the $\mathrm{L}^2$–error (as given by $J(\theta)$) is not a convex function of the weights $(\theta_i)_{1\leq i\leq n-1}$, $\hat{Y}_0$ and $z_0$, the optimization will not usually converge towards zero but towards a local minimizer. Let us denote then $Y_0^M$ the estimate of $Y_0$ obtained using $n-1$ neural networks with $M$ hidden layers. For the sake of simplicity of our notation, we skip here the dependencies with respect to the timesteps $\Delta_{\mathrm{Euler}}$ and $\Delta_{\mathrm{Z}}$. Firstly, we do not know if $Y_0^M$ is above or below $Y_0$. Secondly, if we increase the number of hidden layers $M' > M$ and the $\mathrm{L}^2$–error decreases, it is not clear if the new estimate $Y_0^{M'}$ will be closer to $Y_0$ than $Y_0^M$. In conclusion, we can assess the accuracy of our estimate only in the case when the $\mathrm{L}^2$–error vanishes, a situation difficult to achieve from the non-convexity of the optimization problem.

In order to circumvent this difficulty, we explain below how to complement the deep BSDE solver with the computation of lower and upper bounds, that can be proved to be optimal when the $\mathrm{L}^2$–error vanishes.

6.3. **Deep primal algorithm.** We take $f$ convex and note that the solution of the PDE (21) can be represented as a stochastic control problem (more precisely non-linear PDEs originate from Hamilton-Jacobi-Bellman equations in Finance):

$$u(t,x) = \sup_{a_\cdot \in A} \mathbb{E}_{t,x}\left[e^{\int_t^T a_s ds} g(X_T) - \int_t^T e^{\int_t^s a_u du} f^*(a_s) ds\right] \tag{27}$$

where $f(u) := \sup_{a\in A}\{au - f^*(a)\}$ (and $f^*(a) = \sup_u\{au - f(u)\}$) and the supremum is performed over all adapted control $a_t$ valued in a set $A$. Indeed, $u$ is the solution of the following HJB equation:

$$\partial_t u + \mathcal{L}u + \sup_{a\in A}\{au - f^*(a)\} = 0, \quad u(T,x) = g(x)$$

By definition of the Legendre transform $f^*$, we recover our original PDE (21).

*An (optimal) lower bound.* By definition of the stochastic control problem (27), from arbitrary choice of a control $\tilde{a}_\cdot \in A$, not necessarily optimal, we obtain a lower bound by construction:

$$\mathbb{E}_{t,x}\left[e^{\int_t^T \tilde{a}_s ds} g(X_T) - \int_t^T e^{\int_t^s \tilde{a}_u du} f^*(\tilde{a}_s) ds\right] \leq u(t,x) \tag{28}$$

This lower bound is optimal (and equal to $u(t,x)$) by taking $\tilde{a}_t = a_t^*$ with

$$a_t^* := \mathrm{argmax}\{aY_t - f^*(a)\}$$

and $Y_t$ solution of the BSDE (22). By relying on Equation (28), we have derived a lower bound for the semilinear PDE (21).

**Deep primal algorithm: recipe.**

(1)-(2)-(3) Perform the deep BSDE solver. We note $Y_0^*$, $z_0^*$ and $(z_i^* := a_{\theta_i^*}(X_{t_i}))_{1\leq i\leq n-1}$ the (local) optimizer obtained using our (refined) SGD.

(4) Simulate $M$ independent Monte-Carlo paths $(X_{t_1}^m, X_{t_2}^m, \cdots X_{t_n}^m)_{1\leq m\leq M}$. For each path $(m)$, compute $Y_{t_i}^m$ using the Euler scheme:

$$Y_{t_{i+1}}^m - Y_{t_i}^m = -f(Y_{t_i}^m)\Delta + a_{\theta_i^*}(X_{t_i}^m)\sigma(t_i, X_{t_i}^m).\Delta W_i^m, \quad Y_0^m = Y_0^*$$

and simulate the processes $dI_t := a_t^* dt$ and $dQ_t := e^{\int_0^t a_u^* du} f^*(a_t^*)dt$ by

$$\begin{aligned} I_{t_{i+1}}^m &= I_{t_i}^m + a_{t_i}^{m,*}\Delta, \quad I_0^m := 0 \\ Q_{t_{i+1}}^m &= Q_{t_i}^m + \exp(I_{t_i}^m) f^*(a_{t_i}^{m,*}), \quad Q_0^m := 0 \end{aligned}$$

where

$$a_{t_i}^{m,*} := \text{argmax}\{aY_{t_i}^m - f^*(a)\}$$

(5) Average

$$Y_0^{\text{Lower}} := \frac{1}{M}\sum_{m=1}^{M}\left(\exp(I_{t_n}^m)g(X_{t_n}^m) - Q_{t_n}^m\right)$$

From Equation (28), we have when $M$ is large:

$$Y_0^{\text{Lower}} \leq u(0, X_0)$$

Note that as long as $Y_0^{\text{Lower}}$ increases when we increase the number of hidden layers, one can assert for sure that we come closer to the true solution $u(0, X_0)$, even if the $\text{L}^2$–error is still large or has even increased.

In the next section, we explain how to compute an (optimal) upper bound.

## 7. Deep dual algorithm

7.1. **Interlude - American options.** The arbitrage-free value of an American option with fixed horizon $T$ for a complete model is given by

$$Y_0 := \sup_{\tau\in[0,T]} \mathbb{E}[g(X_\tau)]$$

where the supremum is taken over the space of stopping time valued in $[0, T]$. An approximation for the boundary region is obtained by the Longstaff-Schwartz algorithm [26]. By performing a second-independent Monte-Carlo using this (sub)-optimal stopping time $\tau^{\text{LS}}$, we obtain a lower bound

$$\mathbb{E}[g(X_{\tau^{\text{LS}}})] \leq Y_0 \tag{29}$$

The pricing of American options would not be possible in practice if this lower bound (possibly optimal) could not be complemented with an upper bound. This was obtained in [28] and it is given by

$$Y_0 = \inf_{M\in\mathcal{M}_0} \mathbb{E}[\sup_{t\in[0,T]}\{g(X_t) - M_t\}]$$

where $\mathcal{M}_0$ is the space of continuous martingales starting at zero at $t = 0$. The optimal martingale is given by the martingale component in the Doob-Meyer decomposition of the Snell envelope. If we choose an arbitrary martingale $M_t^* \in \mathcal{M}_0$, we obtain an upper bound by construction:

$$Y_0 \leq \mathbb{E}[\sup_{t\in[0,T]} \{g(X_t) - M_t^*\}] \tag{30}$$

*Dual formula for stochastic control.* In [14], we have explained how to obtain an upper bound (possibly optimal) for a general control stochastic problem. In the simpler case of Equation (10), this reads

$$u(t,x) = \inf_{Z(\cdot,\cdot)} \mathbb{E}_{t,x}[\sup_a F_{t,T}^Z] \tag{31}$$

where

$$F_{t,T} := e^{-\int_t^T a_s ds} g(X_T) + \int_t^T e^{-\int_t^s a_u du} f^*(a_s) ds - \int_t^T e^{-\int_t^s a_u du} Z(s, X_s)\sigma(s, X_s).dW_s$$

Note that the supremum over $a$ in formula (31) is now inside the expectation and corresponds performing a pathwise maximization over all controls valued in the set $A$. The proof of (31) is simple (despite technical details - see [14] and the general formula) and we quickly report its proof:

*Proof.* **(i)** As $M_{t,T}^Z := \int_t^T e^{-\int_t^s a_u du} Z(s, X_s)\sigma(s, X_s).dW_s$ has zero mean, we have

$$\begin{aligned}
u(t,x) &:= \sup_{a.\in A} \mathbb{E}_{t,x}[e^{\int_t^T a_s ds} g(X_T) - \int_t^T e^{\int_t^s a_u du} f^*(a_s) ds], \quad \text{see Equation (10)} \\
&= \sup_{a.\in A} \mathbb{E}_{t,x}[e^{\int_t^T a_s ds} g(X_T) - \int_t^T e^{\int_t^s a_u du} f^*(a_s) ds - M_{t,T}^Z] \\
&\leq \mathbb{E}_{t,x}[\sup_{a.\in A} \{e^{\int_t^T a_s ds} g(X_T) - \int_t^T e^{\int_t^s a_u du} f^*(a_s) ds - M_{t,T}^Z\}] := \mathbb{E}_{t,x}[\sup_{a.\in A} F_{t,T}^Z] \\
&\leq \inf_{Z(\cdot,\cdot)} \mathbb{E}_{t,x}[\sup_{a.\in A} F_{t,T}^Z]
\end{aligned}$$

**(ii)** Let us take $Z^*(t,x) := \nabla_x u(t,x)$. Then,

$$\begin{aligned}
&\inf_{Z(\cdot,\cdot)} \mathbb{E}_{t,x}[\sup_{a.\in A} F_{t,T}^Z] \leq \mathbb{E}_{t,x}[\sup_{a.\in A} \{e^{\int_t^T a_s ds} g(X_T) - \int_t^T e^{\int_t^s a_u du} f^*(a_s) ds \\
- &\int_t^T e^{-\int_t^s a_u du} \nabla_x u(s, X_s)\sigma(s, X_s).dW_s\}] \\
= \; & u(t,x) + \mathbb{E}_{t,x}[\sup_{a.\in A} \{\int_t^T ds e^{\int_t^s a_u du} \left(-f^*(a_s) + a_s u(s, X_s) + (\partial_s + \mathcal{L}u)\, u(s, X_s)\right)\}] \\
= \; & u(t,x) + \mathbb{E}_{t,x}[\sup_{a.\in A} \{\int_t^T ds e^{\int_t^s a_u du} \left(-f^*(a_s) + a_s u(s, X_s) - f(u(s, X_s))\right)\}] \\
\leq \; & u(t,x)
\end{aligned}$$

where we have used Itô's lemma on $e^{-\int_t^s a_u du} u(s, X_s)$ for the second inequality and the PDE (21) in the third line. This gives the reverse inequality and we conclude our proof. □

7.2. **From HJB to random pathwise HJ.** Equation (31) can be simplified by solving the *random pathwise HJ* equation, corresponding to $U(t) := \sup_{a_\cdot \in A} F_{t,T}$. This means that the pathwise supremum over $a$ can be exactly computed. We write $F_{t,T}$ as

$$F_{t,T} = e^{-\int_t^T a_s ds} g(X(T)) + \int_t^T ds e^{-\int_t^s a_u du} \{ -f^*(a_s) - Z(s,X(s))\sigma(s,X(s)) \frac{dW(s)}{ds} + \frac{1}{2} \nabla Z(s,X(s)) \sigma^2(s,X(s)) \}$$

Here we have used a Wong-Zakai approximation for the Brownian motion in order to justify the derivative $\frac{dW(t)}{dt}$ and this corresponds using Stratonovich definition of the stochastic integral, adding an additional term $\frac{1}{2}\nabla Z(t,x)\sigma^2(t,X)$. This gives

$$u(0,X_0) = \inf_{Z(\cdot,\cdot)} \mathbb{E}[U(0)]$$

where $U(t)$ is solution of the random Hamilton-Jacobi:

$$\frac{dU(t)}{dt} + \sup_{a\in A}\{aU(t) - f^*(a)\} - Z(t,X(t))\sigma(t,X(t)).\frac{dW(t)}{dt} + \frac{1}{2}\nabla Z(t,X(t))\sigma^2(t,X(t)) = 0,$$
$$U(T) = g(X(T))$$

Taking the supremum over $a$, we get finally:

$$\frac{dU(t)}{dt} + f(U(t)) - Z(t,x)\sigma(t,X(t)).\frac{dW(t)}{dt} + \frac{1}{2}\nabla Z(t,X(t))\sigma^2(t,X(t)) = 0, \quad U(T) = g(X(T))$$

7.3. **Discretization.** This problem can be discretized and we obtain

$$u^\Delta(0,X_0) = \inf_{(z_i)_{1\le i\le n-1}} \mathbb{E}[U_0^\Delta]$$

with

$$U^\Delta_{t_{i-1}} = U^\Delta_{t_i} + f(U_{t_i})\Delta - z_{i-1}(X_{t_{i-1}})\sigma(t_{i-1},X_{t_{i-1}}).\Delta W_{i-1}, \quad U_{t_n} := g(X_{t_n})$$

With our choice of the discretization of the stochastic integral $\int Z(t,X_t)\sigma(t,X_t)dW_t$ (Itô and not Stratonovich convention), the term $\frac{1}{2}\nabla Z(t,X(t))\sigma^2(t,X(t))$ disappears. Finally, this leads to our dual algorithm:

**Dual-algorithm: recipe.**

(1-5) Perform the primal algorithm.

(6) Minimize over $(\theta_i)_{1\le i\le n-1}$ the functional with a SGD:

$$\frac{1}{M}\sum_{m=1}^M U_0^m$$

where $z_{i-1} := a_{\theta_{i-1}}$,

$U^m_{t_{i-1}} = U^m_{t_i} + f(U^m_{t_i})\Delta - a_{\theta_{i-1}}(X^m_{t_{i-1}})\sigma(t_{i-1},X^m_{t_{i-1}}).\Delta W^m_{i-1}, \quad U^m_{t_n} = g(X^m_{t_n})$

From step (3), we have obtained $Y_0^*$, $z_0^*$ and $(\theta_i^*)_{1\le i\le n-1}$ that can be used as a (good) guess for the optimization in step (6).

Remark that $z_0$ does not appear in the minimization (as $\mathbb{E}[z_0\sigma(0,X_0)\Delta W_0] = 0$). Note that the gradient with respect to $(\theta_i)$ are given by

$$\frac{\partial U_0^m}{\partial \theta_i} = \prod_{k=1}^i (1 + f'(U^m_{t_k})\Delta) \left( -\frac{\partial a_{\theta_i}(X^m_{t_i})}{\partial \theta_i}\sigma(t_i,X^m_{t_i}).\Delta W_i^m \right), \quad i = 1,\cdots,n-1$$

(7) Once the optimizer has converged to a (local) minimizer $(\theta_i^*)_{1\leq i\leq n-1}$, simulate $M$ independent MC paths and compute

$$Y_0^{\text{Upper}} := \frac{1}{M}\sum_{m=1}^{M} U_0^{m,*}$$

with

$$U_{t_{i-1}}^{m,*} = U_{t_i}^{m,*} + f(U_{t_i}^{m,*})\Delta - a_{\theta_{i-1}^*}(X_{t_{i-1}}^m)\sigma(t_{i-1}, X_{t_{i-1}}^m).\Delta W_{i-1}^m, \quad U_{t_n}^m = g(X_{t_n}^m) \tag{32}$$

By construction, we have when $M$ is large:

$$\boxed{Y_0^{\text{Lower}} \leq Y_0 \leq Y_0^{\text{Upper}}}$$

and these last inequalities summarize the core of our deep primal-dual algorithm. Note that the numerical complexity of the primal and dual bounds are similar.

## 8. Numerical examples

8.1. **CVA-like PDE.** We consider the semilinear PDE (21) with

$$f(u) = \beta u^+$$

Here on the stochastic control side, $A := [0, \beta]$ and $f^*(a) := 0$. By setting $u^{\text{CVA}}(t,x) := e^{-\beta(T-t)}u(t,x)$, we have that $u^{\text{CVA}}$ is the solution of PDE

$$\partial_t u^{\text{CVA}} + \mathcal{L}u^{\text{CVA}} + \beta((u^{\text{CVA}})^+ - u^{\text{CVA}}) = 0, \quad u^{\text{CVA}}(T,x) = g(x), \quad x \in \mathbb{R}^d$$

This PDE pops up naturally when one considers the pricing of CVA and $\beta$ coincides with the intensity of default of a counterparty (see e.g. [15]). Below, we have used our primal-dual algorithm for deriving lower and upper estimate of $u^{\text{CVA}}(0, X_0)$ with $\beta = 0.03$.

In all our numerical experiments, for each neural network, we have used 2 hidden layers of dimension $d + 2$ as in [30] and the two timesteps are $\Delta t_{\text{Z}} = 1/20$ and $\Delta t_{\text{Euler}} = 1/100$. The number of Monte-Carlo paths is fixed to $M = 2^{17}$. Furthermore, $(X_t^i)_{1\leq i\leq d}$ is a $d$-dimensional uncorrelated Black-Scholes model with a constant volatility $\sigma := 0.2$:

$$\frac{dX_t^i}{X_t^i} = \sigma dW_t^i, \quad d\langle W^i, W^j\rangle_t = \delta_{ij}dt, \quad X_0^i := 1$$

Following Remark (ii) (see [9]), we have decomposed $z_{t_i}$ as

$$z_{t_i} = a_{\theta_i}(X_{t_i}) + z_{t_i}^{\text{BS}}(X_{t_i})$$

where $z_{t_i}^{\text{BS}}$ corresponds to the Black-Scholes delta at $t_i$.

First, one focuses on the case $d = 1$ as one can compare our results (quoted in percent) with the exact solutions obtained using a PDE solver (see Table 4). For comparison, we have also included the Black-Scholes prices corresponding to $\beta = 0$. Then, for $T = 1$ year, we compute the lower and upper bounds as a function of $d$ (see Table 5). The (non-smooth) payoff is $g(x_1, \cdots, x_d) = \sum_{i=1}^d (1 - 21_{x_i>1})$.

| $T$ (years) | Lower bound | Upper bound | Exact(PDE) | BS($\beta = 0$) |
|---|---|---|---|---|
| 2 | 12.40 | 12.46 | 12.40 | 11.24 |
| 4 | 17.89 | 18.05 | 17.89 | 15.85 |
| 6 | 22.12 | 22.28 | 22.12 | 19.35 |

Table 4. CVA: $\beta = 0.03$, $\sigma = 0.2$ and $d = 1$. $g(x) = 1 - 21_{x>1}$.

| $d$ (number of assets) | Lower bound | Upper bound | BS($\beta=0$) |
|---|---|---|---|
| 2 | 16.65 | 16.64 | 15.91 |
| 3 | 24.75 | 24.97 | 23.87 |
| 4 | 32.79 | 32.81 | 31.82 |
| 5 | 40.83 | 40.96 | 39.78 |
| 6 | 48.80 | 48.83 | 47.73 |

TABLE 5. CVA: $\beta=0.03$, $\sigma=0.2$. $g(x_1,\cdots,x_d)=\sum_{i=1}^d(1-21_{x_i>1})$. $T=1$ year.

One can observe that the lower and upper bounds are tight and closed to the exact solutions for $d=1$. The lower bound depends only on the sign of $(Y_{t_i})_{0\leq i\leq n-1}$ and therefore it is weakly dependent on the estimation of the gradient $(z_{t_i})_{1\leq i\leq n-1}$. Our results are therefore very good even when using a small number of hidden layers (here 2). The upper bound depends more on the gradient $(z_{t_i})_{1\leq i\leq n-1}$ via $(U_{t_i})_{1\leq i\leq n-1}$ (see Equation (32)). As a consequence, the numerical upper bounds depend more on our choice of the number of hidden layers (and also the efficiency of our SGD). This can be easily asserted by looking at the distance between the lower and upper bounds. We should emphasize again that even if our SGD has not converged and the $\mathrm{L}^2$–error is still large, our algorithm produces robust lower and upper bounds (this is why we have deliberately chosen a small number of hidden layers).

8.2. **IM-like PDE.** As a next example, we consider the semilinear PDE:

$$\partial_t u+\frac{1}{2}\sigma^2\sum_{i=1}^d x_i^2\partial^2_{x_i}u+f(x,\nabla u)=0,\quad f(x,\nabla u):=\beta\sqrt{\sum_{i=1}^n(x_i\partial_{x_i}u)^2} \tag{33}$$

The nonlinearity $f(\nabla u)$ can be interpreted as the cost of initial margin computed as the $\alpha:=99\%$-quantile of our portfolio value over a period $\Delta$ (typically 10 days): $\alpha_{\mathrm{IM}}$ such that

$$\mathbb{P}[u_{t+\Delta}-u_t>\alpha_{\mathrm{IM}}]=1-\alpha$$

Using the approximation $u_{t+\Delta}-u_t\approx\sigma(x\nabla_x u).\Delta W$, we get

$$\alpha_{\mathrm{IM}}=\sigma\sqrt{\Delta}N^{-1}(\alpha)f(x,\nabla u)$$

with $N^{-1}$ the inverse cumulative distribution of a standard Gaussian.

**Primal.** As $f$ is convex in $\nabla u$, the associated stochastic control problem is:

$$u(t,x)=\sup_{b.\in S_d^\beta}\mathbb{E}_{t,x}\left[g(X_T^b)\right],\quad dX_t^{b,i}=b_t^iX_t^{b,i}dt+\sigma X_t^{b,i}dW_t^i,\quad d\langle W^i,W^j\rangle_t=\delta_{ij}dt$$

where the supremum is taken over all adapted control valued in $S_d^\beta$, the $d$-dimensional sphere of radius $\beta$. Indeed, the HJB PDE reads

$$\partial_t u+\frac{1}{2}\sigma^2\sum_{i=1}^d x_i^2\partial^2_{x_i}u+\sup_{b\in S_d^\beta}\{b.(x\nabla u)\}=0$$

Taking the supremum over $b\in S_d$, we get PDE (33) (from the identity $\sup_{b\,:\,||b||_2^2=\beta^2}\{b.p\}=\beta||p||_2$). The optimal control is

$$b_t^{i,*}=\beta\frac{X_t^i\partial_{x_i}u(t,X_t)}{\sqrt{\sum_{i=1}^d(X_t^i\partial_{x_i}u)^2}},\quad i=1,\cdots,d$$

For completeness, we give the pathwise gradient flow:

$$\begin{aligned}\frac{\partial Y_{t_n}}{\partial \theta_i} &= (\sigma X_{t_i}.\Delta W_i - \nabla_z f(X_{t_i}, z_{t_i})\Delta)\,\frac{\partial z_{\mathrm{t_i}}}{\partial \theta_i}\\ \frac{\partial Y_{t_n}}{\partial z_0} &= (\sigma X_0.\Delta W_0 - \nabla_z f(X_0, z_0)\Delta)\,, \quad \frac{\partial Y_{t_n}}{\partial \hat{Y}_0} = 1\end{aligned}$$

**Dual.** Furthermore, the dual expression is

$$u(t,x) = \inf_{Z(\cdot,\cdot)} \mathbb{E}_{t,x}\left[\sup_{b\in S_d^\beta}\{g(X_T^b) - \int_0^T Z(t, X_t^b)\sigma X_t^b.dW_t\}\right]$$

Here, in comparison with the dual expression for CVA-like PDE, the supremum over the pathwise $b \in S_d^\beta$ is not known in closed-form and therefore should be computed numerically in our dual algorithm. Step (6) in "Dual-algorithm: recipe" is then modified by

(6) Minimize over $(\theta_i)_{1\le i\le n-1}$ the functional

$$\frac{1}{M}\sum_{m=1}^{M} \sup_{(b_i)_{1\le i\le n-1}\in S_d^\beta} J^m(\theta, b)$$

with

$$J^m(\theta, b) := \{g(X_T^{b,m}) - \sum_{i=1}^{n-1} z_{\theta_i}(X_{t_i}^{b,m})\sigma X_{t_i}^{b,m}.\Delta W_i^m\}$$

The drift $b_t \in S_d^\beta$ has been chosen to be piecewise constant on the intervals $0 < t_1 < \cdots < t_n$. Once the supremum over $(b_i)_{1\le i\le n-1}$ is computed (using an optimization solver with simple quadratic constraints $\sum_{k=1}^d (b_i^k)^2 := \beta^2$ for all $1 \le i \le n-1$), the function reads $J^m(\theta, b^*(\theta))$ with $b^*(\theta)$ the optimizer. This implies that the gradient with respect to $\theta$ can be obtained without recomputing the optimal $b^*(\theta)$ as

$$\begin{aligned}\frac{d}{d\theta_i}J^m(\theta, b^*(\theta)) &= \partial_{\theta_i} J^m(\theta, b^*(\theta)) + \partial_b J^m(\theta, b^*(\theta))\frac{db^*(\theta)}{d\theta_i}\\ &= \partial_{\theta_i} J^m(\theta, b^*(\theta)) \quad \text{as} \quad \partial_b J^m(\theta, b^*(\theta)) = 0\\ &= -\frac{\partial z_{\theta_i}(X_{t_i}^{b^*,m})}{\partial \theta_i}\sigma X_{t_i}^{b^*,m}.\Delta W_i^m\end{aligned}$$

We perform the same experiment (see Table 6) as in Table 5. Note that as we need to compute a pathwise supremum over $b$ (in step (6)), the numerical complexity of the dual bound is here more involved than the primal bound.

| $d$ (number of assets) | Lower bound | Upper bound | BS($\beta = 0$) |
|---|---|---|---|
| 2 | 17.16 | 17.28 | 15.91 |
| 3 | 25.44 | 27.72 | 23.87 |
| 4 | 33.69 | 36.63 | 31.82 |
| 5 | 41.92 | 46.12 | 39.78 |
| 6 | 50.29 | 55.75 | 47.73 |

TABLE 6. IM: $\beta = 0.01\sigma$, $\sigma = 0.2$. $g(x_1, \cdots, x_d) = \sum_{i=1}^d (1 - 21_{x_i>1})$. $T = 1$ year.

Here our results are less efficient than in the case of CVA, in particular when $d$ increases. This is normal as the nonlinearity for IM-like PDEs involves explicitly the gradient $\nabla u$. As a consequence, the numerical lower and upper bounds depend more on our choice of the number of hidden layers (and also the efficiency of our SGD). In all experiments, Adam optimizer and mini-batches are not particularly more efficient than traditional online SGD.

As an illustration, we have plotted the upper and lower bounds for $d = 2$ as a function of the number of hidden layers (see Table 7). By construction, we can assert that our best lower (resp. upper) bound is 17.21% (resp. 17.25%) for a Black-Scholes price (i.e., $\beta = 0$) 15.91%.

| Number of hidden layers | Lower bound | Upper bound |
|---|---|---|
| 2 | 17.16 | 17.28 |
| 4 | 17.19 | 17.61 |
| 8 | **17.21** | **17.25** |
| 20 | 17.19 | 17. 59 |

TABLE 7. IM (in percent) as a function of the number of hidden layers of dimension 2: $\beta = 0.01\sigma$, $\sigma = 0.2$. $T = 1$ year and $d = 2$. $\Delta t_{\mathrm{Z}} = 1/100$.

## REFERENCES


[1] Kingma, D.P., Ba, J. : *Adam: A Method for Stochastic Optimization*, 3rd International Conference for Learning Representations, San Diego, 2015, arXiv:1412.6980.

[2] Bach, F., Moulines, E. : *Non-strongly-convex smooth stochastic approximation with convergence rate O(1/n)*, Advances in Neural Information Processing Systems (NIPS), vol. 26, pp.773-781, 2013.

[3] Beck, C., E, W., Jentzen, A. : *Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations*, arXiv:1709.05963.

[4] Becker, S., Cheridito, P., Jentzen, A. : *Deep optimal stopping*, arXiv:1804.05394.

[5] Goodfellow, I., Bengio, Y., Courville, A. : *Deep Learning*, MIT press (2016).

[6] Bishop, C. : *Pattern Recognition And Machine Learning*, Springer 2006.

[7] Bouchaud, J-P, Potters, M., Sestovic, D. : *Hedge your Monte Carlo*, Risk magazine (March 2001).

[8] Becker, S., Cheridito, P., Jentzen, A. : *Deep optimal stopping*, arXiv:1804.05394.

[9] Fujii, M., Takahashi, A., Takahashi, M. : *Asymptotic Expansion as Prior Knowledge in Deep Learning Method for high dimensional BSDEs*, arXiv:1710.07030.

[10] Guyon, J., Henry-Labordère, P. : *Nonlinear Option Pricing*, Financial Mathematics Series CRC, Chapman Hall (447 p.), 2013.

[11] Guyon, J., Henry-Labordère, P. : *Uncertain Volatility Model: A Monte-Approach based on BSDE*, Journal of Computational Finance, Volume 14/Number 3, Spring 2011.

[12] Han, J., Jentzen, A., E, W. : *Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning*, arXiv:1707.02568.

[13] Han, J., E, W.: *Deep Learning Approximation for Stochastic Control Problems*, arXiv:1611.07422.

[14] Henry-Labordère, P., Litterer, C., Ren, Z. : *Dual Algorithm for Stochastic Control Problems: Applications to Uncertain Volatility Models And CVA* , SIAM J. Finan. Math., 7(1), 159–182 (2016).

[15] Henry-Labordère, P. : *Counterparty Risk Valuation: A Marked Branching Diffusion Approach*, Risk magazine (Jul. 2012).

[16] Henry-Labordère, P., Oudjane, N., Tan, X., Touzi, N., Warin, X. : *Branching diffusion representation of semilinear PDEs and Monte-Carlo approximation*, submitted, arXiv:1603.01727.

[17] Henry-Labordère, P., Touzi, N., Tan, X. : *A numerical algorithm for a class of BSDE via branching process*, Stochastic Processes and their Applications, (2013).

[18] Henry-Labordère, P., Touzi, N. : *Branching Diffusions Representation for Initial Value PDE Problems*, preprint (2017).

[19] Henry-Labordère, P. : *Deep primal-dual algorithm for BSDEs: Applications of machine learning to CVA and IM*, Available at SSRN: https://ssrn.com/abstract=3071506.

[20] Henry-Labordère, P. : *Optimal posting of Collateral with recurrent neural networks*, Available at SSRN: https://ssrn.com/abstract=3140327.

[21] Hernandez, A. : *Model calibration with neural networks*, Risk magazine (June 2017).

[22] Hornik, K. : *Approximation capabilities of multilayer feedforward networks*, Neural Networks, Volume 4, Issue 2, 1991, Pages 251-257.
[23] Hutchinson, J.M., Lo, A. W., Poggio, T. :*A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks*, The Journal of Finance, Vol. 49, No. 3, pp. 851–889 (Jul., 1994).
[24] Kennedy, J., Eberhart, R. : *Particle Swarm Optimization*, Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948. doi:10.1109/ICNN1995.488968 (1995).
[25] Kondratyev, A., Giorgidze, G. : *Evolutionary algos for optimising MVA*, Risk magazine (Dec. 2017).
[26] Longstaff, F.A., Schwartz, E.S. : *Valuing American Options by Simulation: A Simple Least-Squares Approach*, The Review of Financial Studies Spring 2001 Vol. 14. No. 1, pp. 113–147.
[27] Piterbarg, V. : *Stuck with collateral*, Risk magazine (Nov. 2013).
[28] Rogers, L.C.G.: *Monte Carlo valuation of American options*, Mathematical Finance 17, 271-286 (2002).
[29] Xiao, P., Venayagamoorthy, G. K., Corzine, K.A. : *Combined Training of Recurrent Neural Networks with Particle Swarm Optimization and Back-propagation Algorithms for Impedance Identification*, Proceedings of the 2007 IEEE Swarm Intelligence Symposium (SIS 2007).
[30] E, W., Jiequn, J., Jentzen, A.: *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations*, Communications in Mathematics and Statistics December 2017, Volume 5, Issue 4, pp 349–380, arXiv:1706.04702.

Société Générale, Global Market Quantitative Research and CMAP, Ecole Polytechnique.

*E-mail address*: pierre.henry-labordere@sgcib.com