Neural networks-based algorithms for stochastic control and PDEs in finance *

Maximilien Germain, Huyên Pham, Xavier Warin

▶ **To cite this version:**

Maximilien Germain, Huyên Pham, Xavier Warin. Neural networks-based algorithms for stochastic control and PDEs in finance *. 2021. hal-03115503v1

HAL Id: hal-03115503

https://hal.science/hal-03115503v1

Preprint submitted on 19 Jan 2021 (v1), last revised 15 Apr 2021 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Neural networks-based algorithms for stochastic control and PDEs in finance *

Maximilien Germain † Huyên Pham ‡ Xavier Warin §

January 19, 2021

Abstract

This paper presents machine learning techniques and deep reinforcement learning-based algorithms for the efficient resolution of nonlinear partial differential equations and dynamic optimization problems arising in investment decisions and derivative pricing in financial engineering. We survey recent results in the literature, present new developments, notably in the fully nonlinear case, and compare the different schemes illustrated by numerical tests on various financial applications. We conclude by highlighting some future research directions.

1 Breakthrough in the resolution of high dimensional nonlinear problems

The numerical resolution of control problems and nonlinear PDEs– arising in several financial applications such as portfolio selection, hedging, or derivatives pricing–is subject to the so-called "curse of dimensionality", making impractical the discretization of the state space in dimension greater than 3 by using classical PDE resolution methods such as finite differences schemes. Probabilistic regression Monte-Carlo methods based on a Backward Stochastic Differential Equation (BSDE) representation of semilinear PDEs have been developed in [Zha04], [BT04], [GLW05] to overcome this obstacle. These mesh-free techniques are successfully applied upon dimension 6 or 7, nevertheless, their use of regression methods requires a number of basis functions growing fastly with the dimension. What can be done to further increase the dimension of numerically solvable problems?

A breakthrough with deep learning based-algorithms has been made in the last five years towards this computational challenge, and we mention the recent survey by [Bec+20]. The main interest in the use of machine learning techniques for control and PDEs is the ability of deep neural networks to efficiently represent high dimensional functions without using spatial grids, and with no curse of dimensionality [Gro+18], [Hut+20]. Although the use of neural networks for solving PDEs is not new, see e.g. [DPT94], the approach has

*This paper is a contribution for the *Machine Learning for Financial Markets: a guide to contemporary practices*, Cambridge University Press, Editors: Agostino Capponi and Charles-Albert Lehalle. This study was supported by FiME (Finance for Energy Market Research Centre) and the "Finance et Développement Durable - Approches Quantitatives" EDF - CACIB Chair.

†EDF R&D, LPSM, Université de Paris mgermain at lpsm.paris

‡LPSM, Université de Paris, FiME, CREST ENSAE pham at lpsm.paris

§EDF R&D, FiME xavier.warin at edf.fr

been successfully revived with new ideas and directions. Neural networks have known a increasing popularity since the works on Reinforcement Learning for solving the game of Go by Google DeepMind teams. These empirical successes and the introduced methods allow to solve control problems in moderate or large dimension. Moreover, recently developed open source libraries like Tensorflow and Pytorch also offer an accessible framework to implement these algorithms.

A first natural use of neural networks for stochastic control concerns the discrete time setting, with the study of Markov Decision Processes, either in a brute force fashion or by using dynamic programming approaches. In the continuous time setting, and in the context of PDE resolution, we present various methods. A first kind of schemes is rather generic and can be applied to a variety of PDEs coming from a large range of applications. Other schemes rely on BSDE representations, strongly linked to stochastic control problems. In both cases, numerical evidence seems to indicate that the methods can be used in large dimension, greater than 10 and up to 1000 in certain studies. Some theoretical results also illustrates the convergence of specific algorithms. These advances pave the way for new methods dedicated to the study of large population games, studied in the context of mean field games and mean field control problems.

The outline of this article is the following. We first focus on some schemes for discrete time control in Section 2 before presenting generic machine learning schemes for PDEs in Subsection 3.1. Then we review BSDE-based machine learning methods for semilinear equations in Subsection 3.2.1. Existing algorithms for fully non-linear PDEs are detailed in Subsection 3.2.2 before presenting new BSDE schemes designed to treat this more difficult case. Numerical tests on CVA pricing and portfolio selection are conducted in Section 4 to compare the different approaches. Finally, we highlight in Section 5 further directions and perspectives including recent advances for the resolution of mean field games and mean field control problems with or without model.

2 Deep learning approach for stochastic control

We present in this section some recent breakthrough in the numerical resolution of stochastic control in high dimension by means of machine learning techniques. We consider a model-based setting in discrete-time, i.e., a Markov decision process, that could possibly be obtained from the time discretization of a continuous-time stochastic control problem.

Let us fix a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ equipped with a filtration $\mathbb{F} = (\mathcal{F}_t)_t$ representing the available information at any time $t \in \mathbb{N}$ ($\mathcal{F}_0$ is the trivial σ-algebra). The evolution of the system is described by a model dynamics for the state process $(X_t)_{t\in\mathbb{N}}$ valued in $\mathcal{X} \subset \mathbb{R}^d$:

$$X_{t+1} = F(X_t, \alpha_t, \varepsilon_{t+1}), \quad t \in \mathbb{N},$$

where $(\varepsilon_t)_t$ is a sequence of i.i.d. random variables valued in E, with ε_{t+1} $\mathcal{F}_{t+1}$-measurable containing all the noisy information arriving between t and $t+1$, and $\alpha = (\alpha_t)_t$ is the control process valued in $A \subset \mathbb{R}^q$. The dynamics function F is a measurable function from $\mathbb{R}^d \times \mathbb{R}^q \times E$ into $\mathbb{R}^d$, and assumed to be known. Given a running cost function f, a finite horizon $T \in \mathbb{N}^*$, and a terminal cost function, the problem is to minimize over control

process α a functional cost

$$J(\alpha) = \mathbb{E}\Big[\sum_{t=0}^{T-1} f(X_t, \alpha_t) + g(X_T)\Big].$$

In some relevant applications, we may require constraints on the state and control in the form: $(X_t, \alpha_t) \in \mathcal{S}$, $t \in \mathbb{N}$. for some subset $\mathcal{S}$ of $\mathbb{R}^d \times \mathbb{R}^q$. This can be handled by relaxing the state/constraint and introducing into the costs a penalty function $L(x,a)$: $f(x,a) \leftarrow f(x,a) + L(x,a)$, and $g(x) \leftarrow g(x) + L(x,a)$. For example, if the constraint set is in the form: $\mathcal{S} = \{(x,a) \in \mathbb{R}^d \times \mathbb{R}^q : h_k(x,a) = 0, k = 1, \ldots, p, h_k(x,a) \geq 0, k = p+1, \ldots, m\}$, then one can take as penalty functions:

$$L(x,a) = \sum_{k=1}^{p} \mu_k |h_k(x,a)|^2 + \sum_{k=p+1}^{m} \mu_k \max(0, -h_k(x,a)),$$

where μ_k are penalization parameters (large in practice).

2.1 Global approach

The method consists simply in approximating at any time t, the feedback control, i.e. a function of the state process, by a neural network (NN):

$$\alpha_t \simeq \pi^{\theta_t}(X_t), \quad t = 0, \ldots, T-1,$$

where π^θ is a feedforward neural network on $\mathbb{R}^d$ with parameters θ, and then to minimize over the global set of parameters $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{T-1})$ the quantity (playing the role of loss function)

$$\tilde{J}(\boldsymbol{\theta}) = \mathbb{E}\Big[\sum_{t=0}^{T-1} f(X_t^{\boldsymbol{\theta}}, \pi^{\theta_t}(X_t^{\boldsymbol{\theta}})) + g(X_T^{\boldsymbol{\theta}})\Big],$$

where $X^{\boldsymbol{\theta}}$ is the state process associated with the NN feedback controls:

$$X_{t+1}^{\boldsymbol{\theta}} = F(X_t^{\boldsymbol{\theta}}, \pi^{\theta_t}(X_t^{\boldsymbol{\theta}}), \varepsilon_{t+1}), \quad t = 0, \ldots, T-1.$$

This basic idea of approximating control by parametric function of the state was proposed in [GM05], and updated with the use of (deep) neural networks by [HE16]. This method met success due to its simplicity and the easy accessibility of common libraries like TensorFlow for optimizing the parameters of the neural networks. Some recent extensions of this approach dealt with stochastic control problems with delay, see [HH21]. However, such global optimization over a huge set of parameters $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{T-1})$ may suffer from being stuck in suboptimal traps and thus does not converge, especially for large horizon T.

2.2 Backward dynamic programming approach

In [Bac+19], the authors propose methods that combine ideas from numerical probability and deep reinforcement learning. Their algorithms are based on the classical dynamic programming (DP), (deep) neural networks for the approximation/learning of the optimal policy and value function, and Monte-Carlo regressions with performance and value iterations.

The first algorithm, called NNContPI, is a combination of dynamic programming and the approach in [HE16]. It learns sequentially the control by NN $\pi^\theta(.)$ and performance iterations, and is designed as follows:

Algorithm 1: NNContPI

Input: the training distributions $(\mu_t)_{t=0}^{T-1}$;
Output: estimates of the optimal strategy $(\hat{\pi}_t)_{t=0}^{T-1}$;
for $t = T-1, \dots, 0$ **do**
Compute $\hat{\pi}_t := \pi^{\hat{\theta}_t}$ with

$$\hat{\theta}_t \in \arg\min_\theta \mathbb{E}\Big[f\big(X_t, \pi^\theta(X_t)\big) + \sum_{s=t+1}^{T-1} f\big(X_s^\theta, \hat{\pi}_s(X_s^\theta)\big) + g\big(X_T^\theta\big)\Big]$$

where $X_t \sim \mu_t$ and where $\big(X_s^\theta\big)_{s=t+1}^T$ is defined by induction as:

$$\begin{cases} X_{t+1}^\theta &= F\big(X_t, \pi^\theta(X_t), \varepsilon_{t+1}\big), \\ X_{s+1}^\theta &= F\big(X_s^\theta, \hat{\pi}_s(X_s^\theta), \varepsilon_{s+1}\big), \quad \text{for } s = t+1, \dots, T-1. \end{cases}$$

The second algorithm, refereed to as Hybrid-Now, combines optimal policy estimation by neural networks and dynamic programming principle, and relies on an hybrid procedure between value and performance iteration to approximate the value function by neural network $\Phi^\eta(.)$ on $\mathbb{R}^d$ with parameters η.

Algorithm 2: Hybrid-Now

Input: the training distributions $(\mu_t)_{t=0}^{T-1}$;
Output:
– estimate of the optimal strategy $(\hat{\pi}_t)_{t=0}^{T-1}$;
– estimate of the value function $(\hat{V}_t)_{t=0}^{T-1}$;
Set $\hat{V}_T = g$;
for $t = T-1, \dots, 0$ **do**
Compute:

$$\hat{\theta}_t \in \arg\min_\theta \mathbb{E}\Big[f\big(X_t, \pi^\theta(X_t)\big) + \hat{V}_{t+1}(X_{t+1}^\theta)\Big]$$

where $X_t \sim \mu_t$, and $X_{t+1}^\theta = F(X_t, \pi^\theta(X_t), \varepsilon_{t+1})$;
Set $\hat{\pi}_t := \pi^{\hat{\theta}_t}$; ▷ $\hat{\pi}_t$ is the estimate of the optimal policy at time t
Compute

$$\hat{\eta}_t \in \arg\min_\eta \mathbb{E}\Big|f(X_t, \hat{\pi}_t(X_t)) + \hat{V}_{t+1}(X_{t+1}^{\hat{\theta}_t}) - \Phi^\eta(X_t)\Big|^2.$$

Set $\hat{V}_t = \Phi^{\hat{\eta}_t}$; ▷ $\hat{V}_t$ is the estimate of the value function at time t

The convergence analysis of Algorithms NNContPI and Hybrid-Now are studied in [Hur+18], and various applications in finance are implemented in [Bac+19]. These algorithms are well-designed for control problems with continuous control space $A = \mathbb{R}^q$ or a ball in $\mathbb{R}^q$. In the case where the control space A is finite, it is relevant to randomize

controls, and then use classification methods by approximating the distribution of controls with neural networks and Softmax activation function.

3 Machine learning algorithms for nonlinear PDEs

3.1 Deterministic approach by neural networks

In the schemes below, differential operators are evaluated by automatic differentiation of the network function approximating the solution of the PDE. Machine learning libraries such as Tensorflow or Pytorch allow to efficiently compute these derivatives.

• Deep Galerkin Method [SS17].

The Deep Galerkin Method is a meshfree machine learning algorithm to solve PDEs on a domain, eventually with boundary conditions. The principle is to sample time and space points according to a training measure, e.g. uniform on a bounded domain, and minimize a performance measure quantifying how well a neural network satisfies the differential operator and boundary conditions. For instance, when the studied PDE problem is

$$\begin{cases} \partial_t u + \mathcal{F} u \;=\; 0 & \text{on } [0,T) \times \Lambda \\ u(0,\cdot) \;=\; g & \text{on } \Lambda \\ u(t,x) \;=\; h(t,x) & \text{on } [0,T) \times \partial\Lambda. \end{cases}$$

with $\mathcal{F}$ a space differential operator, Λ a subset of $\mathbb{R}^d$, the method consists in minimizing over neural network $\mathcal{U} : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$, the L^2 loss

$$\mathbb{E}|\partial_t \mathcal{U}(\tau,\kappa) + \mathcal{F}\mathcal{U}(\tau,\kappa)|^2 + \mathbb{E}|\mathcal{U}(0,\xi) - g(\xi)|^2 + \mathbb{E}|\mathcal{U}(\tau,\kappa) - h(\tau,\kappa)|^2$$

with κ, τ, ξ independent random variables in $\Lambda \times [0,T] \times \partial\Lambda$. This method is tested on financial problems by [AA+18]. A major advantage to this method is its adaptability to a large range of PDEs with or without boundary conditions. Indeed the loss function is straightforwardly modified according to changes in the constraints one wishes to enforce on the PDE solution. A related approach is the deep parametric PDE method, see [KJY20], and [GW20] applied to option pricing. Extension to path-dependent PDEs is developed in [SZ20].

• Other approximation methods

(i) *Physics informed neural networks* [RPK19]. Physics informed neural networks use both data (obtained for a limited amount of samples from a PDE solution), and theoretical dynamics to reconstruct solutions from PDEs.

(ii) *Deep Ritz method* [EY18]. The Deep Ritz method focuses on the resolution of the variational formulation from elliptic problems where the integral is evaluated by randomly sampling time and space points, like in the Deep Galerkin method [SS17] and the minimization is performed over the parameters of a neural network. This scheme is tested on Poisson equation with different types of boundary conditions.

3.2 Probabilistic approach by neural networks

3.2.1 Semi-linear case

In this paragraph, we consider semilinear PDEs of the form

$$\begin{cases} \partial_t u + \mu \cdot D_x u + \frac{1}{2}\,\mathrm{Tr}(\sigma\sigma^\intercal D_x^2 u) \;=\; f(\cdot,\cdot,u,\sigma^\intercal D_x u) & \text{on } [0,T)\times\mathbb{R}^d \\ u(T,\cdot) \;=\; g & \text{on } \mathbb{R}^d. \end{cases} \tag{3.1}$$

for which we have the forward backward SDE representation

$$\begin{cases} Y_t &=& g(\mathcal{X}_T) - \int_t^T f(s,\mathcal{X}_s,Y_s,Z_s)\mathrm{d}s - \int_t^T Z_s\cdot \mathrm{d}W_s, \quad 0\le t\le T, \\ \mathcal{X}_t &=& \mathcal{X}_0 + \int_0^t \mu(s,\mathcal{X}_s)\mathrm{d}s + \int_0^t \sigma(s,\mathcal{X}_s)\mathrm{d}W_s, \end{cases} \tag{3.2}$$

via the (non-linear) Feynman-Kac formula: $Y_t = v(t,X_t)$, $Z_t = \sigma^\intercal(t,\mathcal{X}_t)D_x v(t,X_t)$, $0\le t\le T$, see [PP90].

Let π be a subdivision $\{t_0 = 0 < t_1 < \cdots < t_N = T\}$ with modulus $|\pi| := \sup_i \Delta t_i$, $\Delta t_i := t_{i+1} - t_i$, satisfying $|\pi| = O\left(\frac{1}{N}\right)$, and consider the Euler-Maruyama discretization $(X_i)_{i=0,\dots,N}$ defined by

$$X_i = \mathcal{X}_0 + \sum_{j=0}^{i-1}\mu(t_j,X_j)\Delta t_j + \sum_{j=0}^{i-1}\sigma(t_j,X_j)\Delta W_j,$$

where $\Delta W_j := W_{t_{j+1}} - W_{t_j}$, $j = 0,\dots,N$. Sample paths of $(X_i)_i$ act as training data in the machine learning setting. Thus our training set can be chosen as large as desired, which is relevant for training purposes as sit does not lead to overfitting.

The time discretization of the BSDE (3.2) can be written in backward induction as

$$Y_i^\pi = \; Y_{i+1}^\pi - f(t_i,X_i,Y_i^\pi,Z_i^\pi)\Delta t_i - Z_i^\pi.\Delta W_i, \quad i = 0,\dots,N-1, \tag{3.3}$$

which can be described as conditional expectation formulae

$$\begin{cases} Y_i^\pi &=& \mathbb{E}_i\Big[Y_{i+1}^\pi - f(t_i,X_i,Y_i^\pi,Z_i^\pi)\Delta t_i\Big] \\ Z_i^\pi &=& \mathbb{E}_i\Big[\frac{\Delta W_i}{\Delta t_i}Y_{i+1}^\pi\Big], \qquad i = 0,\dots,N-1, \end{cases} \tag{3.4}$$

where $\mathbb{E}_i$ is a notation for the conditional expectation w.r.t. $\mathcal{F}_{t_i}$.

• Deep BSDE scheme [EHJ17], [HJE17].

The essence of this method is to write down the backward equation (3.3) as a forward equation. One approximates the initial condition Y_0 and the Z component at each time by networks functions taking the forward process X as input. The objective function to optimize is the error between the reconstructed dynamics and the true terminal condition. More precisely, the problem is to minimize over network functions $\mathcal{U}_0 : \mathbb{R}^d \to \mathbb{R}$, and sequences of network functions $\boldsymbol{\mathcal{Z}} = (\mathcal{Z}_i)_i$, $\mathcal{Z}_i : \mathbb{R}^d \to \mathbb{R}^d$, $i = 0,\dots,N-1$, the global quadratic loss function

$$J_G(\mathcal{U}_0,\boldsymbol{\mathcal{Z}}) = \; \mathbb{E}\Big|Y_N^{\mathcal{U}_0,\boldsymbol{\mathcal{Z}}} - g(X_N)\Big|^2,$$

where $(Y_i^{\mathcal{U}_0,\boldsymbol{\mathcal{Z}}})_i$ is defined by forward induction as

$$Y_{i+1}^{\mathcal{U}_0,\boldsymbol{\mathcal{Z}}} \;=\; Y_i^{\mathcal{U}_0,\boldsymbol{\mathcal{Z}}} + f(t_i,X_i,Y_i^{\mathcal{U}_0,\boldsymbol{\mathcal{Z}}},\mathcal{Z}_i(X_i))\Delta t_i + \mathcal{Z}_i(X_i).\Delta W_i,\; i = 0,\dots,N-1,$$

starting from $Y_0^{\mathcal{U}_0,\boldsymbol{\mathcal{Z}}} = \mathcal{U}_0(\mathcal{X}_0)$. The output of this scheme, for the solution $(\widehat{\mathcal{U}}_0, \widehat{\boldsymbol{\mathcal{Z}}})$ to this global minimization problem, supplies an approximation $\widehat{\mathcal{U}}_0$ of the solution $u(0,.)$ to the PDE at time 0, and approximations $Y_i^{\widehat{\mathcal{U}}_0,\widehat{\boldsymbol{\mathcal{Z}}}}$ of the solution to the PDE (3.1) at times t_i evaluated at $\mathcal{X}_{t_i}$, i.e., of $Y_{t_i} = u(t_i, \mathcal{X}_{t_i})$, $i = 0, \ldots, N$. The convergence of this algorithm through a posteriori error is studied by [HL20]. A variant is proposed by [CWNMW19] which introduces a single neural network $\mathcal{Z}(t,x) : [0,T] \times \mathbb{R}^d \mapsto \mathbb{R}^d$ instead of N independent neural networks. This simplifies the optimization problem and leads to more stable solutions. A close method introduced by [Rai18] uses also a single neural network $\mathcal{U}(t,x) : [0,T] \times \mathbb{R}^d \mapsto \mathbb{R}$ and estimates Z as the automatic derivative in space of $\mathcal{U}$. We also refer to [JO19] for a variation of this deep BSDE scheme to curve-dependent PDEs arising in the pricing under rough volatility model, to [GPR20] for a development of deep BSDE method to XVA solver, to [NR19] for approximations methods for Hamilton-Jacobi-Bellman PDEs, to [KSS20] for extension of deep BSDE scheme to elliptic PDEs with applications in insurance, and to [Ji+20] for the resolution of PDEs associated to fully coupled forward-backward SDEs.

• Deep Backward Dynamic Programming (DBDP) [HPW20].

The method builds upon the backward dynamic programming relation (3.3) stemming from the time discretization of the BSDE, and approximates simultaneously at each time step t_i the processes (Y_{t_i}, Z_{t_i}) with neural networks trained with the forward diffusion process X_i as input. The scheme can be implemented in two similar versions:

1. *DBDP1.* Starting from $\widehat{\mathcal{U}}_N^{(1)} = g$, proceed by backward induction for $i = N-1, \ldots, 0$, by minimizing over network functions $\mathcal{U}_i : \mathbb{R}^d \to \mathbb{R}$, and $\mathcal{Z}_i : \mathbb{R}^d \to \mathbb{R}^d$ the quadratic loss function

$$\begin{aligned} & J_i^{(B1)}(\mathcal{U}_i, \mathcal{Z}_i) \\ = \;& \mathbb{E}\Big|\widehat{\mathcal{U}}_{i+1}^{(1)}(X_{i+1}) - \mathcal{U}_i(X_i) - f(t_i, X_i, \mathcal{U}_i(X_i), \mathcal{Z}_i(X_i))\Delta t_i - \mathcal{Z}_i(X_i).\Delta W_i\Big|^2, \end{aligned}$$

   and update $(\widehat{\mathcal{U}}_i^{(1)}, \widehat{\mathcal{Z}}_i^{(1)})$ as the solution to this local minimization problem.

2. *DBDP2.* Starting from $\widehat{\mathcal{U}}_N^{(2)} = g$, proceed by backward induction for $i = N-1, \ldots, 0$, by minimizing over C^1 network functions $\mathcal{U}_i : \mathbb{R}^d \to \mathbb{R}$ the quadratic loss function

$$\begin{aligned} & J_i^{(B2)}(\mathcal{U}_i) \\ = \;& \mathbb{E}\Big|\widehat{\mathcal{U}}_{i+1}^{(2)}(X_{i+1}) - \mathcal{U}_i(X_i) - f(t_i, X_i, \mathcal{U}_i(X_i), \sigma(t_i, X_i)^\intercal D_x\mathcal{U}_i(X_i))\Delta t_i \\ & - \; D_x\mathcal{U}_i(X_i)^\intercal \sigma(t_i, X_i)\Delta W_i\Big|^2, \end{aligned}$$

   where $D_x\mathcal{U}_i$ is the automatic differentiation of the network function $\mathcal{U}_i$. Update $\widehat{\mathcal{U}}_i^{(2)}$ as the solution to this local minimization problem, and set $\widehat{\mathcal{Z}}_i^{(2)} = \sigma^\intercal(t_i,.)D_x\mathcal{U}_i^{(2)}$.

The output of DBDP supplies an approximation $(\widehat{\mathcal{U}}_i, \widehat{\mathcal{Z}}_i)$ of the solution $u(t_i,.)$ and its gradient $\sigma^\intercal(t_i,.)D_x u(t_i,.)$ to the PDE (3.1) on the time grid t_i, $i = 0, \ldots, N-1$. The study of the approximation error due to the time discretization and the choice of the loss function is accomplished in [HPW20].

Variants and extensions of DBDP schemes

(i) A regression-based machine learning scheme inspired by regression Monte-Carlo methods for numerically computing condition expectations in the time discretization (3.4) of the BSDE, is given by: starting from $\hat{\mathcal{U}}_N = g$, proceed by backward induction for $i = N-1, \dots, 0$, in two regression problems:

(a) Minimize over network functions $\mathcal{Z}_i : \mathbb{R}^d \to \mathbb{R}^d$

$$J_i^{r,Z}(\mathcal{Z}_i) = \mathbb{E}\Big|\frac{\Delta W_i}{\Delta t_i}\widehat{\mathcal{U}}_{i+1}(X_{i+1}) - \mathcal{Z}_i(X_i)\Big|^2$$

and update $\widehat{\mathcal{Z}}_i$ as the solution to this minimization problem

(b) Minimize over network functions $\mathcal{U}_i : \mathbb{R}^d \to \mathbb{R}$

$$J_i^{r,Y}(\mathcal{U}_i) = \mathbb{E}\Big|\widehat{\mathcal{U}}_{i+1}(X_{i+1}) - \mathcal{U}_i(X_i) - f(t_i, X_i, \mathcal{U}_i(X_i), \widehat{\mathcal{Z}}_i(X_i))\Delta t_i\Big|^2$$

and update $\widehat{\mathcal{U}}_i$ as the solution to this minimization problem.

Compared to these regression-based schemes, the DBDP scheme simultaneously estimates the pair component (Y, Z) through the minimization of the loss functions $J_i^{(B1)}(\mathcal{U}_i, \mathcal{Z}_i)$ (or $J_i^{(B2)}(\mathcal{U}_i)$ for the second version), $i = N-1, \dots, 0$. Interestingly, the convergence of the DBDP scheme can be confirmed by computing at each time step the infimum of loss function, which should vanish for the exact solution (up to the time discretization). In contrast, the infimum of the loss functions in usual regression-based schemes is unknown for the true solution as it is supposed to match the residual of L^2-projection. Therefore the scheme accuracy cannot be directly verified.

(ii) The DBDP scheme is based on local resolution, and was first used to solve linear PDEs, see [SVSS18]. It is also suitable to solve variational inequalities and can be used to valuate American options as shown in [HPW20]. Alternative methods consists in using the Deep Optimal Stopping scheme [BCJ19] or the method from [Bec+19b]. Some tests on Bermudan options are also performed by [LXL19] and [FTT19] with some refinements of the Deep BSDE scheme.

(iii) The **Deep Splitting (DS) scheme in [Bec+19a]** combines ideas from the DBDP2 and regression-based schemes. Indeed the current regression-approximation on Z is estimated by the automatic differentiation of the neural network computed at the previous optimization step. The current approximation of Y is then computed by a regression-type optimization problem. It can be seen as a local version of the global algorithm from [Rai18] or as a step by step Feynman-Kac approach. As the scheme is a local one, it can be used to valuate American options. The convergence of this method is studied by [GPW20].

(iv) Local resolution permits to add other constraints such as constraints on a replication portfolio using facelifting techniques as in [KLW20].

(v) The **Deep Backward Multistep (MDBDP) scheme [GPW20]** is described as follows: for $i = N-1, \dots, 0$, minimize over network functions $\mathcal{U}_i : \mathbb{R}^d \to \mathbb{R}$, and $\mathcal{Z}_i$:

$\mathbb{R}^d \to \mathbb{R}^d$ the loss function

$$
\begin{aligned}
&J_i^{MB}(\mathcal{U}_i, \mathcal{Z}_i) \\
&= \mathbb{E}\Big| g(X_N) - \sum_{j=i+1}^{N-1} f(t_j, X_j, \widehat{\mathcal{U}}_j(X_j), \widehat{\mathcal{Z}}_j(X_j))\Delta t_j - \sum_{j=i+1}^{N-1} \widehat{\mathcal{Z}}_j(X_j).\Delta W_j \\
&\qquad - \mathcal{U}_i(X_i) - f(t_i, X_i, \mathcal{U}_i(X_i), \mathcal{Z}_i(X_i))\Delta t_i - \mathcal{Z}_i(X_i).\Delta W_i \Big|^2
\end{aligned}
$$

and update $(\widehat{\mathcal{U}}_i, \widehat{\mathcal{Z}}_i)$ as the solution to this minimization problem. This output provides an approximation $(\widehat{\mathcal{U}}_i, \widehat{\mathcal{Z}}_i)$ of the solution $u(t_i, .)$ to the PDE (3.1) at times t_i, $i = 0, \ldots, N-1$.

MDBDP is a machine learning version of the Multi-step Forward Dynamic Programming method studied by [BD07] and [GT14]. Instead of solving at each time step two regression problems, our approach allows to consider only a single minimization as in the DBDP scheme. Compared to the latter, the multi-step consideration is expected to provide better accuracy by reducing the propagation of errors in the backward induction as it can be shown comparing the error estimated in [GPW20] and [HPW20] both at theoretical and numerical level.

3.2.2 Case of fully non-linear PDEs

In this paragraph, we consider fully non-linear PDEs in the form

$$
\begin{cases}
\partial_t u + \mu \cdot D_x u + \frac{1}{2}\,\mathrm{Tr}(\sigma\sigma^\intercal D_x^2 u) \;=\; F(\cdot, \cdot, u, D_x u, D_x^2 u) & \text{on } [0,T) \times \mathbb{R}^d \\
u(T, \cdot) \;=\; g & \text{on } \mathbb{R}^d,
\end{cases}
\tag{3.5}
$$

For this purpose, we introduce a forward diffusion process $\mathcal{X}$ in $\mathbb{R}^d$ as in (3.2), and associated to the linear part $\mathcal{L}$ of the differential operator in the l.h.s. of the PDE (3.5). Since the function F contains the dependence both on the gradient $D_x u$ and the Hessian $D_x^2 u$, we can shift the linear differential operator (left hand side) of the PDE (3.5) into the function F. However, in practice, this linear differential operator associated to a diffusion process $\mathcal{X}$ is used for training simulations in SGD of machine learning schemes. We refer to Section 3.1 in [PWG19] for a discussion on the choice of the parameters μ, σ. In the sequel, we assume for simplicity that $\mu = 0$, and σ is a constant invertible matrix.

Let us derive formally a BSDE representation for the nonlinear PDE (3.5) on which we shall rely for designing our machine learning algorithm. Assuming that the solution u to this PDE is smooth C^2, and denoting by (Y, Z, Γ) the triple of $\mathbb{F}$-adapted processes valued in $\mathbb{R} \times \mathbb{R}^d \times \mathbb{S}^d$, defined by

$$
Y_t \;=\; u(t, \mathcal{X}_t), \quad Z_t \;=\; D_x u(t, \mathcal{X}_t), \quad \Gamma_t \;=\; D_x^2 u(t, \mathcal{X}_t), \quad 0 \le t \le T,
$$

a direct application of Itô's formula to $u(t, \mathcal{X}_t)$, yields that (Y, Z, Γ) satisfies the backward equation

$$
Y_t = \; g(\mathcal{X}_T) - \int_t^T F(s, \mathcal{X}_s, Y_s, Z_s, \Gamma_s)\mathrm{d}s - \int_t^T Z_s^\intercal \sigma \mathrm{d}W_s, \; 0 \le t \le T. \tag{3.6}
$$

Compared to the case of semi-linear PDE of the form (3.1), the key point is the approximation/learning of the Hessian matrix $D_x^2 u$, hence of the Γ-component of the BSDE (3.6). We present below different approaches for the approximation of the Γ-component.

• Deep 2BSDE scheme [BEJ19].

This scheme relies on the 2BSDE representation of [Che+07]

$$\left\{ \begin{array}{rcl} Y_t &=& g(\mathcal{X}_T) - \int_t^T F(s, \mathcal{X}_s, Y_s, Z_s, \Gamma_s)\mathrm{d}s - \int_t^T Z_s^\intercal \sigma \mathrm{d}W_s, \\ Z_t &=& D_x g(\mathcal{X}_T) - \int_t^T A_s \mathrm{d}s - \int_t^T \Gamma_s \sigma \mathrm{d}W_s, \quad 0 \le t \le T, \end{array} \right. \tag{3.7}$$

with $A_t = \mathcal{L} D_x u(t, \mathcal{X}_t)$. The idea is to adapt the Deep BSDE algorithm to the fully non-linear case. Again, we treat the backward system (3.7) as a forward equation by approximating the initial conditions Y_0, Z_0 and the A, Γ components of the 2BSDE at each time by networks functions taking the forward process $\mathcal{X}$ as input, and aiming to match the terminal condition.

• Second order DBDP (2DBDP) [PWG19].

The basic idea is to adapt the DBDP scheme by approximating the solution u and its gradient $D_x u$ by network functions $\mathcal{U}$ and $\mathcal{Z}$, and then Hessian $D_x^2 u$ by the automatic differentiation $D_x \mathcal{Z}$ of the network function $\mathcal{Z}$ (or double automatic differentiation $D_x^2 \mathcal{U}$ of the network function $\mathcal{U}$), via a learning approach relying on the time discretization of the BSDE (3.6). It turns out that such method approximates poorly Γ inducing instability of the scheme: indeed, while the unique pair solution (Y, Z) to classical BSDEs (3.2) completely characterizes the solution to the related semilinear PDE and its gradient, the relation (3.6) does not allow to characterize directly the triple (Y, Z, Γ). This approach was proposed and tested in [PWG19] where the automatic differentiation is performed on the previous value of $\mathcal{Z}$ with a truncation $\mathcal{T}$ which allows to reduce instabilities.

• Second Order Multistep schemes.

To overcome the instability in the approximation of the Γ-component in the Second order DBDP scheme, we propose a finer approach based on a suitable probabilistic representation of the Γ-component for learning accurately the Hessian function $D_x^2 u$ by using also Malliavin weights. We start from the training simulations of the forward process $(X_i)_i$ on the uniform grid $\pi = \{t_i = i|\pi|, i = 0, \ldots, N\}$, $|\pi| = T/N$, and notice that $X_i = \mathcal{X}_{t_i}$, $i = 0, \ldots, N$ as μ and σ are constants. The approximation of the value function u and its gradient $D_x u$ is learnt simultaneously on the grid π but requires in addition a preliminary approximation of the Hessian $D_x^2 u$ in the fully non-linear case. This will be performed by regression-based machine learning scheme on a subgrid $\hat{\pi} \subset \pi$, which allows to reduce the computational time of the algorithm.

We propose three versions of second order MDBDP based on different representations of the Hessian function. For the second and the third one, we need to introduce a subgrid $\hat{\pi} = \{t_{\hat{\kappa}\ell}, \ell = 0, \ldots, \hat{N}\} \subset \pi$, of modulus $|\hat{\pi}| = \hat{\kappa}|\pi|$, for some $\hat{\kappa} \in \mathbb{N}^*$, with $N = \hat{\kappa}\hat{N}$.

- *Version 1:* Extending the methodology introduced in [PWG19], the current Γ-component at step i can be estimated by automatic differentiation of the Z-component at the previous step while the other Γ-components are estimated by automatic differentiation of their associated Z-components:

$$\Gamma_i \;\simeq\; D_x Z_{i+1}, \quad \Gamma_j \;\simeq\; D_x Z_j, \; j > i.$$

- *Version 2:* The time discretization of (3.6) on the time grid $\hat{\pi}$, where $(Y_\ell^{\hat{\pi}}, Z_\ell^{\hat{\pi}}, \Gamma_\ell^{\hat{\pi}})$ denotes an approximation of the triple

$$\big(u(t_{\hat{\kappa}\ell}, X_{\hat{\kappa}\ell}), D_x u(t_{\hat{\kappa}\ell}, X_{\hat{\kappa}\ell}), D_x^2 u(t_{\hat{\kappa}\ell}, X_{\hat{\kappa}\ell})\big), \quad \ell = 0, \ldots, \hat{N},$$

leads to the standard representation formula for the Z component:

$$Z_\ell^{\hat\pi} = \mathbb{E}_{\hat\kappa\ell}\Big[Y_{\ell+1}^{\hat\pi}\hat H_\ell^1\Big], \quad \ell = 0,\dots,\hat N-1,$$

(recall that $\mathbb{E}_{\hat\kappa\ell}$ denotes the conditional expectation w.r.t. $\mathcal{F}_{t_{\hat\kappa\ell}}$), with the Malliavin weight of order one:

$$\hat H_\ell^1 = (\sigma^\intercal)^{-1}\frac{\hat\Delta W_\ell}{|\hat\pi|}, \quad \hat\Delta W_\ell := W_{t_{\hat\kappa(\ell+1)}} - W_{t_{\hat\kappa\ell}}.$$

By direct differentiation, we then obtain an approximation of the Γ component as

$$\Gamma_\ell^{\hat\pi} \simeq \mathbb{E}_{\hat\kappa\ell}\Big[D_x u(t_{\hat\kappa(\ell+1)}, X_{\hat\kappa(\ell+1)})\hat H_\ell^1\Big].$$

Moreover, by introducing the antithetic variable

$$\hat X_{\hat\kappa(\ell+1)} = X_{\hat\kappa\ell} - \sigma\hat\Delta W_\ell,$$

we then propose the following regression estimator of $D_x^2 u$ on the grid $\hat\pi$ for $\ell = 0,\dots,\hat N-1$ with

$$\begin{cases} \hat\Gamma^{(1)}(t_{\hat\kappa\hat N}, X_{\hat\kappa\hat N}) &= D^2 g(X_{\hat\kappa\hat N}) \\ \hat\Gamma^{(1)}(t_{\hat\kappa\ell}, X_{\hat\kappa\ell}) &= \mathbb{E}_{\hat\kappa\ell}\Big[\frac{D_x u(t_{\hat\kappa(\ell+1)}, X_{\hat\kappa(\ell+1)}) - D_x u(t_{\hat\kappa(\ell+1)}, \hat X_{\hat\kappa(\ell+1)})}{2}\hat H_\ell^1\Big]. \end{cases}$$

- *Version 3:* Alternatively, the time discretization of (3.6) on $\hat\pi$ yields the iterated conditional expectation relation:

$$Y_\ell^{\hat\pi} = \mathbb{E}_{\hat\kappa\ell}\Big[g(X_{\hat\kappa\hat N}) - |\hat\pi|\sum_{m=\ell}^{\hat N-1} F(t_{\hat\kappa m}, X_{\hat\kappa m}, Y_m^{\hat\pi}, Z_m^{\hat\pi}, \Gamma_m^{\hat\pi})\Big], \quad \ell = 0,\dots,\hat N,$$

By (double) integration by parts, and using Malliavin weights on the Gaussian vector X, we obtain a multistep approximation of the Γ-component:

$$\Gamma_\ell^{\hat\pi} \simeq \mathbb{E}_{\hat\kappa\ell}\Big[g(X_{\hat\kappa\hat N})\hat H_{\ell,\hat N}^2 - |\hat\pi|\sum_{m=\ell+1}^{\hat N-1} F(t_{\hat\kappa m}, X_{\hat\kappa m}, Y_m^{\hat\pi}, Z_m^{\hat\pi}, \Gamma_m^{\hat\pi})\hat H_{\ell,m}^2\Big],$$

for $\ell = 0,\dots,\hat N$, where

$$\hat H_{\ell,m}^2 = (\sigma^\intercal)^{-1}\frac{\hat\Delta W_\ell^m(\hat\Delta W_\ell^m)^\intercal - (m-\ell)|\hat\pi| I_d}{(m-\ell)^2|\hat\pi|^2}\sigma^{-1}, \ \hat\Delta W_\ell^m := W_{t_{\hat\kappa m}} - W_{t_{\hat\kappa\ell}}.$$

By introducing again the antithetic variables

$$\hat X_{\hat\kappa m} = X_{\hat\kappa\ell} - \sigma\hat\Delta W_\ell^m, \quad m = \ell+1,\dots,\hat N,$$

we then propose another regression estimator of $D_x^2 u$ on the grid $\hat\pi$ with

$$\begin{aligned} &\hat\Gamma^{(2)}(t_{\hat\kappa\ell}, X_{\hat\kappa\ell}) \\ &= \mathbb{E}_{\hat\kappa\ell}\Big[\frac{g(X_{\hat\kappa\hat N}) + g(\hat X_{\hat\kappa\hat N})}{2}\hat H_{\ell,\hat N}^2 \\ &- \frac{|\hat\pi|}{2}\sum_{m=\ell+1}^{\hat N-1}\Big(F\big(t_{\hat\kappa m}, X_{\hat\kappa m}, u(t_{\hat\kappa m}, X_{\hat\kappa m}), D_x u(t_{\hat\kappa m}, X_{\hat\kappa m}), \hat\Gamma^{(2)}(t_{\hat\kappa m}, X_{\hat\kappa m})\big) \\ &+ F\big(t_{\hat\kappa m}, \hat X_{\hat\kappa m}, u(t_{\hat\kappa m}, \hat X_{\hat\kappa m}), D_x u(t_{\hat\kappa m}, \hat X_{\hat\kappa m}), \hat\Gamma^{(2)}(t_{\hat\kappa m}, \hat X_{\hat\kappa m})\big) \\ &- 2F\big(t_{\hat\kappa\ell}, X_{\hat\kappa\ell}, u(t_{\hat\kappa\ell}, X_{\hat\kappa\ell}), D_x u(t_{\hat\kappa\ell}, X_{\hat\kappa\ell}), \hat\Gamma^{(2)}(t_{\hat\kappa\ell}, X_{\hat\kappa\ell})\big)\Big)\hat H_{\ell,m}^2\Big], \end{aligned}$$

for $\ell = 0, \dots, N-1$, and $\hat{\Gamma}^{(2)}(t_{\hat{\kappa}\hat{N}}, X_{\hat{\kappa}\hat{N}}) = D^2 g(X_{\hat{\kappa}\hat{N}})$. The correction term $-2F$ evaluated at time $t_{\hat{\kappa}\ell}$ in $\hat{\Gamma}^{(2)}(t_{\hat{\kappa}\ell}, X_{\hat{\kappa}\ell})$ does not add bias since

$$\mathbb{E}_{\hat{\kappa}\ell}\Big[F\big(t_{\hat{\kappa}\ell}, X_{\hat{\kappa}\ell}, u(t_{\hat{\kappa}\ell}, X_{\hat{\kappa}\ell}), D_x u(t_{\hat{\kappa}\ell}, X_{\hat{\kappa}\ell}), \hat{\Gamma}^{(2)}(t_{\hat{\kappa}\ell}, X_{\hat{\kappa}\ell})\big)\hat{H}^2_{\ell,m}\Big] = 0,$$

for all $m = \ell+1, \dots, \hat{N}-1$, and by Taylor expansion of F at second order, we see that it allows together with the antithetic variable to control the variance when the time step goes to zero.

Remark 3.1. In the case where the function g has some regularity property, one can avoid the integration by parts at the terminal data component in the above expression of $\hat{\Gamma}^{(2)}$. For example, when g is C^1, $\frac{g(X_{\hat{\kappa}\hat{N}})+g(\hat{X}_{\hat{\kappa}\hat{N}})}{2}\hat{H}^2_{\ell,\hat{N}}$ is alternatively replaced in $\hat{\Gamma}^{(2)}$ expression by $(Dg(X_{\hat{\kappa}\hat{N}}) - Dg(\hat{X}_{\hat{\kappa}\hat{N}}))\hat{H}^1_{\ell,\hat{N}}$, while when it is C^2 it is replaced by $D^2 g(X_{\hat{\kappa}\hat{N}})$. □

Remark 3.2. We point out that in our machine learning setting for the versions 2 and 3 of the scheme, we only solve two optimization problems by time step instead of three as in [FTW11]. One optimization is dedicated to the computation of the Γ component but the $\mathcal{U}$ and $\mathcal{Z}$ components are simultaneously learned by the algorithm. □

We can now describe the three versions of second order MDBDP schemes for the numerical resolution of the fully non-linear PDE (3.5). We emphasize that these schemes do not require *a priori* that the solution to the PDE is smooth.

Algorithm 3: Second order Explicit Multistep DBDP (2EMDBDP)

for $i = N-1, \dots, 0$ **do**

If $i = N-1$, update $\widehat{\Gamma}_i = D^2 g$, otherwise $\widehat{\Gamma}_i = D_x \widehat{\mathcal{Z}}_{i+1}$, $\widehat{\Gamma}_j = D_x \widehat{\mathcal{Z}}_j$, $j \in [\![i+1, N-1]\!]$, `/* Update Hessian */`

Minimize over network functions $\mathcal{U} : \mathbb{R}^d \to \mathbb{R}$, and $\mathcal{Z} : \mathbb{R}^d \to \mathbb{R}^d$ the loss function at time t_i:

$$\begin{aligned} & J_i^{MB}(\mathcal{U}, \mathcal{Z}) \\ &= \mathbb{E}\Big| g(X_N) - |\pi| \sum_{j=i+1}^{N-1} F(t_j, X_j, \widehat{\mathcal{U}}_j(X_j), \widehat{\mathcal{Z}}_j(X_j), \hat{\Gamma}_j(X_j)) \\ &\qquad - \sum_{j=i+1}^{N-1} \widehat{\mathcal{Z}}_j(X_j)^\intercal \sigma \Delta W_j - \mathcal{U}(X_i) \\ &\qquad - |\pi| F(t_i, X_i, \mathcal{U}(X_i), \mathcal{Z}(X_i), \widehat{\Gamma}_i(X_{i+1})) - \mathcal{Z}(X_i) \cdot \sigma \Delta W_i \Big|^2. \end{aligned}$$

Update $(\widehat{\mathcal{U}}_i, \widehat{\mathcal{Z}}_i)$ as the solution to this minimization problem `/* Update the function and its derivative */`

end

The proposed algorithms 3, 4, 5 are in backward iteration, and involve one optimization at each step. Moreover, as the computation of Γ requires a further derivation for Algorithms 4 and 5, we may expect that the additional propagation error varies according to $\frac{|\pi|}{|\hat{\pi}|} = \frac{1}{\hat{\kappa}}$, and thus the convergence of the scheme when $\hat{\kappa}$ is large. In the numerical implementation, the expectation in the loss functions are replaced by empirical average and the minimization over network functions is performed by stochastic gradient descent.

Algorithm 4: Second order Multistep DBDP (2MDBDP)

for $\ell = \hat{N}, \dots, 0$ **do**

If $\ell = \hat{N}$, update $\widehat{\Gamma}_\ell = D^2 g$, otherwise minimize over network functions $\Gamma : \mathbb{R}^d \to \mathbb{S}^d$ the loss function

$$\mathcal{J}_\ell^{1,M}(\Gamma) = \mathbb{E}\Big|\Gamma(X_{\hat{\kappa}\ell}) - \frac{\widehat{\mathcal{Z}}_{\hat{\kappa}(\ell+1)}(X_{\hat{\kappa}(\ell+1)}) - \widehat{\mathcal{Z}}_{\hat{\kappa}(\ell+1)}(\hat{X}_{\hat{\kappa}(\ell+1)})}{2}\hat{H}_\ell^1\Big|^2.$$

Update $\widehat{\Gamma}_\ell$ the solution to this minimization problem `/* Update Hessian */`

for $k = \hat{\kappa} - 1, \dots, 0$ **do**

Minimize over network functions $\mathcal{U} : \mathbb{R}^d \to \mathbb{R}$, and $\mathcal{Z} : \mathbb{R}^d \to \mathbb{R}^d$ the loss function at time t_i, $i = (\ell - 1)\hat{\kappa} + k$:

$$\begin{aligned} & J_i^{MB}(\mathcal{U}, \mathcal{Z}) \\ & = \mathbb{E}\Big|g(X_N) - |\pi| \sum_{j=i+1}^{N-1} F(t_j, X_j, \widehat{\mathcal{U}}_j(X_j), \widehat{\mathcal{Z}}_j(X_j), \widehat{\Gamma}_\ell(X_j)) \\ & \qquad - \sum_{j=i+1}^{N-1} \widehat{\mathcal{Z}}_j(X_j)^\intercal \sigma \Delta W_j - \mathcal{U}(X_i) \\ & \qquad - |\pi| F(t_i, X_i, \mathcal{U}(X_i), \mathcal{Z}(X_i), \widehat{\Gamma}_\ell(X_i)) - \mathcal{Z}(X_i) \cdot \sigma \Delta W_i\Big|^2. \end{aligned}$$

Update $(\widehat{\mathcal{U}}_i, \widehat{\mathcal{Z}}_i)$ as the solution to this minimization problem `/* Update the function and its derivative */`

end

end

Algorithm 5: Second order Multistep Malliavin DBDP (2M^{2}DBDP)

for $\ell = \hat{N}, \dots, 0$ **do**

If $\ell = \hat{N}$, update $\widehat{\Gamma}_\ell = D^2 g$, otherwise minimize over network functions $\Gamma : \mathbb{R}^d \to \mathbb{S}^d$ the loss function

$$\begin{aligned}
&\mathcal{J}_\ell^{2,M}(\Gamma) \\
&= \mathbb{E}\Big|\Gamma(X_{\hat{\kappa}\ell}) - \frac{D^2 g(X_{\hat{\kappa}\hat{N}}) + D^2 g(\hat{X}_{\hat{\kappa}\hat{N}})}{2} \\
&\quad + \frac{|\hat{\pi}|}{2} \sum_{m=\ell+1}^{\hat{N}-1} \Big(F\big(t_{\hat{\kappa}m}, X_{\hat{\kappa}m}, \widehat{\mathcal{U}}_{\hat{\kappa}m}(X_{\hat{\kappa}m}), \widehat{\mathcal{Z}}_{\hat{\kappa}m}(X_{\hat{\kappa}m}), \widehat{\Gamma}_m(X_{\hat{\kappa}m})\big) \\
&\qquad + F\big(t_{\hat{\kappa}m}, \hat{X}_{\hat{\kappa}m}, \widehat{\mathcal{U}}_{\hat{\kappa}m}(\hat{X}_{\hat{\kappa}m}), \widehat{\mathcal{Z}}_{\hat{\kappa}m}(\hat{X}_{\hat{\kappa}m}), \widehat{\Gamma}_m(\hat{X}_{\hat{\kappa}m})\big) \\
&\qquad - 2F\big(t_{\hat{\kappa}\ell}, \hat{X}_{\hat{\kappa}\ell}, \widehat{\mathcal{U}}_{\hat{\kappa}\ell}(\hat{X}_{\hat{\kappa}\ell}), \widehat{\mathcal{Z}}_{\hat{\kappa}\ell}(\hat{X}_{\hat{\kappa}\ell}), \widehat{\Gamma}_\ell(\hat{X}_{\hat{\kappa}\ell})\big) \Big) \hat{H}_{\ell,m}^2 \Big|^2.
\end{aligned}$$

Update $\widehat{\Gamma}_\ell$ the solution to this minimization problem `/* Update Hessian */`

for $k = \hat{\kappa} - 1, \dots, 0$ **do**

Minimize over network functions $\mathcal{U} : \mathbb{R}^d \to \mathbb{R}$, and $\mathcal{Z} : \mathbb{R}^d \to \mathbb{R}^d$ the loss function at time t_i, $i = (\ell - 1)\hat{\kappa} + k$:

$$\begin{aligned}
&J_i^{MB}(\mathcal{U}, \mathcal{Z}) \\
&= \mathbb{E}\Big| g(X_N) - |\pi| \sum_{j=i+1}^{N-1} F(t_j, X_j, \widehat{\mathcal{U}}_j(X_j), \widehat{\mathcal{Z}}_j(X_j), \widehat{\Gamma}_\ell(X_j)) \\
&\qquad - \sum_{j=i+1}^{N-1} \widehat{\mathcal{Z}}_j(X_j)^\intercal \sigma \Delta W_j - \mathcal{U}(X_i) \\
&\qquad - |\pi| F(t_i, X_i, \mathcal{U}(X_i), \mathcal{Z}(X_i), \widehat{\Gamma}_\ell(X_i)) - \mathcal{Z}(X_i) \cdot \sigma \Delta W_i \Big|^2.
\end{aligned}$$

Update $(\widehat{\mathcal{U}}_i, \widehat{\mathcal{Z}}_i)$ as the solution to this minimization problem `/* Update the function and its derivative */`

end

end

4 Numerical applications

We test our different algorithms on various examples and by varying the state space dimension. If not stated otherwise, we choose the maturity $T = 1$. In each example we use an architecture composed of 2 hidden layers with $d + 10$ neurons. We apply Adam gradient descent [KB14] with a decreasing learning rate, using the Tensorflow library [Aba+16]. Each numerical experiment is conducted using a node composed of 2 Intel® Xeon® Gold 5122 Processors, 192 Go of RAM, and 2 GPU nVidia® Tesla® V100 16Go. We use a batch size of 1000.

4.1 Numerical tests on credit valuation adjustment pricing

We consider an example of model from [HL17] for the pricing of CVA in a d-dimensional Black-Scholes model

$$\mathrm{d}X_t = \sigma X_t \, \mathrm{d}W_t, \; X_0 = 1_d$$

with $\sigma > 0$, given by the nonlinear PDE

$$\begin{cases} \partial_t u + \frac{\sigma^2}{2} \mathrm{Tr}(x^\top D_x^2 u \; x) + \beta(u_+ - u) = 0 & \text{on } [0,T] \times \mathbb{R}^d \\ u(T,x) = |\sum_{i=1}^d x_i - d| - 0.1 & \text{on } \mathbb{R}^d \end{cases}$$

with a straddle type payoff. We compare our results with the DBDP scheme [HPW20] with the ones from the Deep BSDE solver [HJE17]. The results in Table 1 are averaged over 10 runs and the standard deviation is written in parentheses. We use ReLu activation functions.

Dimension d	DBDP [HPW20]	DBSDE [HJE17]
1	0.05950 (0.000257)	0.05949 (0.000264)
3	0.17797 (0.000421)	0.17807 (0.000288)
5	0.25956 (0.000467)	0.25984 (0.000331)
10	0.40930 (0.000623)	0.40886 (0.000196)
15	0.52353 (0.000591)	0.52389 (0.000551)
30	0.78239 (0.000832)	0.78231 (0.001266)

Table 1: CVA value with $X_0 = 1, T = 1, \beta = 0.03, \sigma = 0.2$ and 50 time steps.

We observe in Table 1 that both algorithms give very close results and are able to solve the nonlinear pricing problem in high dimension d. The variance of the results is quite small and similar from one to another but increases with the dimension. The same conclusions arise when solving the PDE for the larger maturity $T = 2$.

4.2 Portfolio allocation in stochastic volatility models

We consider several examples from [PWG19] that we solve with Algorithms 3 (2EMDBDP), 4 (2MDBDP), and 5 (2M^{2}DBDP) designed in this paper. Notice that some comparison tests with the 2DBSDE scheme [BEJ19] have been already done in [PWG19]. For a resolution with $N = 120, \hat{N} = 30$, the execution of our multitep algorithms takes between 10000 s. and 30000 s. (depending on the dimension) with a number of gradient descent iterations fixed at 4000 at each time step except 80000 at the first one. We use tanh as activation function.

We consider a portfolio selection problem formulated as follows. There are n risky assets of uncorrelated price process $P = (P^1, \dots, P^n)$ with dynamics governed by

$$\mathrm{d}P_t^i = P_t^i \sigma(V_t^i)\big[\lambda_i(V_t^i)\mathrm{d}t + \mathrm{d}W_t^i\big], \quad i = 1, \dots, n,$$

where $W = (W^1, \dots, W^n)$ is a n-dimensional Brownian motion, $\lambda = (\lambda^1, \dots, \lambda^n)$ is the market price of risk of the assets, σ is a positive function (e.g. $\sigma(v) = e^v$ corresponding to the Scott model), and $V = (V^1, \dots, V^n)$ is the volatility factor modeled by an Ornstein-Uhlenbeck (O.U.) process

$$\mathrm{d}V_t^i = \kappa_i[\theta_i - V_t^i]\mathrm{d}t + \nu_i \mathrm{d}B_t^i, \quad i = 1, \dots, n,$$

with $\kappa_i, \theta_i, \nu_i > 0$, and $B = (B^1, \dots, B^n)$ a n-dimensional Brownian motion, s.t. $d < W^i, B^j >= \delta_{ij}\rho_{ij}dt$, with $\rho_i := \rho_{ii} \in (-1, 1)$. An agent can invest at any time an amount $\alpha_t = (\alpha_t^1, \dots, \alpha_t^n)$ in the stocks, which generates a wealth process $\mathcal{X} = \mathcal{X}^\alpha$ governed by

$$\mathrm{d}\mathcal{X}_t = \sum_{i=1}^n \alpha_t^i \sigma(V_t^i)\big[\lambda_i(V_t^i)\mathrm{d}t + \mathrm{d}W_t^i\big].$$

The objective of the agent is to maximize her expected utility from terminal wealth:

$$\mathbb{E}\big[U(\mathcal{X}_T^\alpha)\big] \quad \leftarrow \quad \text{maximize over } \alpha$$

It is well-known that the solution to this problem can be characterized by the dynamic programming method (see e.g. [Pha09]), which leads to the Hamilton-Jacobi-Bellman for the value function on $[0, T) \times \mathbb{R} \times \mathbb{R}^n$:

$$\begin{cases} \partial_t u + \sum_{i=1}^n \big[\kappa_i(\theta_i - v_i)\partial_{v_i} u + \frac{1}{2}\nu_i^2 \partial_{v_i}^2 u\big] \\ = \frac{1}{2}R(v)\frac{(\partial_{\mathrm{x}} u)^2}{\partial_{\mathrm{xx}}^2 u} + \sum_{i=1}^n \big[\rho_i \lambda_i(v_i)\nu_i \frac{\partial_{\mathrm{x}} u \partial_{\mathrm{x}v_i}^2 u}{\partial_{\mathrm{xx}}^2 u} + \frac{1}{2}\rho_i^2 \nu_i^2 \frac{(\partial_{\mathrm{x}v_i}^2 u)^2}{\partial_{\mathrm{xx}}^2 u}\big] \\ u(T, \mathrm{x}, v) = U(\mathrm{x}), \qquad \mathrm{x} \in \mathbb{R},\ v \in \mathbb{R}^n, \end{cases}$$

with a Sharpe ratio $R(v) := |\lambda(v)|^2$, for $v = (v_1, \dots, v_n) \in (0, \infty)^n$. The optimal portfolio strategy is then given in feedback form by $\alpha_t^* = \hat{a}(t, \mathcal{X}_t^*, V_t)$, where $\hat{a} = (\hat{a}_1, \dots, \hat{a}_n)$ is given by

$$\begin{aligned} &\hat{a}_i(t, \mathrm{x}, v) \\ &= -\frac{1}{\sigma(v_i)}\Big(\lambda_i(v_i)\frac{\partial_{\mathrm{x}} u}{\partial_{\mathrm{xx}}^2 u} + \rho_i \nu_i \frac{\partial_{\mathrm{x}v_i}^2 u}{\partial_{\mathrm{xx}}^2 u}\Big),\ (t, \mathrm{x}, v = (v_1, \dots, v_n)) \in [0, T) \times \mathbb{R} \times \mathbb{R}^n, \end{aligned}$$

for $i = 1, \dots, n$.

We shall test this example when the utility function U is of exponential form: $U(\mathrm{x}) = -\exp(-\eta \mathrm{x})$, with $\eta > 0$, and under different cases for which explicit solutions are available. We refer to [PWG19] where these solutions are described.

(1) *Merton problem.* This corresponds to a degenerate case where the factor V, hence the volatility σ and the risk premium λ are constant ($v_i = \theta_i,\ \nu_i = 0$). We train our algorithms with the forward process

$$X_{k+1} = X_k + |\lambda|\Delta t_k + \Delta W_k, \quad k = 0, \dots, N,\ X_0 = x_0.$$

(2) *One risky asset:* $n = 1$. We train our algorithms with the forward process

$$\begin{aligned} \mathcal{X}_{k+1} &= \mathcal{X}_k + \lambda(\theta)\Delta t_k + \Delta W_k, \quad k = 0, \dots, N-1, \ \mathcal{X}_0 = \mathrm{x}_0 \\ V_{k+1} &= V_k + \nu \Delta B_k, \qquad k = 0, \dots, N-1, \ V_0 = \theta. \end{aligned}$$

We test our algorithm with $\lambda(v) = \lambda v$, $\lambda > 0$, for which we have an explicit solution.

(3) *No leverage effect, i.e.,* $\rho_i = 0$, $i = 1, \dots, n$. We train with the forward process

$$\begin{aligned} \mathcal{X}_{k+1} &= \mathcal{X}_k + \sum_{i=1}^{n} \lambda_i(\theta_i)\Delta t_k + \Delta W_k, \quad k = 0, \dots, N-1, \ \mathcal{X}_0 = \mathrm{x}_0 \\ V^i_{k+1} &= V^i_k + \nu_i \Delta B^i_k, \qquad k = 0, \dots, N-1, \ V^i_0 = \theta_i. \end{aligned}$$

We test our algorithm with $\lambda_i(v) = \lambda_i v_i$, $\lambda_i > 0$, $i = 1, \dots, n$, $v = (v_1, \dots, v_n)$, for which we have an explicit solution.

Merton Problem. We take $\eta = 0.5$, $\lambda = 0.6$, $N = 120$, $\hat{N} = 30$, $T = 1$, $x_0 = 1$. We plot in Figure 1 the neural networks approximation of $u, D_x u, D_x^2 u$, and the feedback control $\hat{a}$ (for one asset) computed from our different algorithms, together with their analytic values (in orange). As also reported in the estimates of Table 2, the multistep algorithms improve significantly the results obtained in [PWG19], where the estimation of the Hessian is not really accurate (see blue curve in Figure 1).

	Average	Standard deviation	Relative error (%)
[PWG19]	-0.50561	0.00029	0.20
2EMDBDP	**-0.50673**	**0.00019**	**0.022**
2MDBDP	-0.50647	0.00033	0.030
$2M^2$DBDP	-0.50644	0.00022	0.035

Table 2: Estimate of $u(0, 1.)$ in the Merton problem with $N = 120$, $\hat{N} = 30$. Average and standard deviation observed over 10 independent runs are reported. The theoretical solution is -0.50662.

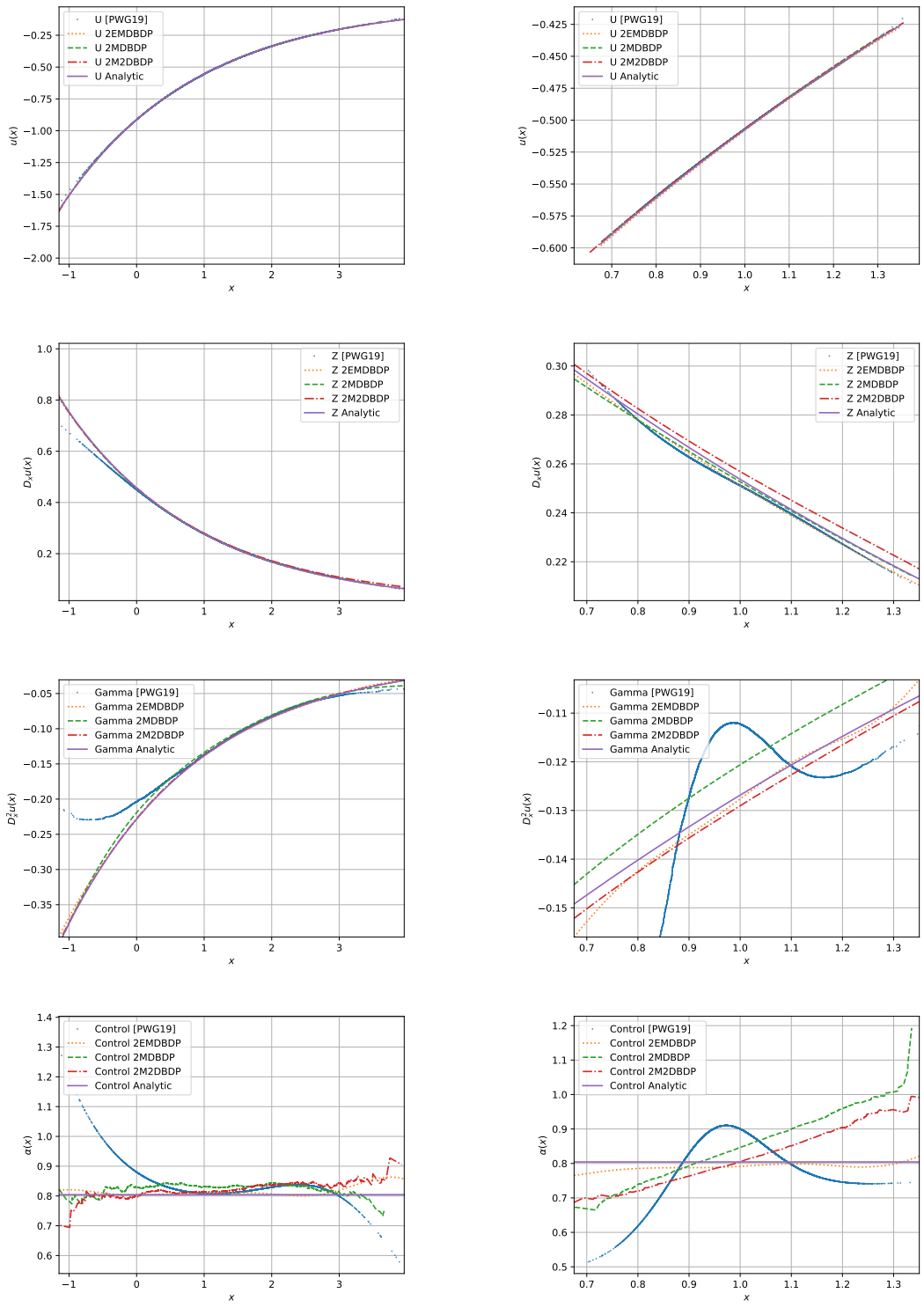

Figure 1: Estimates of u, $D_x u$, $D_x^2 u$ and of the optimal control α on the Merton problem with $N = 120$, $\hat{N} = 30$. We take $x_0 = 1.$, at the left $t = 0.5042$, and at the right $t = 0.0084$.

One asset $n = 1$ in Scott volatility model. We take $\eta = 0.5$, $\lambda = 1.5$, $\theta = 0.4$, $\nu = 0.4$, $\kappa = 1$, $\rho = -0.7$, $T = 1$, $x_0 = 1$. For all tests we choose $N = 120$, $\hat{N} = 30$

and $\sigma(v) = e^v$. We report in Table 3 the relative error between the neural networks approximation of $u, D_{\mathrm{x}}u, D^2_{\mathrm{x}}u$ computed from our different algorithms and their analytic values. It turns out that the multistep extension of [PWG19], namely 2EMDBDP scheme, yields a very accurate approximation result, much better than the other algorithms, with also a reduction of the standard deviation.

	Average	Standard deviation	Relative error (%)
[PWG19]	-0.53431	0.00070	0.34
2EMDBDP	**-0.53613**	**0.00045**	**0.007**
2MDBDP	-0.53772	0.00046	0.304
2M^{2}DBDP	-0.53205	0.00050	0.755

Table 3: Estimate of $u(0, 1, \theta)$ on the One Asset problem with stochastic volatility ($d = 2$) and $N = 120$, $\hat{N} = 30$. Average and standard deviation observed over 10 independent runs are reported. The exact solution is -0.53609477.

No Leverage in Scott model. In the case with one asset we take $\eta = 0.5$, $\lambda = 1.5$, $\theta = 0.4$, $\nu = 0.2$, $\kappa = 1$, $T = 1$, $x_0 = 1$. For all tests we choose $N = 120$, $\hat{N} = 30$ and $\sigma(v) = e^v$. We report in Table 4 the relative error between the neural networks approximation of $u, D_{\mathrm{x}}u, D^2_{\mathrm{x}}u$ computed from our different algorithms and their analytic values. All the algorithms yield quite accurate results, but compared to the case with correlation in Table 3, it appears here that the best performance in terms of precision is achieved by Algorithm 2M^{2}DBDP.

	Average	Standard deviation	Relative error (%)
[PWG19]	-0.49980	0.00073	0.35
2EMDBDP	-0.50400	0.00229	0.485
2MDBDP	-0.50149	**0.00024**	0.015
2M^{2}DBDP	**-0.50157**	0.00036	**0.001**

Table 4: Estimate of $u(0, 1, \theta)$, with 120 time steps on the No Leverage problem with 1 asset ($d = 2$) and $N = 120$, $\hat{N} = 30$. Average and standard deviation observed over 10 independent runs are reported. The exact solution is -0.501566.

In the case with four assets ($n = 4$, $d = 5$), we take $\eta = 0.5$,
$\lambda = \begin{pmatrix}1.5 & 1.1 & 2. & 0.8\end{pmatrix}$, $\theta = \begin{pmatrix}0.1 & 0.2 & 0.3 & 0.4\end{pmatrix}$, $\nu = \begin{pmatrix}0.2 & 0.15 & 0.25 & 0.31\end{pmatrix}$, $\kappa = \begin{pmatrix}1. & 0.8 & 1.1 & 1.3\end{pmatrix}$. The results are reported in Table 5. We observe that the algorithm in [PWG19] provides a not so accurate outcome, while its multistep version (2EMDBDP scheme) divides by 10 the relative error and the standard deviation.

	Average	Standard deviation	Relative error (%)
[PWG19]	-0.43768	0.00137	0.92
2EMDBDP	**-0.4401**	**0.00051**	**0.239**
2MDBDP	-0.43796	0.00098	0.861
2M^{2}DBDP	-0.44831	0.00566	1.481

Table 5: Estimate of $u(0,1,\theta)$, with 120 time steps on the No Leverage problem with 4 assets ($d = 5$) and $N = 120$, $\hat{N} = 30$. Average and standard deviation observed over 10 independent runs are reported. The theoretical solution is -0.44176462.

In the case with nine assets ($n = 9$, $d = 10$), we take $\eta = 0.5$,
$\lambda = \big(1.5 \quad 1.1 \quad 2. \quad 0.8 \quad 0.5 \quad 1.7 \quad 0.9 \quad 1. \quad 0.9\big)$,
$\theta = \big(0.1 \quad 0.2 \quad 0.3 \quad 0.4 \quad 0.25 \quad 0.15 \quad 0.18 \quad 0.08 \quad 0.91\big)$,
$\nu = \big(0.2 \quad 0.15 \quad 0.25 \quad 0.31 \quad 0.4 \quad 0.35 \quad 0.22 \quad 0.4 \quad 0.15\big)$,
$\kappa = \big(1. \quad 0.8 \quad 1.1 \quad 1.3 \quad 0.95 \quad 0.99 \quad 1.02 \quad 1.06 \quad 1.6\big)$. The results are reported in Table 6. The approximation is less accurate than in lower dimension, but we observe again that compared to one-step scheme in [PWG19] , the multistep versions improve significantly the standard deviation of the result. However the best performance in precision is obtained here by the [PWG19] scheme.

	$\hat{N}$	Average	S.d.	Relative error (%)
[PWG19]		**-0.27920**	0.05734	**1.49**
2EMDBDP		-0.26631	**0.00283**	3.19
2MDBDP	30	-0.28979	0.00559	5.34
2MDBDP	60	-0.28549	0.00948	3.78
2MDBDP	120	**-0.28300**	0.01129	**2.87**
2M^{2}DBDP	30	NC	NC	NC

Table 6: Estimate of $u(0,1,\theta)$, with 120 time steps on the No Leverage problem with 9 assets ($d = 10$) and $N = 120$. Average and standard deviation (S.d.) observed over 10 independent runs are reported. The theoretical solution is -0.27509173.

5 Extensions and perspectives

• Solving mean-field control and mean-field games through McKean-Vlasov FBSDEs.

These methods solve the optimality conditions for mean-field problems through the stochastic Pontryagin principle from [CD18]. The law of the solution influences the coupled FBSDEs dynamics so they are of McKean-Vlasov type. Variations around the Deep BSDE method [HJE17] are used to solve such a system by [CL19], [FZ20]. [GMW19] uses the Merged method from [CWNMW19] and solves several numerical examples in dimension 10 by introducing an efficient law estimation technique. [CL19] also proposes another method dedicated to mean field control to directly tackle the optimization problem with a neural network as the control in the stochastic dynamics.

• Solving mean-field control through master Bellman equation and symmetric neural networks.

[Ger+20] solves the master Bellman equation arising from dynamic programming principle applied to mean-field control problems (see [PW17]). The paper approximates the value function evaluated on the empirical measure stemming from particles simulation of a training forward process. It provides a rate for the particle method convergence. The symmetry between iid particles is enforced by optimizing over exchangeable high-dimensional neural networks, invariant by permutation of their inputs.

• **Reinforcement Learning for mean-field control and mean-field games [CLT19; AKS19; AFL20; Gu+20; Guo+20].**

Some works focus on similar problems but with unknown dynamics. Thus they rely on trajectories sampled from a simulator and reinforcement learnings– especially Q-learning– to estimate the state action value function and optimal control without a model. The idea is to optimize a neural network by relying on a memory of past state action transitions used to train the network in order for it to verify the Bellman equation on samples from memory replay.

• **Machine learning framework for solving high-dimensional mean field game and mean field control problems [Rut+20]**

This paper focuses on potential mean field games, in which the cost functions depending on the law can be written as the linear functional derivative of a function with respect to a measure. A Lagrangian method with Deep Galerkin type penalization is used. In this case the potential is approached by a neural network and solving mean-field games amounts to solve an unconstrained optimization problem.

• **Deep quantum neural networks** [Sak20]

We briefly mention this work studying the use of deep quantum neural networks which exploit the quantum superposition properties by replacing bits by "qubits". Promising results are obtained when using these networks for regression in financial contexts such as implied volatility estimation. Future works may study the application of such neural networks to control problems and PDEs.

• **Path signature for path-dependent PDE** [SVSS20]

This work extends previously developed methods for solving state-dependent PDEs to the linear path-dependent setting coming for instance from the pricing and hedging of path-dependent options. A path-dependent Feynman-Kac representation is numerically computed through a global minimization over neural networks. The authors show that using LSTM networks taking the forward process' path signatures (coming from the rough paths literature) as input yields better results than taking the discretized path as input of a feedforward network.

References

[AA+18] A. Al-Aradi et al. "Solving Nonlinear and High-Dimensional Partial Differential Equations via Deep Learning". In: *arXiv:1811.08782* (2018).

[Aba+16] M. Abadi et al. "TensorFlow: A system for large-scale machine learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283.

[AFL20] A. Angiuli, J.-P. Fouque, and M. Laurière. "Unified Reinforcement Q-Learning for Mean Field Game and Control Problems". In: *arXiv:2006.13912* (2020).

[AKS19] B. Anahtarcı, C. Deha Karıksız, and N. Saldi. "Fitted Q-Learning in Mean-field Games". In: *arXiv:1912.13309* (2019).

[Bac+19] A. Bachouch et al. "Deep neural networks algorithms for stochastic control problems on finite horizon: numerical computations". In: *Methodol. Comput. Appl. Probab, to appear* (2019).

[BCJ19] S. Becker, P. Cheridito, and A. Jentzen. "Deep optimal stopping". In: *J. Mach. Learn. Res.* 20 (2019), pp. 1–25.

[BD07] C. Bender and R. Denk. "A forward scheme for backward SDEs". In: *Stochastic Process. Appl.* 117.12 (2007), pp. 1793 –1812.

[Bec+19a] C. Beck et al. "Deep splitting method for parabolic PDEs". In: *arXiv:1907.03452* (July 2019).

[Bec+19b] S. Becker et al. "Solving high-dimensional optimal stopping problems using deep learning". In: *arXiv:1908.01602* (2019).

[Bec+20] C. Beck et al. "An overview on deep learning-based approximation methods for partial differential equations". In: *arXiv:2012.12348* (2020).

[BEJ19] C. Beck, W. E, and A. Jentzen. "Machine Learning Approximation Algorithms for High-Dimensional Fully Nonlinear Partial Differential Equations and Second-order Backward Stochastic Differential Equations". In: *J. Nonlinear Sci.* 29.4 (2019), pp. 1563–1619.

[BT04] B. Bouchard and N. Touzi. "Discrete-time approximation and Monte-Carlo simulation of backward stochastic differential equations". In: *Stochastic Process. Appl.* 111.2 (2004), pp. 175 –206.

[CD18] R. Carmona and F. Delarue. *Probabilistic Theory of Mean Field Games with Applications vol I. and II.* Vol. 83. Probability Theory and Stochastic Modelling. Springer, 2018.

[Che+07] P. Cheridito et al. "Second-order backward stochastic differential equations and fully nonlinear parabolic PDEs". In: *Comm. Pure Appl. Math.* 60.7 (2007), pp. 1081–1110.

[CL19] R. Carmona and M. Laurière. "Convergence analysis of machine learning algorithms for the numerical solution of mean-field control and games: II The finite horizon case". In: *arXiv:1908.01613* (2019).

[CLT19] R. Carmona, M. Laurière, and Z. Tan. "Model-Free Mean-Field Reinforcement Learning: Mean-Field MDP and Mean-Field Q-Learning". In: *arXiv:1910.12802* (2019).

[CWNMW19] Q. Chan-Wai-Nam, J. Mikael, and X. Warin. "Machine Learning for Semi Linear PDEs". In: *J. Sci. Comput.* 79 (2019), pp. 1667–1712.

[DPT94] M.W.M. Dissanayake and N. Phan-Thien. "Neural network-based approximations for solving partial differential equations". In: *Commun. Numer. Methods Eng.* 10.3 (1994), pp. 195–201.

[EHJ17] W. E, J. Han, and A. Jentzen. "Deep Learning-Based numerical methods for high dimensional parabolic partial differential equations and backward stochastic differential equations". In: *Commun. Math. Stat.* 5.4 (2017), pp. 349–380.

[EY18] W. E and B. Yu. "The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems". In: *Commun. Math. Stat.* 6 (2018), pp. 1–12.

[FTT19] M. Fujii, A. Takahashi, and M. Takahashi. "Asymptotic expansion as prior knowledge in deep learning method for high dimensional BSDEs". In: *Asia Pacific Financial Markets* 26.3 (2019), pp. 391–408.

[FTW11] A. Fahim, N. Touzi, and X. Warin. "A probabilistic numerical method for fully nonlinear parabolic PDEs". In: *Ann. Appl. Probab.* 21.4 (Aug. 2011), pp. 1322–1364.

[FZ20] J-.P. Fouque and Z. Zhang. "Deep Learning Methods for Mean Field Control Problems with Delay". In: *Frontiers in Applied Mathematics and Statistics* 6 (2020).

[Ger+20] M. Germain et al. "Solving mean-field PDEs with symmetric neural networks". In: *in preparation* (2020).

[GLW05] E. Gobet, J-P. Lemor, and X. Warin. "A regression-based Monte Carlo method to solve backward stochastic differential equations". In: *Ann. Appl. Probab.* 15.3 (2005), pp. 2172–2202.

[GM05] E. Gobet and R. Munos. "Sensitivity analysis using Itô-Malliavin calculus and martingales, and application to stochastic optimal control". In: *SIAM J. Control Optim.* 43.5 (2005), pp. 1676–1713.

[GMW19] M. Germain, J. Mikael, and X. Warin. "Numerical resolution of McKean-Vlasov FBSDEs using neural networks". In: *arXiv:1909.12678* (2019).

[GPR20] A. Gnoatto, A. Picarelli, and C. Reisinger. "Deep XVA solver-a neural network based counterparty credit risk management framework". In: *arXiv: 2005.02633* (2020).

[GPW20] M. Germain, H. Pham, and X. Warin. "Deep backward multistep schemes for nonlinear PDEs and approximation error analysis". In: *arXiv:2006.01496v1* (2020).

[Gro+18] P. Grohs et al. "A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equation". In: *to appear in Memoirs of the American mathematical society* (2018).

[GT14] E. Gobet and P. Turkedjiev. "Linear regression MDP scheme for discrete backward stochastic differential equations under general conditions". In: *Math. Comp.* 85 (Mar. 2014).

[Gu+20] H. Gu et al. "Q-Learning Algorithm for Mean-Field Controls, with Convergence and Complexity Analysis". In: *arXiv:2002.04131* (2020).

[Guo+20] X. Guo et al. "A General Framework for Learning Mean-Field Games". In: *arXiv:2003.06069* (2020).

[GW20] K. Glau and L. Wunderlich. "The deep parametric PDE method: application to option pricing". In: *arXiv:2012.06211* (2020).

[HE16] J. Han and W. E. "Deep learning approximation for stochastic control problems". In: *Deep Reinforcement Learning Workshop* (2016).

[HH21] J. Han and R. Hu. "Recurrent Neural Networks for Stochastic Control Problems with Delay". In: *arXiv:2101.01385* (2021).

[HJE17] J. Han, A. Jentzen, and W. E. "Solving high-dimensional partial differential equations using deep learning". In: *Proc. Natl. Acad. Sci. USA* 115 (2017).

[HL17] P. Henry-Labordere. "Deep Primal-Dual Algorithm for BSDEs: Applications of Machine Learning to CVA and IM". In: *SSRN: 3071506* (2017).

[HL20] J. Han and J. Long. "Convergence of the Deep BSDE Method for Coupled FBSDEs". In: *Probab. Uncertain. Quant. Risk* 5.1 (2020), pp. 1–33.

[HPW20] C. Huré, H. Pham, and X. Warin. "Deep backward schemes for high-dimensional nonlinear PDEs". In: *Math. Comp.* 89.324 (2020), pp. 1547–1580.

[Hur+18] C. Huré et al. "Deep neural networks algorithms for stochastic control problems on finite horizon: convergence analysis". In: *arXiv:1812.04300, to appear in SIAM J. Numer. Anal.* (2018).

[Hut+20] M. Hutzenthaler et al. "A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equation". In: *SN partial differential equations and applications* 1.10 (2020), pp. 1–34.

[Ji+20] S. Ji et al. "Three algorithms for solving high-dimensional fully coupled FBSDEs through deep learning". In: *Intelligent systemsn, IEEE* 35.3 (2020), pp. 71–84.

[JO19] A. Jacquier and M. Oumgari. "Deep curve-dependent PDEs for affine rough volatility". In: *arXiv:1906.02551* (2019).

[KB14] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014.

[KJY20] Y. Khoo, J.Lu, and L. Ying. "Solving parametric PDE problems with artificial neural networks". In: *European Journal of Applied Mathematics* (2020), pp. 1–15.

[KLW20] I. Kharroubi, T. Lim, and X. Warin. "Discretization and Machine Learning Approximation of BSDEs with a Constraint on the Gains-Process". In: *arXiv preprint arXiv:2002.02675* (2020).

[KSS20] S. Kremsner, A. Steinicke, and M. Szölgyenyi. "A deep neural network algorithm for semilinear elliptic PDEs with applications in insurance mathematics". In: *arXiv:2010.15757* (2020).

[LXL19] J. Liang, Z. Xu, and P. Li. "Deep Learning-Based Least Square Forward-Backward Stochastic Differential Equation Solver for High-Dimensional Derivative Pricing". In: *arXiv:1907.10578* (2019).

[NR19] N. Nüskens and L. Richter. "Solving high-dimensional Hamilton-Jacobi-Bellman PDEs using neural networks: perspective from the theory of controlled diffusions and measures on path space". In: *arXiv:2005.05409* (2019).

[Pha09] H. Pham. *Continuous-time Stochastic Control and Optimization with Financial Applications*. Vol. 61. SMAP. Springer, 2009.

[PP90] E. Pardoux and S. Peng. "Adapted solution of a backward stochastic differential equation". In: *Systems & Control Letters* 14.1 (1990), pp. 55 –61. ISSN: 0167-6911.

[PW17] H. Pham and X. Wei. "Dynamic programming for optimal control of stochastic McKean-Vlasov dynamics". In: *SIAM J. Control Optim.* 55.2 (2017), pp. 1069–1101.

[PWG19] H. Pham, X. Warin, and M. Germain. "Neural networks-based backward scheme for fully nonlinear PDEs". In: *arXiv:1908.00412, SN Partial Differential Equations and Applications, to appear* (2019).

[Rai18] M. Raissi. "Forward-Backward Stochastic Neural Networks: Deep Learning of High-dimensional Partial Differential Equations". In: *arXiv:1804.07010* (2018).

[RPK19] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *J. Comput. Phys.* 378 (2019), pp. 686 –707.

[Rut+20] L. Ruthotto et al. "A machine learning framework for solving high-dimensional mean field game and mean field control problems". In: *Proc. Natl. Acad. Sci. USA* 117.17 (2020), pp. 9183–9193.

[Sak20] T. Sakuma. "Application of deep quantum neural networks to finance". In: *arXiv:2011.07319v1* (2020).

[SS17] J. Sirignano and K. Spiliopoulos. "DGM: A deep learning algorithm for solving partial differential equations". In: *J. Comput. Phys.* 375 (Aug. 2017).

[SVSS18] M. Sabate Vidales, D. Siska, and L. Szpruch. "Unbiased deep solvers for parametric PDEs". In: *arXiv:1810.05094v2* (2018).

[SVSS20] M. Sabate Vidales, D. Siska, and L. Szpruch. "Solving path dependent PDEs with LSTM networks and path signatures". In: *arXiv:2011.10630v1* (2020).

[SZ20] Y. Saporito and Z. Zhang. "PDGM: a neural network approach to solve path-dependent partial differential equations". In: *arXiv:2003.02035* (2020).

[Zha04] J. Zhang. "A numerical scheme for BSDEs". In: *Ann. Appl. Probab.* 14.1 (2004), pp. 459–488.