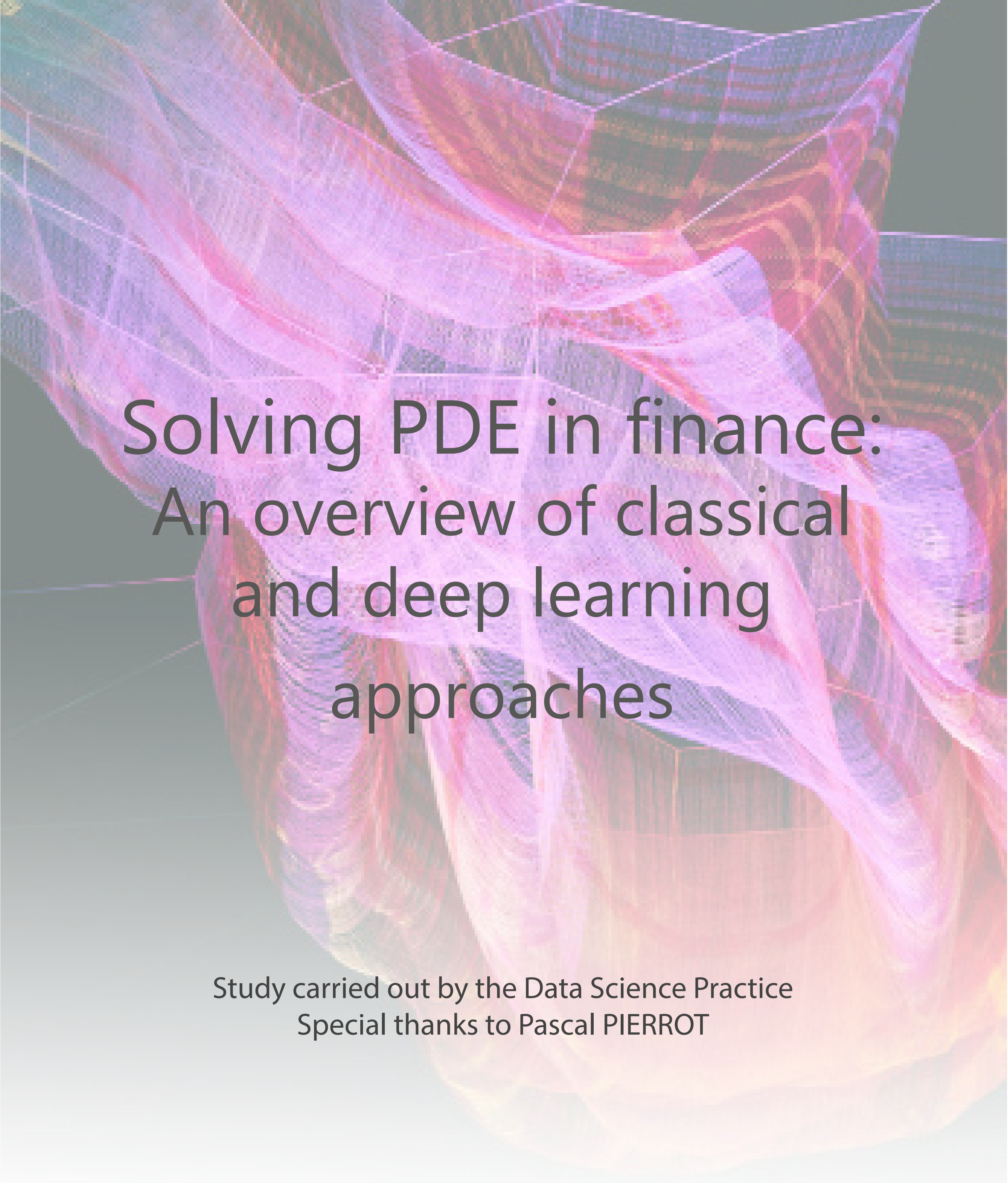

Solving PDE in finance: An overview of classical and deep learning approaches

Study carried out by the Data Science Practice
Special thanks to Pascal PIERROT

coperneec

MAI 2022

Summary

Introduction

The aim of this note is to introduce some recent development in numerical solving of partial differential equations (PDE) with the use of deep neural networks. We particularly focus on PDE that arise in option pricing.

In a first part, we introduce the notion of partial differential equation and give some example of such equation in finance. In a second part, we briefly develop some classical numerical methods such as Finite difference, its convergence and limits. Then we introduce the Galerkin method and its deep learning version which is a deterministic approach particularly adapted for high dimension PDE. In a last part, we introduce the probabilistic approach with Backward stochastic differential equation representation of PDE and a first approach to solve it with Neural Networks. Finally, we test these different approaches to option pricing problems.

1 General overview of PDE

1.1 Some definitions

A PDE represents the relation between a multivariate function and its partial derivatives. It can be defined as a general expression of the form :

$$F\left(D^k u(x), D^{k-1}u(x), .., Du(x), u(x), x\right) = 0$$

where $x \in \Omega \subset \mathbb{R}^n$, $u : \Omega \to \mathbb{R}$ is an unknown function and D^j a collection of partial derivatives of order j :

$$D^j \equiv \left(\frac{\partial^j}{\partial x_1^{i_1}...\partial x_n^{i_n}}\right)_{i_1+..+i_n=j}$$

A PDE might be subject to boundary conditions. the Dirichlet boundary condition is expressed as $u(x) = g(x) \quad \forall x \in \partial\Omega$ where g is a known function and $\partial\Omega$ is the boundary of its domain Ω. Other boundary conditions exists such as Neumann (conditions on the first derivative of u) or Cauchy (conditions on both value and first derivative).

PDE can be classified into different forms:

- linear PDE are of the form $\sum_{|j|\leq k} a_j(x)D^j u(x) - f(x) = 0$
- semi-linear PDE are of the form $\sum_{|j|=k} a_j(x)D^j u(x) - f\left(D^{k-1}u, .., Du, u, x\right) = 0$
- quasi-linear PDE : $\sum_{|j|=k} a_j(D^{k-1}u, .., Du, u, x)D^j u(x) - f\left(D^{k-1}u, .., Du, u, x\right) = 0$
- Fully-nonlinear depends non-linearly on the highest order derivatives

1.2 Examples in finance

PDE arise in many problems of quantitative finance : derivative pricing, optimal portfolio theory or optimal execution. In this note, we limit our study to common option pricing in the Black-Scholes model.

European option

In the Black-Scholes framework, a non-dividend paying asset price is assumed to follow a Geometric Brownian process which can be expressed under the risk neutral measure as:

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{1}$$

A PDE for the option (call) price can be derived by applying the Ito lemma to the option price expressed as a function of t and S_t (possible by the Markovian property of the process), and by using the principle of portfolio replication :

$$\begin{aligned} &\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial V^2}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \\ &V(t = T, S_T) = (S_T - k)^+ \end{aligned} \tag{2}$$

European Basket option

A basket option is an option on several underlyings $S_t = (S_t^1, ..., S_t^d)$ each following a Geometric Brownian in the BS framework: $dS_t^i = rS_t^i dt + \sigma_i S_t^i dW_t^i$ with correlation matrix ρ, that is we have $\forall i, j : d\langle W^i, W^j\rangle_t = \rho_{i,j}dt$

In the European case, if we note $\phi(S_T) = \phi(S_T^1, ..., S_T^d)$ the payoff, we can derive a PDE using the multi-dimensional Ito formula on the pricing function V which depends on $t, S_t^1, .., S_t^d$:

$$\begin{aligned} &\frac{\partial V}{\partial t} + \frac{1}{2}\sum_{i,j}\sigma_i\sigma_j\rho_{i,j}S^iS^j\frac{\partial^2 V}{\partial S^i \partial S^j} + r\sum_i S^i\frac{\partial V}{\partial S^i} - rV = 0 \\ &V(T, S_T) = \phi(S_T) \end{aligned}$$

American option

An American option gives the right to its owner to exercise the option at any time $t \leq T$. If we consider the same underlying model as previously, it can be shown assuming some smoothness (see Ref [1]) that the price V of the option with payoff ϕ verifies the following PDE (more precisely a variational inequality) :

$$\begin{aligned} &\max\left\{\frac{\partial V}{\partial t} + \mathcal{L}V - rV, \phi - V\right\} = 0 \quad (t, x) \in [0, T] \times \mathbb{R}^+ \\ &V(T, x) = \phi(x) \quad x \in \mathbb{R}^+ \end{aligned} \tag{3}$$

$\mathcal{L}$ being the differential operator $\mathcal{L} := rx\frac{\partial}{\partial x} + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2}{\partial x^2}$.

As a simple interpretation, when $V(t, x) < \phi(x)$ the second term must be equal to zero, that is the price verifies the usual Black-Scholes PDE. When $V(t, x) = \phi(x)$, the option value equals the exercise value and is exercised. This is an example of free boundary problem as the boundary curve $\{(t, x) : V(t, x) = \phi(x)\}$ is determined alongside the price function V.

An American option on multiple underlyings (Basket) verifies the same form of PDE with:

$$\mathcal{L} := \frac{1}{2}\sum_{i,j}\sigma_i\sigma_j\rho_{i,j}S^iS^j\frac{\partial^2 V}{\partial S^i \partial S^j} + r\sum_i S^i\frac{\partial V}{\partial S^i}$$

2 The finite difference method

As a number of PDE cannot be solve analytically, this is the case for the American option, numerical methods are considered. A first and usual approach to solve PDE in finance, particularly for option pricing problem is the finite difference.

2.1 The principle

The finite difference method consists in the discretization of the domain Ω and the approximation of partial derivatives from the Taylor formula:

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x-h)}{2h} + o(h) \\ f''(x) &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + o(h^2) \end{aligned} \tag{4}$$

Where the terms are obtained by subtracting (resp. summing) the Taylor formulas in $x + h$ and $x - h$

Let's consider the Black-Scholes equation (2) to illustrate the principle.

The domain of the equation (2) is $\Omega = [0, T] \times \mathbb{R}^+$. If we want to discretize the equation, we need to set up boundary value for $S \in \mathbb{R}^+$. This is done by first chosing a value S_{max} sufficiently large and discretizing over $[0, T] \times [0, S_{max}]$. For this we define $\{S_0, S_1, ..., S_I = S_{max}\}$ where $S_i = i\delta S = iS_{max}/I$ and $\{t_0, ..., t_N\}$ where $t_n = n\delta_t = nT/N$.

We can now apply the discretization scheme on our equation. For this we define V_i^n the grid values to find (ie the solution of the numerical scheme) which attempt to approximate the exact solution of the PDE: $V_i^n \approx V(t_n, S_i)$.

Then, we need to setup a value for the prices in $S = S_{max}$: $(V_I^n)_{n=0..N}$. Indeed, in order to calculate the approximated derivatives in I, we need the values V_{I+1} which are not defined. We know from the Black-Scholes theory that for a call we have $V(t, S) \sim S - K\exp(-\int_t^T r ds)$ when $S \to +\infty$. Thus, we set $V_I^n = S_{max} - K\exp(-\int_{t_n}^T r ds)$. We also have in $S = 0$: $V_0^n = 0$

As the price is only known in $t = T$, we need to proceed backward-in-time. There are three classical numerical schemes for PDE.

The first one is the **explicit Euler scheme** which is the following :

$$\frac{V_i^{n+1} - V_i^n}{\delta t} + \frac{1}{2}\sigma^2 S_i^2 \frac{V_{i+1}^{n+1} - 2V_i^{n+1} + V_{i-1}^{n+1}}{\delta S^2} + rS_i \frac{V_{i+1}^{n+1} - V_{i-1}^{n+1}}{2\delta S} - rV_i^{n+1} = 0$$

$V_i^N = (S_i - K)^+$

In this scheme, the values for $(V_i^n)_{0 \le i \le I}$ can be obtained from the values $(V_i^{n+1})_{0 \le i \le I}$. It is possible to put the expression in the following general form : $V_i^n = a_i V_{i-1}^{n+1} + b_i V_i^{n+1} + c_i V_{i+1}^{n+1}$ or in a more condensed way $V^n = MV^{n+1}$ where M is a tridiagonal matrix. This makes the computation quite efficient in comparison to other schemes. Nevertheless, we will see in the next section that this method is potentially unstable and that is a reason why it is not often used in practice.

The second method is the **Implicit Euler scheme** :

$$\frac{V_i^{n+1} - V_i^n}{\delta t} + \frac{1}{2}\sigma^2 S_i^2 \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{\delta S^2} + rS_i \frac{V_{i+1}^n - V_{i-1}^n}{2\delta S} - rV_i^n = 0$$

$V_i^N = (S_i - K)^+$

Which is of the form: $V_i^{n+1} = a_i V_{i-1}^n + b_i V_i^n + c_i V_{i+1}^n$ or $V^{n+1} = MV^n$

In this case the values $(V_i^n)_{0 \le i \le I}$ cannot be obtained directly from values in $n+1$ and need the resolution of the above linear system which is more computationally demanding. Yet this method has some advantages that we will see. We show how to solve such system in Appendix.

The third method is the so-called **Crank-Nicolson** scheme. The scheme is also an implicit scheme obtained by taking the average of the two precedent one:

$$\begin{aligned} \frac{V_i^{n+1} - V_i^n}{\delta t} + \frac{1}{2}\Bigg[& \frac{1}{2}\sigma^2 S_i^2 \frac{V_{i+1}^{n+1} - 2V_{i-1}^{n+1} + V_{i-1}^{n+1}}{\delta S^2} + rS_i \frac{V_{i+1}^{n+1} - V_i^{n+1}}{2\delta S} - rV_i^{n+1} \\ & + \frac{1}{2}\sigma^2 S_i^2 \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{\delta S^2} + rS_i \frac{V_{i+1}^n - V_{i-1}^n}{2\delta S} - rV_i^n = 0 \end{aligned}$$

The Crank-Nicolson scheme is a particular case of more general θ-schemes where a weighted average of the first two schemes is taken ($\theta = \frac{1}{2}$ for CN).

2.2 consistency, stability and convergence

There are three important notions to be considered when numerically solving a PDE:

- **Consistency**: a numerical scheme is said to be consistent if when the exact solution is plugged into the numerical scheme, i.e V_i^n is replaced by $V(t_n, S_i)$, the error (the equation term) tends to zero when $\delta t, \delta S \to 0$.
- **Stability**: a scheme is said to be stable if the solution of the numerical scheme is bounded by a constant times a boundary value. For example, in our case if $\exists C > 0$ s.a $\|V^n\| \le C\|V^N\|$. In practice we use L^∞ or L^2-norm
- **Convergence**: convergence ensure that the numerical solution tend to an exact solution of the original PDE : For V solution of the PDE and (V_i^n) solution of the numerical scheme we have : $V_i^n \xrightarrow[\delta t, \delta S \to 0]{(n\delta t, i\delta S) \to (t,S)} V(t, S)$.

It can be shown that a consistent and stable numerical scheme is convergent. The three previous methods are consistent. Indeed, from Taylor expansions (4), the left term of the numerical schemes is bounded by $C(\delta t + \delta S^2)$. Concerning the stability, the Euler explicit scheme is stable only if δt and δS meet the following CFL conditions : $\delta t \le \alpha \delta S^2$. This requires choosing a relatively small time step and slows down the execution. On the other hand, the two other schemes have the advantage to be unconditionally stable. Finally, the estimate of convergence is given by the estimate of consistency error (which is the truncation error, depending on $\delta t, \delta S^2$). The additional advantage of CN is that its error in time is quadratic.

2.3 Numerical example

In this section we apply the three precedent schemes to an European call option with following parameters : $r = 0.05$, $sigma = 0.2$, $T = 1$, $K = 110$ and $S_{max} = 2K$. We compare with the true option values given by the Black-Scholes analytical formula.

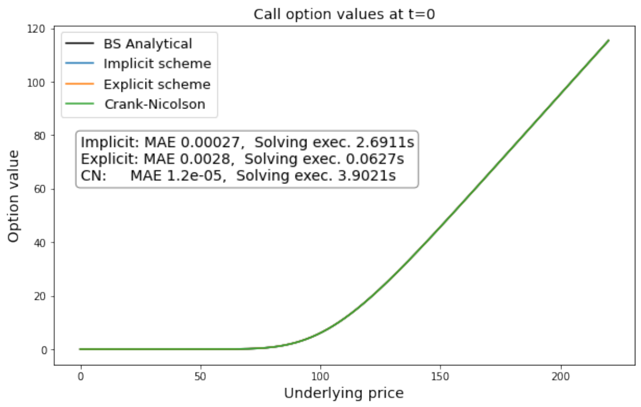

Figure 1: Initial call price obtained for a 1000×1000-size grid for implicit schemes, 1000×100-size grid for explicit scheme and full-grid performance

2.4 Higher dimensions and American case

For the American option equation, the application of the Euler explicit scheme to solve (3) is quite easy as we obtain the following formulation :

$$\max\left((MV^{n+1} - V^n)_i, \phi(S_i) - V_i^n\right) = 0$$

$$\text{which can be expressed as: } V_i^n = \max\left((MV^{n+1})_i, \phi(S_i)\right)$$

Nevertheless, for the implicit schemes, two difficulties arise. The first one is that applying directly these schemes to (3) leads to a non linear system of equations for determining V^n which arise the question of well-posedness. The second is the difficulty of convergence study. Methods exist to circumvent the first one such as the splitting method presented in [1]. Authors of [1] also study the convergence in the more general class of Hamilton-Jacobi equations.

For higher dimension in space as in the case of multi-underlying options or stochastic volatility for instance, it is still possible to apply a FD method. Yet in practice, the exponential growth of the grid (a d-dimensional problem would need a $O(N^d)$ grid size involving as many operations) limits the use of finite difference methods up to 3 dimensions.

3 The Deep Galerkin approach

The Galerkin method is an alternative to finite differences which consists in choosing an approximation space for u with a finite basis.

3.1 Principle of the Galerkin method

To introduce the Galerkin method, let's consider our PDE under the form $F(x) = y$ where $F : X \to Y$ is an operator (possibly non-linear) between two functional spaces X and Y (basically Banach spaces) equipped with an inner product $\langle \cdot, \cdot \rangle$. We suppose there exists a linearly independent basis $\{\phi_i\}_{i=1}^{\infty}$ and $\{\psi_i\}_{j=1}^{\infty}$ of X and Y respectively. In the Galerkin method, $x \in X$ is approximated by : $x \approx x_n := \sum_{i=1}^n c_i\phi_i$ and the coefficient are determined by the following system of equations:

$$\left\langle F\left(\sum_{i=1}^n c_i\phi_i\right), \psi_j\right\rangle = \langle y, \psi_j\rangle \quad j = 1, \dots, n$$

In practice the choice of basis and n are important to ensure the well-posedness of such system and to ensure the convergence of x_n toward x.

3.2 The Deep version

The Deep Galerkin method as introduced in [2] is mesh-free algorithm where the solution of a PDE is approximated by a neural network instead of the above linear combination of basis. The authors apply this method to quasilinear parabolic PDE.

Let's consider a PDE of the following form:

$$\frac{\partial u}{\partial t}(t,x) + \mathcal{L}u(t,x) = 0 \quad (t,x) \in [0,T] \times \Omega$$
$$u(T,x) = g(x) \quad x \in \Omega$$
$$u(t,x) = h(t,x) \quad (t,x) \in [0,T] \times \partial\Omega$$

where the second term could also be initial conditions. The idea is to approximate the true solution u by a neural network $f(t,x,\theta)$, θ being a set of weights and biases. For this the authors of [2] introduce the following objective function :

$$J(f) = \left\| \frac{\partial f}{\partial t}(t,x;\theta) + \mathcal{L}f(t,x;\theta) \right\|^2_{[0,T]\times\Omega,\nu_1} + \|f(t,x;\theta) - h(t,x)\|^2_{[0,T]\times\partial\Omega,\nu_2} + \|f(T,x;\theta) - g(x)\|^2_{\Omega,\nu_3}$$

Where the norm is the L^2 norm : $\|f\|^2_{\mathcal{Y},\nu} = \int_{\mathcal{Y}} |f(y)|^2\nu(y)dy$, ν being a positive probability density on $\mathcal{Y}$.

The first term measure how well the approximation satisfies the differential operator, the second term how well it satisfies the boundary conditions and the last term the same for terminal/initial conditions.

Since the above objective involves integrals, a direct numerical estimation and minimization of $J(f)$ would be rapidly unmanageable when dimension increases. Instead, the idea is to use a stochastic Gradient descent algorithm with points randomly drawn from the PDE domains. More precisely, it consists in first initializing parameters θ_0 and learning rate α_0, then performing the following steps :

- generate sample points s_n from the different domains: (t_n, x_n) from $[0,T] \times \Omega$, (τ_n, y_n) from $[0,T] \times \partial\Omega$ and z_n from Ω according to corresponding densities ν_1, ν_2, ν_3.
- compute the square error :
  $L(\theta, s_n) = \left(\frac{\partial f}{\partial t}(t_n, x_n; \theta_n) + \mathcal{L}f(t_n, x_n; \theta_n)\right)^2 + (f(\tau_n, y_n; \theta_n) - h(\tau_n, y_n))^2$
  $(f(T, z_n; \theta_n) - g(z_n))^2$
- perform a gradient descent : $\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta L(\theta_n, s_n)$

- repeat 1-3 until some convergence criterion is satisfied.

A similar procedure with some adaptations is also proposed by the authors to solve the Free-boundary problem (3) for American-style options.

The network architecture adopted by Sirignano and Spiliopoulos [3] for f is the following :

$$
\begin{aligned}
S^1 &= \sigma(W^1 \cdot x + b^1) \\
Z^l &= \sigma(U^{z,l} \cdot x + W^{z,l} S^l + b^{z,l}) \\
G^l &= \sigma(U^{g,l} \cdot x + W^{g,l} S^l + b^{g,l}) \\
R^l &= \sigma(U^{r,l} \cdot x + W^{r,l} S^l + b^{r,l}) \\
H^l &= \sigma(U^{h,l} \cdot x + W^{h,l}(S^l \odot R^l) + b^{h,l}) \\
S^{l+1} &= (1 - G^l) \odot H^l + Z^l \odot S^l \quad l = 1, ..., L \\
f(t, x, \theta) &= W S^{L+1} + b
\end{aligned}
$$

Where $\odot$ denotes the element-wise multiplication, L the number of layers (originally, the authors take three layers including one hidden). Here x is the spatial vector completed with the time dimension, that is $x := (t, x)$

Let's develop the rational and some intuition behind this choice. One first notices that the architecture is similar to LSTM (Long Short Term Memory) and Highway network. LSTM is a kind of recurrent neural network which overcomes the gradient vanishing problem. The gradient vanishing arises for deep architectures when the loss derivative with respect to weights in the first layers is too small to update properly these weights (this is due to successive derivative multiplications in backpropagation chain rules). Thus, this deteriorates and slow down the learning. LSTM as well as Highway introduce a notion of memory which controls how many information (gradient) should pass directly from a layer to another through the use of different gates (with their own weights). The optimal portion of information that should passed is learned by the network. In detail, the gates Z and G control how much information from previous layer need to be passed to the next layer, the hidden state H involving the reset gate R acts as a memory and controls how many information to forget from the previous layer. In each DGM layer, the original input x is also used in the calculation and prevents the gradient vanishing of the output with respect to x. Additionally, the authors argue that including the repeated element-wise multiplication of non-linear function of the inputs helps to capture the "sharp turn" present in the payoff.

3.3 Convergence of deep Galerkin

The authors of [2] state and demonstrate some important convergence results which motivate the use of neural network to approximate the solution of quasilinear PDE.

Denoting $\mathcal{NN}^n$ the class of single-layer feedforward neural networks with n units and $f^n = \underset{f \in \mathcal{NN}^n}{\operatorname{argmin}} J(f)$. they show first that under certain assumptions: $J(f^n) \to 0$, as $n \to \infty$ then that $f^n \to u$ as $n \to \infty$ in $L^\rho([0, T] \times \Omega)$, $\rho < 2$

The proof relies on regularity properties of $\mathcal{L}$ and on the universal approximation power of neural network which states that any sufficiently regular function on a compact subset of $\mathbb{R}^k$ can be approximated by an arbitrarily wide single hidden layer feedforward NN (in other words, $\mathcal{NN} := \bigcup_{i=1}^{\infty} \mathcal{NN}^n$ is dense on compacts of $\mathcal{C}^m(\mathbb{R}^k)$ if $\sigma \in \mathcal{C}^m(\mathbb{R}^k)$).

4 The probabilistic approach

In the two previous sections, we used deterministic approaches for solving a PDE. In this section, we present the probabilistic approach which consists in solving a PDE by its stochastic representation. This approach is the common way to solve high dimension PDE.

4.1 The stochastic representation of PDE

In this section, we briefly present the link between PDE and Backward stochastic differential equation (BSDE). A detailed study of BSDE can be found in reference.

We are interested in solving a semilinear (parabolic) PDE of the following form :

$$
\begin{aligned}
&\frac{\partial u}{\partial t}(t, x) + \mathcal{L}u(t, x) + f(t, x, u(t, x), \sigma D_x u(t, x)) = 0 \quad (t, x) \in [0, T] \times \mathbb{R} \\
&u(T, x) = g(x) \quad x \in \mathbb{R}^d
\end{aligned}
$$

where $\mathcal{L}u := b(x)D_x u + \frac{1}{2} Tr(\sigma\sigma^T(x) D_x^2 u)$ is a linear operator. This form of PDE arise in option pricing as for instance the European-like options of section 1.2.

Let X be the solution of the following SDE and $\mathcal{F}$ the natural filtration of its Brownian:

$$dX_t = b(X_t)dt + \sigma(X_t)dW_t \tag{5}$$

Let u be a solution to the PDE. We want to characterise the dynamic of $Y_t = u(t, X_t)$. Then, for any couple (t, x) we will be able to find a representation $u(t, x) = u(t, X_t^{t,x}) = Y_t^{t,x}$ where $X^{t,x}$ is the solution of (5) such as $X_t = x$. Assuming u is smooth enough and applying the Ito lemma to Y_t gives:

$$
\begin{aligned}
dY_t &= \left(\frac{\partial u}{\partial t}(t, X_t) + \mathcal{L}u(t, X_t) \right) dt + D_x u(t, X_t)\sigma(X_t)dW_t \\
&= -f(t, X_t, Y_t, \sigma(x)D_x u(t, X_t))dt + D_x u(t, X_t)\sigma(X_t)dW_t
\end{aligned}
$$

Considering the terminal conditions and defining $Z_t = D_x u(t, X_t)\sigma(X_t)$, we obtain the following BSDE:

$$
\begin{aligned}
dY_t &= -f(t, X_t, Y_t, Z_t)dt + Z_t dW_t \\
Y_T &= g(X_T)
\end{aligned}
\tag{6}
$$

where the pair of processes (Y, Z) appears to be a solution to this equation. The well-posedness of (6) needs further Lipschitz and regularity conditions on f and g. It is possible to prove (see reference [3]) that under certain assumptions, the BSDE (6) is indeed a stochastic representation of our PDE, that is finding u is equivalent to find (Y, Z) in some sense. More precisely, the semilinear PDE is associated to the system of equations formed by (6) and (5) with initial setup which is a so-called Forward Backward SDE (FBSDE).

Note: In the particular case of linear PDE (that is f is linear in x, y, z the stochastic representation corresponds in fact to the classical Feynman-Kac theorem where the solution of the PDE can be expressed as a conditional expectation (Ref [3]) and can be estimated with Monte-Carlo simulations on X, often used in option pricing.

4.2 The Deep BSDE scheme

In this section, we want to solve the above BSDE representation. The FBSDE system (5)+(6) can be discretized using an Euler scheme. Let's define a grid $\pi := \{t_0 = 0 < t_1 < ... < t_n = T\}$ and denote $\Delta t_i := t_{i+1} - t_i$. We have then :

$$\begin{aligned} X^\pi_{t_{i+1}} &= X^\pi_{t_i} b(X^\pi_{t_i})\Delta t_i + \sigma(X^\pi_{t_i})\Delta W_{t_i} \quad X^\pi_0 = X_0 \\ Y_{t_i} &\simeq Y_{t_{i+1}} + f(X^\pi_{t_i}, Y_{t_i}, Z_{t_i})\Delta t_i - Z_{t_i}\Delta W_{t_i} \end{aligned} \tag{7}$$

where X^π is the solution to the Euler scheme, $\Delta W_{t_i} := W_{t_{i+1}} - W_{t_i}$ and the terminal condition Y_T is approximated as $Y_T \simeq g(X^\pi_T)$. The classical numerical method to solve it consists in applying separately the followings to the second equation in (7):

- Take the conditional expectation on both sides gives: $Y_{t_i} \simeq \mathbb{E}[Y_{t_{i+1}}|\mathcal{F}_{t_i}] + f(X^\pi_{t_i}, Y_{t_i}, Z_{t_i})\Delta t_i$.
- Multiply by ΔW_{t_i} and take the same conditional expectation gives: $0 \simeq \mathbb{E}[Y_{t_{i+1}}\Delta W_{t_i}|\mathcal{F}_{t_i}] - Z_{t_i}\Delta t_i$

Combining the two relations previously obtained leads to the following backward scheme to be solved :

$$\begin{aligned} Z^\pi_{t_i} &= \mathbb{E}[Y_{t^\pi_{i+1}}\frac{\Delta W_{t_i}}{\Delta t_i}|\mathcal{F}_{t_i}] \\ Y^\pi_{t_i} &= \mathbb{E}[Y^\pi_{t_{i+1}}|\mathcal{F}_{t_i}] + f(X^\pi_{t_i}, Y^\pi_{t_i}, Z^\pi_{t_i})\Delta t_i \end{aligned}$$

with the terminal condition $Y^\pi_T \simeq g(X^\pi_T)$. This is an implicit scheme and solving it implies the computation of the conditional expectations. These last are usually approximated by quantization or Least-Square regression.

In the deep learning approach, the equation (7) is considered forward-in-time instead. We start from an estimation of the initial value $\mathcal{Y}_0 \simeq Y_0 = u(0, X_0)$ and parameterize the values $(Z_{t_i})_{0\leq i\leq n-1}$ by a family of feed forward neural networks: $Z_{t_i} \simeq \mathcal{Z}_i(X_{t_i}; \theta_i)$ where the values X_{t_i} are taken as input of the networks and θ_i is a set of weights and biases $(W^i_l, b^i_l)_{1\leq l\leq L_i}$. The objective function to minimize over $\theta := (\mathcal{Y}_0, \theta_0, ..., \theta_n)$ is then the difference between the reconstructed dynamics and the terminal condition:

$$J(\theta) = \mathbb{E}\left|Y^\theta_T - g(X_T)\right|^2$$

To Solve the problem we perform the following steps :

- Construct and initialize the neural networks parameters $(\theta_i)_i$ and $\mathcal{Y}_0$.
- forward diffuse X_t by generating Monte-Carlo paths $(X^m_{t_0}, ..., X^m_{t_n})^M_{m=0}$ at the discretization dates. Each samples $(X^m_{t_i})_{0\leq m\leq M}$ will act as a training set for the network $\mathcal{Z}_i$.
- Forward induction : compute subsequently the values :

$$\begin{aligned} Y^{\theta,m}_{t_0} &= \mathcal{Y}_0 \\ Y^{\theta,m}_{t_{i+1}} &= Y^{\theta,m}_{t_i} - f\left(X^m_{t_i}, Y^{\theta,m}_{t_i}, \mathcal{Z}_i(X^m_{t_i}; \theta_i)\right)\Delta t_i + \mathcal{Z}_i(X^m_{t_i}; \theta_i)\Delta W^m_{t_i} \end{aligned}$$

- compute the terminal loss : $L(\theta) = \frac{1}{M}\sum_{m=1}^M |Y^{\theta,m}_{t_n} - g(X^m_{t_n})|^2$
- Update parameters $\mathcal{Y}_0$ and $(\theta_i)_i$ by backpropagation and gradient descent.
- repeat until convergence the three last steps or the four last steps in case of mini-batch gradient descent (M becoming the batch size).

In order to update parameters θ_i, we need to compute $\partial L/\partial\theta_i$ which implies the calculation of $\partial Y^{\theta,m}_{t_n}/\partial\theta_i$ values. One can find by recurrent calculation from $\partial Y^{\theta,m}_{t_n}/\partial\theta_{n-1}$ the following expression for pathwise gradients which can then be calculated by automatic differentiation techniques:

$$\begin{aligned} \frac{\partial Y^{\theta,m}_{t_n}}{\partial\theta_i} &= \left(1 - \frac{\partial f}{\partial z}(X^m_{t_i}, Y^{\theta,m}_{t_i}, \mathcal{Z}_i)\Delta t_i\right)\frac{\partial\mathcal{Z}_i}{\partial\theta_i}.\Delta W^m_{t_i} \\ &\times\prod_{j=i+1}^{n-1}\left(1 - \frac{\partial f}{\partial y}(X^m_{t_j}, Y^{\theta,m}_{t_j}, \mathcal{Z}_j)\Delta t_j\right) \end{aligned}$$

where $\partial\mathcal{Z}_i/\partial\theta_i$ are calculated by backpropagation through the network $\mathcal{Z}_i$. A relation for the derivatives $\partial Y^{\theta,m}_{t_n}/\partial\mathcal{Y}_0$ can also be obtained for the update of $\mathcal{Y}_0$.

Another possibility is to stack the networks into one as in reference [4]. In this case, solving the problem is equivalent to train a single neural network where the paths $\{X_{t_i}\}_{0\leq i\leq n-1}$ and $\{\Delta W_{t_i}\}_{0\leq i\leq n}$ are the inputs and the value Y^θ_T is the output. The initial value of interest $\mathcal{Y}_0 \simeq u(0, X_0)$ and $\mathcal{Z}_0$ are trainable variables of the network and the family of feed forward networks previously described are now subnetworks that output intermediary values $\mathcal{Z}_i(X_{t_i}; \theta_i)$. The figure 5 of the Appendix helps to visualize this global architecture.

After training, a new set of paths is generated and a last forward step into the network enable to recover the value Y^θ_0, and $Y^{\theta,m}_{t_i}$, $1 \leq t_i \leq t_n$ for exposure profile (which could be used for xVA computation).

4.3 The deep backward DP scheme

The deep backward dynamic programming scheme (DBDP) is an alternative to the Deep BSDE proposed by the authors of Ref [6]. The principle is to consider the equation (7) backward-in-time, starting from the terminal condition. Then approximate simultaneously at each step (Y_{t_i}, Z_{t_i}) with neural networks: $Y_{t_i} \simeq \mathcal{Y}_i(X_{t_i}; \eta_i)$ and $Z_{t_i} \simeq \mathcal{Z}_i(X_{t_i}; \zeta_i)$, minimizing over $\theta_i := (\eta_i, \zeta_i)$ the following:

$$\begin{aligned} J_i(\theta_i) = \mathbb{E}\,|\,&\mathcal{Y}_{i+1}(X_{t_{i+1}}; \hat{\eta}_{i+1}) - \mathcal{Y}_i(X_{t_i}; \eta_i) \\ &+ f\left(X_{t_i}, \mathcal{Y}_i(X_{t_i}; \eta_i), \mathcal{Z}_i(X_{t_i}; \zeta_i)\right)\Delta t_i - \mathcal{Z}_i(X_{t_i}; \zeta_i)\Delta W_{t_i}\,|^2 \end{aligned}$$

where $\hat{\theta}_i = (\hat{\eta}_i, \hat{\zeta}_i)$ is a solution to the local minimization problem at step t_i.

For a better efficiency, at each time step, the weights and biases of the current neural network are initialized using the trained weights and biases from the previous network (transfer learning).

5 Numerical application

In this section, we apply the deep learning methods of the two last sections fist to the same European call option of section 2.3 then to a higher dimension problem with an European basket option. The algorithms are implemented in Python using Tensorflow 2.

European call

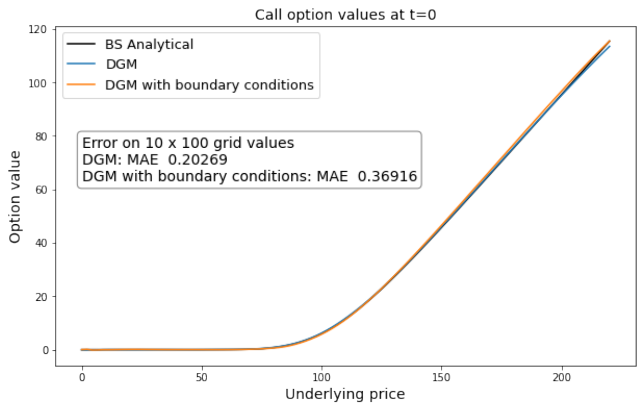

Figure 2: Initial call price obtained for a DGM with 3 layers, tanh activation after 100 sampling step of 20 mini-batch size

European basket call

- For the DGM approach, the computation of second order derivatives in the PDE becomes rapidly inefficient as the dimension increase. To overcome this, we use a trick to approximate the Hessian (Ref [5]) by finite differences of first order derivatives. The authors of [2] also propose a similar approach based on Monte-Carlo simulation.
- The BSDE for our basket option is the following :

$$dV(t,X_t) = rV(t,X_t)dt + \sum_{k=1}^{d} \sigma_k X_t^k \frac{\partial V}{\partial x_k}(t,X_t)dW_t^k$$

$$g(X_T) = V(T,X_T) = \left(\frac{1}{d}\sum_{k=1}^{d} X_T - K\right)^+$$

Which is then Euler-discretized using 100 steps. The two approaches were implemented for this case (multiple networks and the single connected network of [4]). The second case turned out to be more efficient in practice. In our case, each sub-network for Z approximates the vector-valued deltas $\mathcal{Z}_i(X_{t_i};\theta) \simeq \left(\frac{\partial V}{\partial x_k}(t_i, X_{t_i})\right)_{1\leq k\leq d}$. Results are summarized below.

d asset Eur Basket option			
K=100, T=1, r=0.05, S_0 = 100, σ_i=0.2, $\rho_{ij,i\neq j}$ = 0.25			
	model	t_0 price	exec. time
d = 10	Monte-Carlo	7.305	conf: [7.283-7.327]
	DGM	7.347	27min21s
	Deep BSDE	7.306	4min56s
d = 100	Monte-Carlo	6.858	conf: [6.838-6.876]
	Deep BSDE	6.837	41min54

For the DGM, 20 sampling steps were used with 10 batch size. For the deep BSDE, we used 2 hidden layers with $d + 10$ units as in [4], 4000 iterations of 64 batch of paths for $d = 10$ and 10000 iterations for $d = 100$. Finally, this last methods was faster, more precised and more scalable in high dimension. Yet, the efficiency and convergence are sensible to the choice of the initial value range for $\hat{Y}_0$, learning rate and batch size and slow down when the dimension increases. The learning rate has to be properly adjusted during the training progress for more efficiency. The graphs in Appendix show the convergence aspect of the different Deep learning models.

Conclusion

In this note, we have presented some way of solving PDE that arise in option pricing with a focus an recent approaches with deep learning. We particularly developed two methods, one using a deterministic approach, the second using a stochastic approach. These methods were applied to european-like options. The tested options obviously have more efficient ways to be priced in practice but the tests shows that in these simple cases, the deep learning approach can be precised and solve the problem in a reasonable time in comparison with other numerical PDE methods in higher dimension. They becomes interesting for complex equations such as non-linear ones that arise in American-like option pricing or stochastic control problems. Applying these methods to such problems may be future work.

Appendix

Solving the tridiagonal system for implicit FD

In the case of implicit scheme of finite difference, we need to solve the following tridiagonal system : $V^{n+1} = MV^n$ where

$$M = \begin{pmatrix} b_0 & a_0 & 0 & \cdots & \cdots & 0 \\ a_1 & b_1 & c_1 & 0 & \cdots & 0 \\ 0 & a_2 & b_2 & c_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots & \cdots & \vdots \\ 0 & \cdots & \cdots & 0 & a_{I-1} & b_{I-1} \end{pmatrix}$$

is a tridiagonal matrix.

A solution is to use the LU decomposition. That is we can write $M = L \cdot U$ where L is a lower triangular and U an upper triangular matrix. L and U are chosen as following :

$$L = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ l_1 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \cdots & \cdots & \vdots \\ 0 & \cdots & 0 & l_{I-1} & 1 \end{pmatrix} U = \begin{pmatrix} d_0 & u_0 & \cdots & \cdots & 0 \\ 0 & d_1 & u_1 & \cdots & \vdots \\ \vdots & \ddots & \cdots & \cdots & u_{I-2} \\ 0 & \cdots & \cdots & 0 & d_{I-1} \end{pmatrix}$$

which then reads $u_i = c_i$, $d_0 = b_0$, $l_i = a_i/d_{i-1}$ and $d_i = b_i - l_i u_{i-1}$.

We can now write our system as

$$V^{n+1} = MV^n = LUV^n = L \cdot (UV^n)$$

and solve first $L \cdot x = V^{n+1}$ then $U \cdot V^n = x$ which are triangular systems easier to solve.

Learning curve for Numerical application

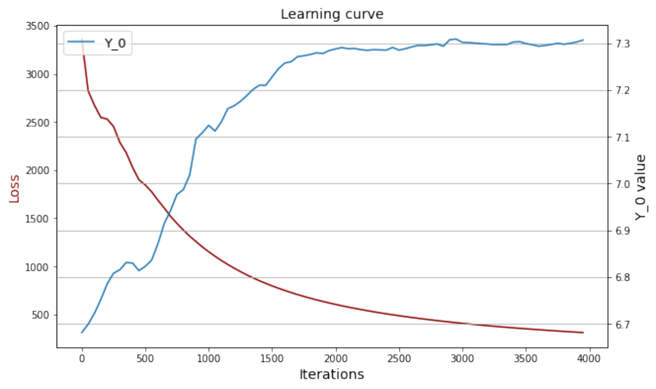

Figure 3: Learning curve for Deep BSDE scheme for $d = 10$ Basket option

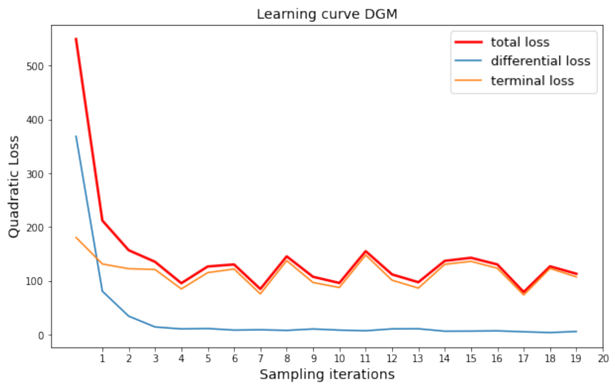

Figure 4: Learning curve of the DGM for $d = 10$ Basket option

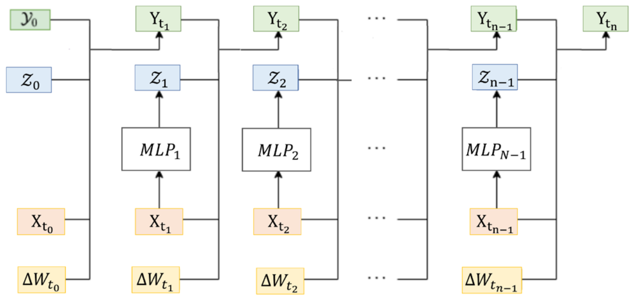

Figure 5: Architecture of the single network version of Deep BSDE

References

[1] O. Bokanowski T. Lelièvre Y. Achdou. Partial differential equations in finance. 2007.

[2] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. 2018.

[3] Nicolas Perkowski. Backward stochastic differential equations: an introduction. 2019.

[4] Jiequn Han Weinan E and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. 2017.

[5] Danilo Naiff Gabriel Jardim Ali Al-Aradi, Adolfo Correira. Solving nonlinear and high-dimensional partial differential equations via deep learning. 2018.

[6] Xavier Warin Côme Huré, Huyên Pham. Deep backward schemes for high-dimensional nonlinear pdes. 2020.

A propos de Coperneec

« From Revolution to Performance »

Coperneec est un cabinet de conseil cross-sectoriel spécialiste de la valorisation de la Data et du développement IT.

Nos méthodes et techniques scientifiques éprouvées permettent de résoudre des problématiques dans tous les secteurs de l'industrie.

Notre vocation : extraire la connaissance à partir des données et pérenniser les avancées technologiques qui en découlent. La R&D est au cœur de notre ADN et les expertises de nos consultants sont en permanence challengées afin d'accompagner au plus près les révolutions technologiques et scientifiques.

Contactez-nous

Aymeric LISBONNE
Partner
alisbonne@coperneec.com
06 88 69 67 75

coperneec

est une marque de

canopee group