DEEP PRIMAL-DUAL ALGORITHM FOR BSDES:
APPLICATIONS OF MACHINE LEARNING TO CVA AND IM

PIERRE HENRY-LABORDÈRE

Abstract. Building heavily on the recent nice paper [18], we introduce a primal-dual method for solving BSDEs based on the use of neural networks, stochastic gradient descent and a dual formulation of stochastic control problems [9]. Our algorithm is illustrated with two examples relevant in Mathematical Finance: the pricing of counterparty risk and the computation of initial margin.

1. Introduction

Solving numerically high-dimensional non-linear partial differential equations (in short PDE) is a huge topic in Applied Mathematics. Because of the curse of dimensionality, deterministic methods, such as finite elements, are not applicable and therefore one needs to rely on probabilistic Monte-Carlo algorithms that are in principle, from the central limit theorem, immune to the dimension. In Finance, by including market frictions such as transaction costs, uncertain volatility, illiquidity, American optimality, linear Black-Scholes PDEs need to be replaced by non-linear second-order parabolic PDEs (see [6] for additional examples). Parabolic PDEs satisfy a comparison principle (as opposed to hyperbolic PDEs) and therefore only this class is fully compatible with the no-arbitrage condition.

This deep connection with option pricing has generated widespread activity in this field and notable numerical methods have been developed in this context. Let us cite primal-dual American Monte-Carlo methods and backward stochastic differential equations (in short BSDE) that can be seen as the t-value of a delta hedged portfolio of a financial claim in layman's term. These methods are described in details in [6]. Recent market adjustments, such as counterparty value adjustment (CVA) and initial margin (IM), have revived again this interest in efficient Monte-Carlo schemes. The resulting nonlinear Black-Scholes PDEs can then be tackled in principle with BSDEs. Unfortunately, in practice this method requires the calculation of conditional expectations using (parametric) regressions. Finding good quality regressors is notably difficult, especially for multi-asset portfolios.

This restriction leads us to introduce a forward Monte-Carlo algorithm, in the context of the pricing of CVA, based on marked branching diffusions [10]. This numerical scheme has then be extended to cover fully non-linear PDEs [11, 12] and also general PDEs [13], not only restricted to the parabolic class, including hyperbolic and dispersive PDEs. As a main restriction, the marked branching diffusions can however cover exactly only polynomial non-linearities in the solution and their derivatives.

In the particular case of CVA, the non-linearity is given by the function $u \mapsto u^+$, representing the positive part of the mark-to-market position, and therefore one needs to introduce a polynomial

Received by the editors November 2017.
Key words and phrases. BSDE, stochastic control, machine learning, CVA, initial margin.

approximation if one wants to use marked branching diffusions. Using such a polynomial approximation of order M from above and below, $\underline{P}_M(u) \leq u^+ \leq \overline{P}_M(u)$, one obtains then lower and upper estimates of the true CVA from the maximum principle, that do not require nested Monte-Carlo computations (see in particular [6], Chapter 13 where automatic polynomial approximations are built for a given value M). However, when the polynomial approximation increases (with M), it appears sometimes that the corresponding PDEs with non-linearities $\overline{\underline{P}}_M(u)$ blow up in finite time. It is therefore not possible to take the limit $M \to \infty$ and converge towards the exact CVA price, although the lower and upper estimates, for some values of M, could be tight enough for practical purposes. In the case of IM, the non-linearity depends on the gradient of the solution ∇u through $\sqrt{\nabla u^\dagger \rho \nabla u}$ and the appearance of the square root leads to the same restriction.

Recently, a new numerical scheme for BSDEs, called *deep BSDE solver*, which does not require the computation of conditional expectations, was introduced. In financial terms, this scheme consists in approximating the deltas by a parametric form depending on some weights W, parameterizing neural networks, and then minimizing the variance of the P&L of the financial claim at maturity over W. This method is similar in spirit to the "hedged Monte-Carlo" method introduced in [4] for pricing linear BSDEs. If the L^2–error converges to zero, one can assert that we have converged towards the true solution. However, in practice, this error is non-zero and therefore it is difficult to see if we are close or still far away from the exact solution.

In this paper, by relying strongly on the link between non-linear parabolic PDEs and stochastic control theory, we explain how to deduce robust lower and upper estimates. These bounds are optimal and equal when the L^2–error vanishes. Our approach is similar in spirit to the method used for pricing American options: (1) The Longstaff-Schwartz algorithm is used to determine a sub-optimal policy by backward induction. (2) This sub-optimal policy is then used for deducing a lower bound using a forward Monte-Carlo. (3) Finally, by using a dual representation of the stopping time formulation, we can deduce an upper bound. Step (1) corresponds precisely to the deep BSDE solver. Our paper explains how to complement step (1) with steps (2–3).

The contents of our paper is as follows. In the first part, we give a short introduction to neural networks, in particular the forward and backward propagation, that involves automatic differentiation, familiar in quantitative finance. Then, in the second part, after a quick recap of the deep BSDE solver as introduced in [18] (see also [2, 7, 8, 18]), we explain our primal and dual deep algorithm. In the last part, we illustrate our method with some numerical experiments and conclude with some ongoing projects.

2. Neural networks in a nutshell

2.1. **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: 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 function can be exactly recovered using one-dimensional $n(2n+1)$ universal C^0 functions and $(2n+1)$ one-dimensional specific functions. Note that the proof is constructive. Having in mind this result, a neural network with linear outputs, seen as an "universal approximator", is a function $a_N : \mathbb{R}^p \to \mathbb{R}^q$ obtained by successive compositions of one-dimensional functions,

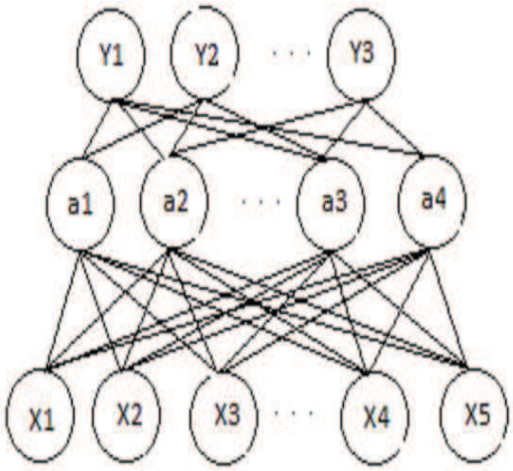

FIGURE 1. Representation of a neural network with 1 hidden layer ($p = 5, q = 3$).

linear and non-linear (here tanh):

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

with

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

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) and some vector weights (b_l). Here $N-1$ is called the number of hidden layers.

2.2. **Forward and backward propagation.** A key feature of 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 chain rule, the gradients for a function $j(a_N(x))$ are given by backward induction as

$$\begin{aligned} \delta_i^N : &= [\nabla j(y)]_i (1 - (a_N^i)^2), \quad y := a_N(x), \quad i = 1, \cdots, q \\ \left[\frac{\partial j}{\partial W_N}\right]_{ij} &= \delta_i^N a_{N-1}^j, \quad \left[\frac{\partial j}{\partial b_N}\right]_i = \delta_i^N, \quad \left[\frac{\partial j}{\partial a_{N-1}}\right]_i = \left[W_N^\dagger\right]_{ik} \delta_k^N \\ \delta_i^{N-1} &:= \left[\frac{\partial j}{\partial a_{N-1}}\right]_i (1 - (a_{N-1}^i)^2) \\ \left[\frac{\partial j}{\partial W_{N-1}}\right]_{ij} &= \delta_i^{N-1} a_{N-2}^j, \quad \left[\frac{\partial j}{\partial b_{N-1}}\right]_i = \delta_i^{N-1}, \quad \left[\frac{\partial j}{\partial a_{N-2}}\right]_i = \left[W_{N-1}^\dagger\right]_{ik} \delta_k^{N-1}, \quad \cdots \end{aligned} \tag{2.1}$$

Note that we have used here that $\tanh'(x) = 1 - \tanh(x)^2$.

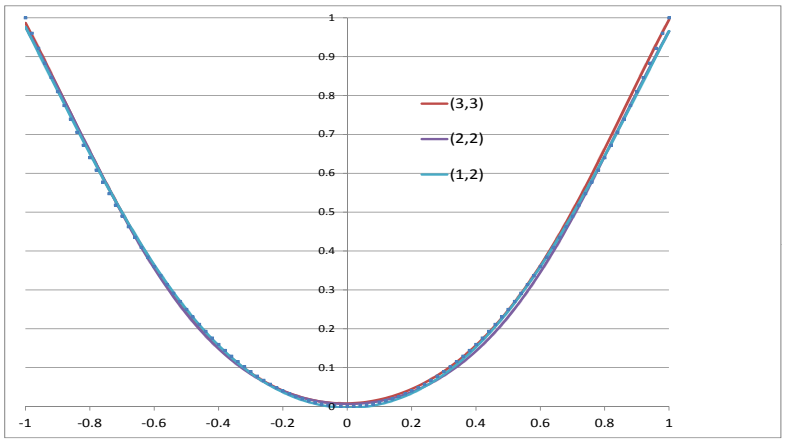

FIGURE 2. Fit over $f(x) = x^2$ with $N-1$ hidden layers with matrix weights W of dimension M (denoted (M, N)).

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

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

over θ can be obtained by using a stochastic gradient descent (in short SGD). Living apart efficient extensions (see for example [1]), the basic method consists in drawing an uniform random variable I in $[[1, m]]$ and then applying the (online) gradient descent on $j(a_N(x^I))$:

$$\theta_t \mapsto \theta_t - \eta_t \nabla_\theta j(a_N(x^I)), \quad I \text{ Uniform},$$

for a positive parameter $\eta_t \in \mathbb{R}_+^*$ at a step t and then iterate until convergence. If we impose that $\sum_{t=1}^{\infty} \eta_t = \infty$ and $\sum_{t=1}^{\infty} \eta_t^2 < \infty$, one can prove that $\lim_{t\to\infty} \theta_t = \theta^*$ almost surely with θ^* a local minimizer of J (minimizer if J is convex). For details, the reader is strongly advised to consult book [3].

In Figure 2, as a simple example, we have plotted the one-dimensional function $y = a_N(x)$ obtained by minimizing the L^2–norm with respect to θ:

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

over a train set $(x_i)_{1\leq i\leq m} \in [-1, 1]$ with $f(x) := x^2$, and composed of $m = 100$ points. As a more involved example in [14], 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. In the next section, we deviate from this paradigm where neural networks are used to learn pricing or calibration formulas (we will come back to this topic in Section 5) and explain their efficiency for solving also non-linear PDEs.

3. Deep primal-dual BSDE solver

3.1. **Semilinear PDE and BSDE.** As a simple example, let us consider the semilinear PDE

$$\partial_t u + \mathcal{L}u + f(u) = 0, \quad u(T,x) = g(x), \quad x \in \mathbb{R}^d \tag{3.1}$$

where f is a non-linear Lipschitz function and $\mathcal{L} := b.\nabla + \frac{1}{2}\sigma\sigma^\dagger\nabla^2$ is the infinitesimal generator of an Itô process $X \in \mathbb{R}^d$:

partie à expliquer

$$dX_t = b(t,X_t)dt + \sigma(t,X_t).dW_t \tag{3.2}$$

Note that although our primal-dual algorithm, as described in this paper, applies to fully nonlinear PDEs of the form

$$\partial_t u + \mathcal{L}u + f(t,x,u,\nabla u,\nabla^2 u) = 0, \quad u(T,x) = g(x), \quad x \in \mathbb{R}^d \tag{3.3}$$

we focus on the class (3.1) for the sake of simplicity. Following closely the description of our algorithm, the extension to the general case is then straightforward.

PDE (3.1) 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

$$dY_t = -f(Y_t)dt + Z_t\sigma(t,X_t).dW_t \tag{3.4}$$

supplemented with the terminal condition $Y_T = g(X_T)$ (instead of an initial condition as for the SDE (3.2)). The link between the BSDE and the PDE is given by the relation

$$Y_t = u(t,X_t), \quad Z_t = \nabla_x u(t,X_t) \tag{3.5}$$

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 (3.1). Note that by integration from $t = 0$ to $t = T$, the BSDE can be written as

$$g(X_T) = Y_0 - \int_0^T f(Y_t)dt + \int_0^T Z_t\sigma(t,X_t).dW_t \tag{3.6}$$

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.

dire a l'oral

3.2. **Deep BSDE solver.** Because of the $1-1$ unique correspondence (3.5), solving our semilinear PDE (3.1) is therefore equivalent to solving BSDE (3.4). 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 (3.6)), 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:

$$Y_0^* := \text{argmin}_{\hat{Y}_0} \min_{z_t} \mathbb{E}[(g(X_T) - Y_T^z)^2] \tag{3.7}$$

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 $z_t := z(t,X_t)$. Indeed, if we set $z_t = \nabla_x u(t,X_t)$ and $\hat{Y}_0 = u(0,X_0)$, the L^2–error vanishes. Furthermore,

this solution is unique if f is assumed to be Lipschitz. Indeed, by assuming that we have 2 solutions $(Y_0^1, z_\cdot^1)$ and $(Y_0^2, z_\cdot^2)$, we have from Equation (3.6):

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

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

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

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 [18] (see also [2, 7, 8]) and can be described by the following steps.

Deep BSDE solver: recipe [18]**.**

(1) Simulate M Monte-Carlo paths $(X_{t_1}^m, X_{t_2}^m, \cdots, X_{t_n}^m)_{1\leq m\leq M}$ at the discretization dates $0 := t_0 < t_1 < \cdots < t_n := T$.
(2) Parameterize the functions $(z_i)_{1\leq i\leq n-1}$ with $n-1$ neural networks depending on some weights $\theta_i := (W_l^i, b_l^i)_{1\leq l\leq N_i}$:

$$z_i = a_{\theta_i}(X_{t_i}), \quad i = 1, \cdots, n-1$$

(3) Minimize over $(\theta_i)_{1\leq i\leq n-1}$, $\hat{Y}_0$ and z_0 with a SGD:

$$J(\theta) := \frac{1}{2M} \sum_{m=1}^M (Y_{t_n}^m - g(X_{t_n}^m))^2 \tag{3.8}$$

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 $(\theta_i)_{1\leq i\leq n-1}$, $\hat{Y}_0$ and z_0 can be obtained by automatic differentiation through the formulas:

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

where the gradients $\frac{\partial z_i}{\partial \theta_i}$ are given by the backward propagation of the (i) neural network (see Equations (2.1)). The (online) SGD of the functional $J(\theta)$ (3.8) with respect to the weights denoted generically $(\theta_i)_{1\leq i\leq n+1}$ (here $\theta_n := \hat{Y}_0$ and $\theta_{n+1} := z_0$) is then given by

$$\theta_i \mapsto \theta_i - \eta_t(Y^I_{t_n} - g(X^I_{t_n}))\frac{\partial Y^I_{t_n}}{\partial \theta_i}$$

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

The deep BSDE solver has been implemented in [18] 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.

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 [1]). 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 [5]) for the choice of $\hat{Y}_0$ and z_0 and also decompose $z_{t_i} := z^{f=0}_{t_i} + a_{\theta_i}(X_{t_i})$.

(iii) Despite these various acceleration techniques for the optimization, since the 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^M_0 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 Δ_{Euler} and Δ_{Z}. Firstly, we do not know if Y^M_0 is above or below Y_0. Secondly, if we increase the number of hidden layers $M' > M$ and the L^2–error decreases, it is not clear if the new estimate $Y^{M'}_0$ will be closer to Y_0 than Y^M_0. In conclusion, we can assess the accuracy of our estimate only in the case when the 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 L^2–error vanishes.

3.3. **Deep primal algorithm.** We take f convex and note that the solution of the PDE (3.1) can be represented as a stochastic control problem (more precisely non-linear PDEs originate from Hamilton-Jacobi-Bellman equations in Finance):

$$u(t,x) = \sup_{a_\cdot \in A} \mathbb{E}_{t,x}\left[e^{\int_t^T a_s ds} g(X_T) - \int_t^T e^{\int_t^s a_u du} f^*(a_s) ds \right] \tag{3.9}$$

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 (3.1).

An (optimal) lower bound. By definition of the stochastic control problem (3.9), from arbitrary choice of a control $\tilde{a}_\cdot \in A$, not necessarily optimal, we obtain a lower bound by construction:

$$\mathbb{E}_{t,x}\left[e^{\int_t^T \tilde{a}_s ds} g(X_T) - \int_t^T e^{\int_t^s \tilde{a}_u du} f^*(\tilde{a}_s) ds \right] \leq u(t,x) \tag{3.10}$$

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 (3.4). By relying on Equation (3.10), we have derived a lower bound for the semilinear PDE (3.1).

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

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

(5) Average

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

From Equation (3.10), we have when M is large:

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

Note that as long as Y_0^{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 L^2–error is still large or has even increased.

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

3.4. **Deep dual algorithm.**

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 [16]. By performing a second-independent Monte-Carlo using this (sub)-optimal stopping time τ^{LS}, we obtain a lower bound

$$\mathbb{E}[g(X_{\tau^{\text{LS}}})] \leq Y_0 \tag{3.11}$$

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 [17] 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:

$$Y_0 \leq \mathbb{E}[\sup_{t \in [0,T]} \{g(X_t) - M_t^*\}] \tag{3.12}$$

Dual formula for stochastic control. In [9], we have explained how to obtain an upper bound (possibly optimal) for a general control stochastic problem. In the simpler case of Equation (3.9), this reads

$$u(t, x) = \inf_{Z(\cdot,\cdot)} \mathbb{E}_{t,x}[\sup_a F_{t,T}^Z] \tag{3.13}$$

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 (3.13) is now inside the expectation and corresponds performing a pathwise maximization over all controls valued in the set A. The proof of (3.13) is simple (despite technical details - see [9] and the general formula) and we quickly report its proof:

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

$$\begin{aligned}
u(t,x) &:= \sup_{a_\cdot \in A} \mathbb{E}_{t,x}[e^{\int_t^T a_s ds} g(X_T) - \int_t^T e^{\int_t^s a_u du} f^*(a_s) ds], \quad \text{see Equation (3.9)} \\
&= \sup_{a_\cdot \in A} \mathbb{E}_{t,x}[e^{\int_t^T a_s ds} g(X_T) - \int_t^T e^{\int_t^s a_u du} f^*(a_s) ds - M^Z_{t,T}] \\
&\leq \mathbb{E}_{t,x}[\sup_{a_\cdot \in A}\{e^{\int_t^T a_s ds} g(X_T) - \int_t^T e^{\int_t^s a_u du} f^*(a_s) ds - M^Z_{t,T}\}] := \mathbb{E}_{t,x}[\sup_{a_\cdot \in A} F^Z_{t,T}] \\
&\leq \inf_{Z(\cdot,\cdot)} \mathbb{E}_{t,x}[\sup_{a_\cdot \in A} F^Z_{t,T}]
\end{aligned}$$

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

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

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 (3.1) in the third line. This gives the reverse inequality and we conclude our proof. □

From HJB to random pathwise HJ. Equation (3.13) 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

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

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:

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

Taking the supremum over a, we get finally:

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

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 (θ_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 $(\theta_i^*)_{1\le i\le n-1}$, simulate M independent MC paths and compute

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

with

$$U^{m,*}_{t_{i-1}} = U^{m,*}_{t_i} + f(U^{m,*}_{t_i})\Delta - a_{\theta^*_{i-1}}(X^m_{t_{i-1}})\sigma(t_{i-1},X^m_{t_{i-1}}).\Delta W^m_{i-1}, \quad U^m_{t_n} = g(X^m_{t_n}) \tag{3.14}$$

By construction, we have when M is large:

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

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.

4. Numerical Experiments

4.1. **CVA-like PDE.** We consider the semilinear PDE (3.1) 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^{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 β coincides with the intensity of default of a counterparty (see e.g. [10]). Below, we have used our primal-dual algorithm for deriving lower and upper estimate of $u^{\mathrm{CVA}}(0, X_0)$ 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 [18] and the two timesteps are $\Delta t_{\mathrm{Z}} = 1/20$ and $\Delta t_{\mathrm{Euler}} = 1/100$. 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 [5]), we have decomposed z_{t_i} as

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

where $z_{t_i}^{\mathrm{BS}}$ 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 1). 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 2). 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 1. 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 2. 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 (3.14)). 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 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).

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

$$\partial_t u + \frac{1}{2}\sigma^2 \sum_{i=1}^{d} x_i^2 \partial^2_{x_i} u + f(x, \nabla u) = 0, \quad f(x, \nabla u) := \beta \sqrt{\sum_{i=1}^{n} (x_i \partial_{x_i} u)^2} \tag{4.1}$$

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 Δ (typically 10 days): α_{IM} such that

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

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

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

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

Primal. As f is convex in ∇u, the associated stochastic control problem is:

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

where the supremum is taken over all adapted control valued in S_d^β, the d-dimensional sphere of radius β. 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 (4.1) (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 $(b_i)_{1\leq i\leq n-1}$ is computed (using an optimization solver with simple quadratic constraints $\sum_{k=1}^d (b_i^k)^2 := \beta^2$ for all $1 \leq i \leq n-1$), the function reads $J^m(\theta, b^*(\theta))$ with $b^*(\theta)$ the optimizer. This implies that the gradient with respect to θ 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 3) as in Table 2. 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 3. 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 ∇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 4). 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 4. 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 $d = 2$. $\Delta t_{\mathrm{Z}} = 1/100$.

5. Learning the Black-Scholes formula: revisited

We conclude our paper with some ongoing projects we think deserve further studies. In [15], neural networks were used for learning option pricing formula. As a simple illustration, let us consider the Black-Scholes formula which depends on a spot value S_0, a strike K, a maturity T (we consider for the sake of simplicity zero rates, repos and dividends):

$$C = \mathrm{BS}(S_0, T, K, \sigma)$$

Then we train our neural network over an M-dimensional trained set $(C_i := \mathrm{BS}(S_0^i, T^i, K^i, \sigma^i))_{1\leq i\leq M}$:

$$\frac{1}{M}\sum_{i=1}^{M}(a_N(S_0^i, T^i, K^i, \sigma^i) - C_i)^2$$

By proceeding like this, our learning process should be poor. Indeed, the Black-Scholes formula is not just a black-box formula but it is characterized as the (unique) replication price of an option with maturity T and payoff $(S_T - K)^+$ in a log-normal model. In BSDE term, it consists in solving

$$Y_0^{S_0,T,K,\sigma} := \mathrm{argmin}_{\tilde{Y}_0} \min_{z_\cdot} \mathbb{E}[(g(S_T^\sigma) - Y_T^{z,\sigma})^2]$$

$$dY_t^{z,\sigma} = z_t\sigma S_t dW_t, \quad dS_t = \sigma S_t dW_t, \quad Y_0^{z,\sigma} = \hat{Y}_0$$

By taking in account this characterization, our learning process can be written instead as

$$\frac{1}{M}\sum_{i=1}^{M}(Y_0^{S_0^i,T^i,K^i,\sigma^i} - C_i)^2$$

where $Y_0^{S_0^i,T^i,K^i,\sigma^i}$ is computed using a neural network for $(z_k = a_{\theta_k}(S_k))_{1\leq k\leq n-1}$. Note that our parametrization of a pricing formula through the use of a neural network can be useful for computing quickly for example (an approximation of) CVA and IM or performing P&L backtesting.

References

[1] 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.

[2] 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.

[3] Bishop, C. : *Pattern Recognition And Machine Learning*, Springer 006.

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

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

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

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

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

[9] 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).

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

[11] 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.

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

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

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

[15] 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).
[16] 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.
[17] Rogers, L.C.G.: *Monte Carlo valuation of American options*, Mathematical Finance 17, 271-286 (2002).
[18] 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 Ecole Polytechnique Paris, Centre de Mathématiques Appliquées.

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