Algorithms for Solving High Dimensional PDEs: From Nonlinear Monte Carlo to Machine Learning

Weinan E[1,2], Jiequn Han[1], and Arnulf Jentzen[3]

[1]Department of Mathematics, Princeton University
[2]Program in Applied and Computational Mathematics, Princeton University
[3]Faculty of Mathematics and Computer Science, University of Münster

September 14, 2020

Abstract

In recent years, tremendous progress has been made on numerical algorithms for solving partial differential equations (PDEs) in a very high dimension, using ideas from either nonlinear (multilevel) Monte Carlo or deep learning. They are potentially free of the curse of dimensionality for many different applications and have been proven to be so in the case of some nonlinear Monte Carlo methods for nonlinear parabolic PDEs.

In this paper, we review these numerical and theoretical advances. In addition to algorithms based on stochastic reformulations of the original problem, such as the multilevel Picard iteration and the Deep BSDE method, we also discuss algorithms based on the more traditional Ritz, Galerkin, and least square formulations. We hope to demonstrate to the reader that studying PDEs as well as control and variational problems in very high dimensions might very well be among the most promising new directions in mathematics and scientific computing in the near future.

Contents

1

1 Introduction

The mathematical models for many problems around us are in the form of partial differential equations (PDEs) in high dimensions. Notable examples include:

- The Hamilton-Jacobi-Bellman (HJB) equation in control theory

$$\frac{\partial u}{\partial t} + H(x, \nabla_x u) = 0. \tag{1}$$

  Here the dimensionality is the dimensionality of the state space of the original control problem. If the original control problem is described by a PDE, then the corresponding HJB equation is formulated in infinite dimensional space.

- The Black-Scholes equations for pricing financial derivatives

$$\frac{\partial u}{\partial t} + \frac{1}{2}\sigma^2 \sum_{i=1}^{d} x_i^2 \frac{\partial^2 u}{\partial x_i^2} + r\left\langle \nabla_x u, x \right\rangle - ru + f(u) = 0. \tag{2}$$

  Here the dimensionality is the number of underlying financial assets. Nonlinear terms $f(u)$ may result when default risks, transaction costs, or other factors are taken into account.

- Many electron Schrödinger equation in quantum mechanics

$$i\frac{\partial u}{\partial t} = \Delta_x u + V(x)u. \tag{3}$$

  Here the dimensionality is roughly three times the number of electrons in the considered quantum mechanical system.

Solving these PDEs has been a notoriously difficult problem in scientific computing and computational science, due to the well-known **curse of dimensionality (CoD)**: The computational complexity grows exponentially as a function of the dimensionality of the problem [13]. In fact, for this reason, traditional numerical algorithms such as finite

difference and finite element methods have been limited to dealing with problems in a rather low dimension. The use of sparse grids can extend the applicability to, say, around 10 dimensions. But beyond that, there seemed to be little hope except for special PDEs.

We are interested in PDEs and algorithms that do not suffer from CoD, i.e., algorithms whose computational complexity scales algebraically with the problem dimension. More precisely, to reach some error tolerance $\varepsilon > 0$, the computational cost should be no more than

$$c(d, \varepsilon) \sim C d^\alpha \varepsilon^{-\beta} \tag{4}$$

where $C, \alpha, \beta \geq 0$ are absolute, dimension-independent constants. In particular, they do not depend on the dimension $d \in \mathbb{N} = \{1, 2, 3, \dots\}$. We are also interested in PDEs for which such algorithms do exist. In fact, we are interested in developing a theory of high dimensional PDEs based on the complexity with which the solutions can be approximated using particular schemes, such as neural network models. We believe that such a theory should be part of the foundation for a theoretical understanding of high dimensional control theory, reinforcement learning, and a host of other problems that will become important research topics in mathematics.

The golden standard for high dimensional problems is the approximation of high dimensional integrals. Let $g \colon [0, 1]^d \to \mathbb{R}$ be a Lebesgue square integrable function defined on the set $X = [0, 1]^d$ and let

$$I(g) = \int_X g(x)\, dx. \tag{5}$$

Typical grid-based quadrature rules, such as the Trapezoidal rule and the Simpson's rule, all suffer from CoD. The one algorithm that does not suffer from CoD is the Monte Carlo algorithm which works as follows. Let x_j, $j \in \{1, 2, \dots, n\}$, be independent, continuous uniformly distributed random samples on X and let

$$\mathcal{I}_n(g) = \frac{1}{n} \sum_{j=1}^n g(x_j). \tag{6}$$

Then a simple calculation gives us

$$\mathbb{E}\big[|I(g) - \mathcal{I}_n(g)|^2\big] = \frac{\mathrm{Var}(g)}{n} \qquad \text{and} \qquad \mathrm{Var}(g) = \int_X |g(x)|^2\, dx - \left[\int_X g(x)\, dx\right]^2. \tag{7}$$

The $O(1/\sqrt{n})$ rate is independent of $d \in \mathbb{N}$. To reduce the error to a tolerance of $\varepsilon > 0$, the number of samples needed must be of order ε^{-2}. This is a situation with $\beta = 2$ in (4).

The value of α is more subtle. We need to examine specific classes of examples in different dimensions. For example, one can ask about the value of α for the Ising or Heisenberg model in statistical physics. At the moment, results in this direction are still quite sparse.

Algorithms and results similar to those of the approximative computation of integrals in (5)–(7) above have been developed in the case of linear PDEs of the Kolmogorov type but, for a very long time, little progress was made in developing algorithms with quantitatively similar computational complexity for high dimensional nonlinear PDEs and this has impeded advances in several fields such as optimal control and quantum mechanics.

Things have changed dramatically in the last few years with the appearance of so-called *full history recursive multilevel Picard approximation* methods [37, 38, 89] and a host of machine learning-based algorithms for high dimensional PDEs beginning with the *Deep BSDE method* [36, 70]. Full history recursive multilevel Picard approximation methods (in the following we abbreviate *full history recursive multilevel Picard* by MLP) are some recursive nonlinear variants of the classical Monte-Carlo approximation methods and, in that sense, MLP approximation methods are nonlinear Monte-Carlo approximation methods. For every arbitrarily small $\delta \in (0, \infty)$ it has been shown that MLP approximation methods achieve (4) with $\alpha = 1$ and $\beta = 2 + \delta$ for a wide class of nonlinear (parabolic) equations (see Section 6 below for details). Although a complete theory is still lacking, the Deep BSDE method has demonstrated very robust performance in practice for a range of problems and has been extended in many different ways. These developments will be reviewed below.

Along with the work on developing algorithms, there has been some effort to develop an open-source platform where codes, review papers, and other information can be shared. The interested reader should consult: http://deeppde.org.

Before launching into a more detailed discussion, it is useful to review briefly the two main ideas that we focus on in this paper: machine learning approximation methods (see Section 1.1 below) and MLP approximation methods (see Section 1.2 below).

1.1 A brief introduction of supervised learning

The basic problem in supervised learning is as follows: Given a natural number $n \in \mathbb{N}$ and a sequence $(x_j, y_j) = (x_j, f^*(x_j))$, $j \in \{1, 2, \ldots, n\}$, of pairs of input-output data, we want to recover the target function f^* as accurately as possible. We will assume that the input data x_j, $j \in \{1, 2, \ldots, n\}$, is sampled from the probability distribution μ on $\mathbb{R}^d$.

Step 1. Choose a hypothesis space. This is a set of trial functions $\mathcal{H}_m$ where $m \in \mathbb{N}$ is a natural number that is strongly related to the dimensionality of $\mathcal{H}_m$. One might choose piecewise polynomials or wavelets. In modern machine learning the most popular choice is neural network functions. Two-layer neural network functions (one input layer, one output layer that usually does not count, and one hidden layer) take the form

$$f_m(x, \theta) = \frac{1}{m} \sum_{j=1}^{m} a_j \sigma(\langle w_j, x \rangle) \tag{8}$$

where $\sigma \colon \mathbb{R} \to \mathbb{R}$ is a fixed scalar nonlinear function and where $\theta = (a_j, w_j)_{j \in \{1, 2, \ldots, m\}}$ are the parameters to be optimized (or trained). A popular example for the nonlinear function $\sigma \colon \mathbb{R} \to \mathbb{R}$ is the rectifier function (sometimes also referred to as ReLU (rectified linear unit) activation function in which case we have for all $z \in \mathbb{R}$ that $\sigma(z) = \max\{z, 0\}$. Roughly speaking, deep neural network (DNN) functions are obtained if one composes two-layer neural network functions several times. One important class of DNN models are residual neural networks (ResNet). They closely resemble discretized ordinary differential

4

equations and take the form

$$z_{l+1} = z_l + \frac{1}{LM} \sum_{j=1}^{M} a_{j,l} \sigma(\langle z_l, w_{j,l} \rangle), \qquad z_0 = Vx, \qquad f_L(x, \theta) = \langle \alpha, z_L \rangle \qquad (9)$$

for $l \in \{0, 1, \ldots, L-1\}$ where $L, M \in \mathbb{N}$. Here the parameters are $\theta = (\alpha, V, (a_{j,l})_{j,l}, (w_{j,l})_{j,l})$. ResNets are the model of choice for truly deep neural network models.

Step 2. Choose a loss function. The primary consideration for the choice of the loss function is to fit the data. Therefore one most obvious choice is the L^2 loss:

$$\hat{\mathcal{R}}_n(f) = \frac{1}{n} \sum_{j=1}^{n} |f(x_j) - y_j|^2 = \frac{1}{n} \sum_{j=1}^{n} |f(x_j) - f^*(x_j)|^2. \qquad (10)$$

This is also called the "empirical risk". Sometimes we also add regularization terms.

Step 3. Choose an optimization algorithm. The most popular optimization algorithms in machine learning are different versions of the gradient descent (GD) algorithm, or its stochastic analog, the stochastic gradient descent (SGD) algorithm. Assume that the objective function we aim to minimize is of the form

$$F(\theta) = \mathbb{E}_{\xi \sim \nu} \big[l(\theta, \xi) \big] \qquad (11)$$

(ν could be an empirical distribution on a finite set). The simplest form of SGD iteration takes the form

$$\theta_{k+1} = \theta_k - \eta \nabla l(\theta_k, \xi_k), \qquad (12)$$

for $k \in \mathbb{N}_0$ where ξ_k, $k \in \mathbb{N}_0 = \{0, 1, 2, \ldots\}$, is a sequence of i.i.d. random variables sampled from the distribution ν and η is the learning rate which might also change during the course of the iteration. In contrast GD takes the form

$$\theta_{k+1} = \theta_k - \eta \nabla \mathbb{E}_{\xi \sim \nu} \big[l(\theta_k, \xi) \big]. \qquad (13)$$

Obviously this form of SGD can be adapted to loss functions of the form (10) which can also be regarded as an expectation. The DNN-SGD paradigm is the heart of modern machine learning.

High dimensional stochastic control problems. One of the earliest applications of deep learning to problems in scientific computing is for the stochastic control problem [67]. This example was chosen because of its close resemblance to the DNN-SGD paradigm in deep learning. From an abstract viewpoint, DNN can be viewed as a (discrete) dynamical system, of which ResNet is a good example. SGD is a natural consequence when applying GD to stochastic optimization problems, in which the objective function is an expectation.

Consider the stochastic dynamical system:

$$s_{t+1} = s_t + b_t(s_t, a_t) + \xi_{t+1}. \qquad (14)$$

Here s_t, a_t, ξ_t are respectively the state, the control, and the noise at time t. We assume that the objective function for the control problem is of the form:

$$\min_{\{a_t\}_{t=0}^{T-1}} \mathbb{E}\Big[\sum_{t=0}^{T-1} c_t(s_t, a_t(s_t)) + c_T(s_T) \mid s_0 \Big], \tag{15}$$

where T is the time horizon and $a_t = a_t(s_t)$ are the feedback controls. One can see the close analogy between the stochastic control problem and the DNN-SGD paradigm: (14) plays the role of (9) and (15) is in the form of a stochastic optimization problem. In this analogy, the role of the training data is played by the noise $\{\xi_t\}$.

To develop an algorithm for this problem, one can approximate the feedback control function a_t by any machine learning model, in particular some neural network model:

$$a_t(s_t) \approx a_t(s_t|\theta_t), \tag{16}$$

where θ_t is the parameter to be trained at time t. The loss function can be defined by

$$L(\{\theta_t\}) = \mathbb{E}\left[\sum_{t=0}^{T-1} c_t(s_t, a_t(s_t|\theta_t)) + c_T(s_T) \right], \tag{17}$$

which can be optimized using SGD over different random samples of ξ_t, $t \in \{1, 2, \ldots, T\}$. An example of energy storage is shown in Figure 1. It is an allocation problem, with the objective being optimizing total revenue from multiple storage devices and a renewable wind energy source while satisfying stochastic demand. More details of the problem can be found in [67, 95].
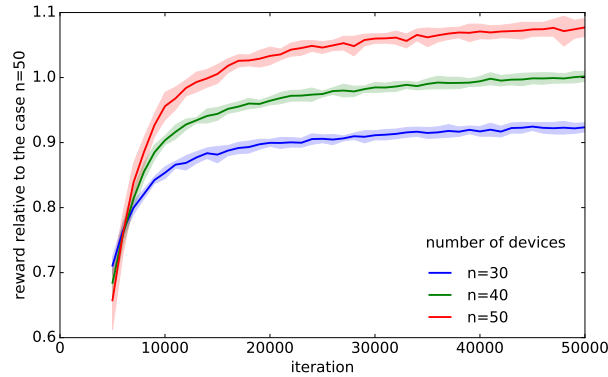

Figure 1: Relative reward to the case of number of devices $n = 50$ (with controls satisfying constraints strictly). The space of control function is $\mathbb{R}^{n+2} \to \mathbb{R}^{3n}$ for $n \in \{30, 40, 50\}$, with multiple equality and inequality constrains. The shaded area depicts the mean $\pm$ the standard deviation over five different random seeds. Reprinted from [67].

1.2 A brief introduction to multilevel Picard approximation methods

Despite the great performance of deep learning-based approximation schemes in various numerical simulations, until today, there is no rigorous mathematical analysis in the

scientific literature which proves (or disproves) the conjecture that there exists a deep learning-based approximation method which overcomes the curse of dimensionality in the numerical approximation of PDEs. However, for MLP approximation methods it has been proven in the scientific literature that such approximation methods do overcome the curse of dimensionality in the numerical approximation of semilinear PDEs with general time horizons and, in particular, the convergence results for MLP approximation methods (see [9, 37, 89, 90, 7, 53, 6, 86, 91, 87] and Section 6 below for details) have revealed that semilinear PDEs with general time horizons can be approximated without the curse of dimensionality.

Let us briefly illustrate this in the case of semilinear heat PDEs with bounded initial value functions. Let $T, c \in (0, \infty)$, let $f \colon \mathbb{R} \to \mathbb{R}$ be Lipschitz continuous, for every $d \in \mathbb{N}$ let $u_d \in C^{1,2}([0,T] \times \mathbb{R}^d, \mathbb{R})$, and assume for every $d \in \mathbb{N}$, $t \in [0,T]$, $x \in \mathbb{R}^d$ that $|u_d(t,x)| \leq c$ and

$$(\tfrac{\partial}{\partial t} u_d)(t,x) = (\Delta_x u_d)(t,x) + f(u_d(t,x)). \tag{18}$$

In the linear case where $f \equiv 0$ vanishes, it is known for a long time that classical Monte-Carlo approximation methods can approximate $u_d(T,0) \in \mathbb{R}$ with $\alpha = 1$ and $\beta = 2$ in (4). In the general nonlinear case, classical Monte-Carlo approximation methods are not applicable anymore but it has recently been shown in the scientific literature (see Hutzenthaler et al. [89] and Theorem 3 below) that for every arbitrarily small $\delta \in (0,1]$ it holds that MLP approximation methods can approximate $u_d(T,0) \in \mathbb{R}$ in the general nonlinear case with $\alpha = 1$ and $\beta = 2 + \delta$ in (4). The convergence results for MLP approximation methods in the scientific literature have thus revealed that semilinear heat PDEs can, up to an arbitrarily small polynomial order of convergence, been solved with the same computational complexity as linear heat PDEs.

In the following we briefly sketch some of the main ideas in the derivation of MLP approximation schemes. One of the key ideas in the derivation and the convergence analysis of the MLP approximation scheme is to rewrite the PDE in (18) as a stochastic fixed point equation. More formally, we note that (18) ensures that for all $t \in [0,T]$, $x \in \mathbb{R}^d$ it holds that

$$u(t,x) = \mathbb{E}\left[u\big(0, x + \sqrt{2}W_t\big) + \int_0^t f\big(u(s, x + \sqrt{2}W_{t-s})\big)\,ds\right]. \tag{19}$$

where $W \colon [0,T] \times \Omega \to \mathbb{R}^d$ is a standard Brownian motion on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ (cf., e.g., Beck et al. [8, Theorem 1.1]). Now we can also write (19) as the fixed point equation $u = \Phi(u)$ where Φ is the self-mapping on the set of all bounded functions in $C([0,T] \times \mathbb{R}^d, \mathbb{R})$ which is described through the right hand side of (19). Using Φ one can define Picard iterates $\mathfrak{u}_n$, $n \in \mathbb{N}_0$, through the recursion that for all $n \in \mathbb{N}$ it holds that $\mathfrak{u}_0 = 0$ and $\mathfrak{u}_n = \Phi(\mathfrak{u}_{n-1})$. In the next step we note that a telescoping sum argument shows that for all $n \in \mathbb{N}$ it holds that

$$\mathfrak{u}_n = \mathfrak{u}_1 + \sum_{k=1}^{n-1} [\mathfrak{u}_{k+1} - \mathfrak{u}_k] = \Phi(0) + \sum_{l=1}^{n-1} [\Phi(\mathfrak{u}_k) - \Phi(\mathfrak{u}_{k-1})]. \tag{20}$$

MLP approximations are then derived by approximating the expectations in (20) within the fixed point mapping Φ by means of Monte-Carlo approximations with different levels of accuracy.

The procedure in (20) is inspired by multilevel Monte Carlo (MLMC) approximations in the scientific literature (see Heinrich [74], Heinrich & Sindambiwe [76], Giles [51] and, e.g., [75, 52] and the references mentioned therein). There are, however, also several differences when comparing MLP approximations to "classical" MLMC approximations. In particular, we note that MLP approximations are full history recursive in the sense that for all $n \in \mathbb{N}$ we have that computations of realizations of MLP approximations in the n-th iterate require realizations for MLP approximations in the 1st, 2nd, ..., $(n-1)$-th iterate the analysis of MLP approximations (see (74) in Theorem 3 below for details). Taking this into account, the convergence analysis for MLP approximations in the scientific literature turns out to be more much subtle, and we refer to Section 6 below for a sketch of the some of the main ideas for the complexity analysis of MLP approximations and also for references to research articles on MLP approximations in the scientific literature.

In the comparison between classical linear Monte-Carlo methods and MLP approximation methods in (18) above we have restricted ourselves just for simplicity to the problem of approximating semilinear *heat* PDEs with *bounded* solutions at the space-time point $t = T$, $x = 0$ and similar results hold in the case of much more general classes of PDEs and more general approximation problems. Until today, MLP approximation schemes are the only approximation schemes in the scientific literature for which it has been proved that they do overcome the curse of dimensionality in the numerical approximation of semilinear PDEs with general time horizons. We refer to Section 6 for further details and, in particular, for a comprehensive literature overview on research articles on MLP approximation methods.

2 General remarks about algorithms for solving PDEs in high dimensions

We begin with a brief overview of the algorithms that have been developed for high dimensional PDEs.

Special classes of PDEs: The representative special classes of PDEs which we review within this subsection are second-order linear parabolic PDEs of the Kolmogorov type and first-order Hamilton-Jacobi equations.

Consider the linear parabolic PDE with a terminal condition specified at $t = T$ described by

$$\frac{\partial u}{\partial t} + \frac{1}{2}\mathrm{Tr}\big(\sigma\sigma^{\mathrm{T}}(\mathrm{Hess}_x\, u)\big) + \langle \nabla_x u, \mu \rangle + f = 0, \quad u(T, \cdot) = g(\cdot) \tag{21}$$

Our objective is to compute $u(0, \cdot)$. For this purpose, we consider the diffusion process described by the stochastic differential equation (SDE)

$$dX_t = \mu(t, X_t)\, dt + \sigma(t, X_t)\, W_t. \tag{22}$$

The solution to the PDE in (21) can be expressed as an expectation in the sense that

$$u(0, x) = \mathbb{E}\left[g(X_T) + \int_0^T f(s, X_s)ds \,\Big|\, X_0 = x \right]. \tag{23}$$

This is the classical Feynman-Kac formula [98, 119]. Using this formula, one can readily evaluate $u(0, \cdot)$ using Monte Carlo without suffering from CoD.

In a similar spirit, solutions of the Hamilton-Jacobi equations can also be expressed using the Hopf formula. Consider the PDE

$$\frac{\partial u}{\partial t} + H(\nabla u) = 0, \qquad u(x, 0) = g(x). \tag{24}$$

Assume that H is convex. Then we have the Hopf formula:

$$u(x, t) = \inf_y \left(g(y) + tH^* \left(\frac{x - y}{t} \right) \right). \tag{25}$$

where H^* is the Legendre transform of H (see Evans [43]). The right hand side of the above equation can be computed using some optimization algorithms. A particularly attractive algorithm along these lines was developed in Darbon & Osher [32].

Control problems: The first application of deep learning to solving scientific computing problems in high dimensions was in the area of stochastic control. In 2016, Han and E [67] developed a deep neural network-based algorithm for stochastic control problems. The reason for choosing stochastic control was its very close analogy with the setup of deep learning, as we will see later (see Section 4 below). Deep learning-based algorithms for deterministic control problems were first developed in [117].

Schrödinger equation for spins and electrons: In an influential paper, Carleo and Troyer developed an algorithm for solving the Schrödinger equation for spins using the restricted Boltzmann machine (RBM) as the trial function. The variational Monte Carlo (VMC) approach was used for ground-state calculations. To solve the dynamic equation, the least square approach was used, i.e., the total integral of the square of the residual was used as the loss function [22].

For many-electron Schrödinger equation, the story is quite different. The configuration space is now continuous, instead of being discrete. In addition, the wave function should satisfy the anti-symmetry constraint. This has proven to be a difficult issue in solving the Schrödinger equation. In [73], Han, Zhang, and E developed a deep neural network-based methodology for computing the ground states. Compared with traditional VMC, a permutation-symmetric neural network-based ansatz is used for the Jastrow factor. The anti-symmetric part was treated in a rather simplified fashion. This has been improved in the work [112, 122, 81].

Despite these progresses, it is fair to say that we are still at an early stage for developing neural network-based algorithms for the many-body Schrödinger equation. Since the issues for solving the Schrödinger equation are quite specialized, we will not discuss them further in this review.

Nonlinear parabolic PDEs: The first class of algorithms developed for general nonlinear parabolic PDEs with general time horizons in really high dimensions ($d \geq 40$) is the *multilevel Picard method* [37, 38, 89]. At this moment, this is also the only algorithm for which a relatively complete theory has been established (see Section 6 below). Among other things, this theory offers a proof that the MLP method overcomes

9

CoD. Shortly after, E, Han, and Jentzen developed the deep neural network-based *Deep BSDE method*, by making use of the connection between nonlinear parabolic equations and backward stochastic differential equations (BSDE) [36, 70]. This was the first systematic application of deep learning to solving general high dimensional PDEs. Later, Sirignano and Spiliopoulos developed an alternative deep learning-based algorithm using the least squares approach [132], extending the work of Carleo and Troyer [22] to general PDEs. Such deep learning-based approximation methods for PDEs have also been extended in different ways and to other parabolic and even elliptic problems; see, e.g., [5, 10, 126, 4, 12, 85, 25, 78, 73, 94, 72].

Some special semilinear parabolic PDEs can be formulated in terms of branching processes. One such example is the Fisher-KPP (Kolmogorov-Petrovski-Piscounov) equation [48, 103, 115]. For such PDEs, Monte Carlo methods can be developed, and such Monte Carlo approximation algorithms overcome the CoD (see [133, 142, 77, 80, 26, 139, 79]) in the specific situation where the time horizon and/or the PDE nonlinearity is sufficiently small.

Variational problems: It is fairly straightforward to construct neural network-based algorithms for solving variational problems. One way of doing this is simply to use the Ritz formulation. The "Deep Ritz method", to be discussed below, is such an example; see E & Yu [41] and also Khoo et al. [99]. It is natural to ask whether one can develop a similar Galerkin formulation, i.e., using a weak form of the PDE. In fact, [41] was written as a preparation for developing what would be a "Deep Galerkin method". However, formulating robust neural network algorithms using a weak form has proven to be quite problematic. The difficulty seems to be analogous to the ones encountered in generative adversarial networks (GAN); cf. Goodfellow et al. [62]. For some progress in this direction we refer to the article Zhang et al. [143]. It should also be noted that even though the deep learning-based methodology proposed in Sirignano & Spiliopoulos [132] was named as a "Deep Galerkin method", the methodology in [132] is somehow based on a least square formulation rather than a Galerkin formulation.

Parametric PDEs: One of the earliest applications of deep learning to PDEs is in the study of parametric PDEs. In [100], Khoo, Lu, and Ying developed a methodology for solving low dimensional PDEs with random coefficients in which the neural network models are used to parametrize the random coefficients. Recently the neural networks are also applied to solve low-dimensional stochastic PDEs [144]. This is a promising direction thought it will not be covered in this review. Another closely related area is solving inverse problems governed by PDEs, which is intrinsically high dimensional as well. Recent works [127, 45, 46, 101, 27] have demonstrated the advantages of approximating the forward and inverse maps with carefully designed neural networks.

Game theory A stochastic game describes the behavior of a population of interactive agents among which everyone makes his/her optimal decision in a common environment. Many scenarios in finance, economics, management science, and engineering can be formulated as stochastic games. With a finite number of agents, the Markovian Nash equilibrium of a game is determined by a coupled system of parabolic PDEs. To solve these problems, Han et al. extend the Deep BSDE method in [68, 69] with the idea of fictitious

play. In a different direction, with an infinite number of agents and no common noise, one can use the mean-field game theory developed in [107, 108, 109, 84, 83] to reduce the characterization of the Nash equilibrium to two coupled equations: a Hamilton-Jacobi-Bellman equation and a Fokker-Planck equation. Neural network-based algorithms have been developed in [23, 24, 131, 111] to solve these equations.

Besides the literature mentioned above, certain deep learning-based approximation methods for PDEs have been proposed (see, e.g., [16, 34, 47, 49, 63, 92, 113, 114, 118, 124, 125, 21, 145]) and various numerical simulations for such methods suggest that deep neural network approximations might have the capacity to indeed solve high dimensional problems efficiently. Actually, the attempts of applying neural networks for solving PDEs can be dated back to the 90s (cf., e.g., [110, 136, 106, 96]), nevertheless, with a focus on low-dimensional PDEs. Apart from neural networks, there are also other attempts in literature in solving high dimensional PDEs with limited success (see, e.g., [3, 137, 146, 19, 57, 33, 14, 56, 20, 50, 55, 44, 28, 31, 29, 30, 105, 123, 66, 58, 60, 59, 141, 140, 130, 18]).

This review will be focused on nonlinear parabolic PDEs and related control problems. There are two main reasons for choosing these topics. The first is that these classes of problems are fairly general and have general interest. The second is that the study of high dimensional problems is in better shape for these classes of problems, compared to others (e.g., the Schrödinger equation discussed above).

We should also mention that the heart of reinforcement learning is solving approximately the Bellman equation, even though reinforcement learning algorithms are not always formulated that way. The dimensionality in these problems is often very high. This is another topic that will not be covered in this review.

3 The Deep BSDE method

The Deep BSDE method was the first deep learning-based numerical algorithm for solving general nonlinear parabolic PDEs in high dimensions [36, 70]. It begins by reformulating the PDE as a stochastic optimization problem. This is done with the help of BSDEs, hence the name "Deep BSDE method". As a by-product, the Deep BSDE method is also an efficient algorithm for solving high dimensional BSDEs.

3.1 PDEs and BSDEs

Consider the semilinear parabolic PDE

$$\frac{\partial u}{\partial t} + \frac{1}{2}\mathrm{Tr}\big(\sigma\sigma^{\mathrm{T}}(\mathrm{Hess}_x u)\big) + \langle \nabla u, \mu \rangle + f\big(t, x, u, \sigma^{\mathrm{T}}(\nabla_x u)\big) = 0, \quad u(T, x) = g(x). \quad (26)$$

In the same way as in Section 2 above, we consider the diffusion process 扩散过程

$$X_t = \xi + \int_0^t \mu(s, X_s)\, ds + \int_0^t \sigma(s, X_s)\, dW_s. \quad (27)$$

11

Using Itô's lemma, we obtain that

$$u(t, X_t) - u(0, X_0)$$
$$= -\int_0^t f\big(s, X_s, u(s, X_s), [\sigma(s, X_s)]^{\mathrm{T}}(\nabla_x u)(s, X_s)\big)\, ds \tag{28}$$
$$+ \int_0^t [\nabla u(s, X_s)]^{\mathrm{T}}\, \sigma(s, X_s)\, dW_s.$$

To proceed further, we recall the notion of backward stochastic differential equations (BSDEs)

$$\begin{cases} X_t = \xi + \int_0^t \mu(s, X_s)\, ds + \int_0^t \sigma(s, X_s)\, dW_s, & \text{前向} \tag{29} \\[2mm] Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)\, ds - \int_t^T (Z_s)^{\mathrm{T}}\, dW_s & \text{反向} \tag{30} \end{cases}$$

存在一个唯一的随机过程

introduced by Pardoux and Peng [120]. It was shown in [120, 121] that there is an up-to-equivalence unique adapted stochastic process (X_t, Y_t, Z_t), $t \in [0, T]$, with values in $\mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d$ that satisfies the pair of stochastic equations in (29)–(30) above.

The connection between the BSDE in (29)–(30) and the PDE in (26) is as follows [120, 121]. Let $u \colon [0, T] \times \mathbb{R}^d \to \mathbb{R}$ be a solution of the PDE in (26). If we define

$$Y_t = u(t, X_t) \qquad \text{and} \qquad Z_t = [\sigma(t, X_t)]^{\mathrm{T}}(\nabla_x u)(t, X_t). \tag{31}$$

Then (Y_t, Z_t), $t \in [0, T]$, is a solution for the BSDE in (29)–(30). With this connection in mind, one can formulate the PDE problem as the following variational problem:

$$\inf_{Y_0, \{Z_t\}_{0 \le t \le T}} \mathbb{E}\big[|g(X_T) - Y_T|^2\big], \tag{32}$$

$$\text{s.t.} \quad X_t = \xi + \int_0^t \mu(s, X_s)\, ds + \int_0^t \Sigma(s, X_s)\, dW_s, \tag{33}$$

$$Y_t = Y_0 - \int_0^t h(s, X_s, Y_s, Z_s)\, ds + \int_0^t (Z_s)^{\mathrm{T}}\, dW_s. \tag{34}$$

The minimizer of this variational problem is the solution to the PDE and vice versa.

3.2 The Deep BSDE Method

A key idea of the Deep BSDE method is to approximate the unknown functions $X_0 \mapsto u(0, X_0)$ and $X_t \mapsto [\sigma(t, X_t)]^{\mathrm{T}}(\nabla_x u)(t, X_t)$ by feedforward neural networks ψ and ϕ. To that purpose, we work with the variational formulation described above and discretize time, say using the Euler scheme on a grid $0 = t_0 < t_1 < \ldots < t_N = T$:

$$\inf_{\psi_0, \{\phi_n\}_{n=0}^{N-1}} \mathbb{E}|g(X_T) - Y_T|^2, \tag{35}$$

$$\text{s.t.} \quad X_0 = \xi, \quad Y_0 = \psi_0(\xi), \tag{36}$$

$$X_{t_{n+1}} = X_{t_i} + \mu(t_n, X_{t_n})\Delta t + \sigma(t_n, X_{t_n})\Delta W_n, \tag{37}$$

$$Z_{t_n} = \phi_n(X_{t_n}), \tag{38}$$

$$Y_{t_{n+1}} = Y_{t_n} - f(t_n, X_{t_n}, Y_{t_n}, Z_{t_n})\Delta t + (Z_{t_n})^{\mathrm{T}}\Delta W_n. \tag{39}$$

At each time slide t_n, we associate a subnetwork. We can stack all these subnetworks together to form a deep composite neural network. This network takes the paths $\{X_{t_n}\}_{0\leq n\leq N}$ and $\{W_{t_n}\}_{0\leq n\leq N}$ as the input data and gives the final output, denoted by $\hat{u}(\{X_{t_n}\}_{0\leq n\leq N}, \{W_{t_n}\}_{0\leq n\leq N})$, as an approximation to $u(t_N, X_{t_N})$.

The error in the matching of the given terminal condition defines the loss function

$$l(\theta) = \mathbb{E}\left[\left|g(X_{t_N}) - \hat{u}\big(\{X_{t_n}\}_{0\leq n\leq N}, \{W_{t_n}\}_{0\leq n\leq N}\big)\right|^2\right]. \tag{40}$$
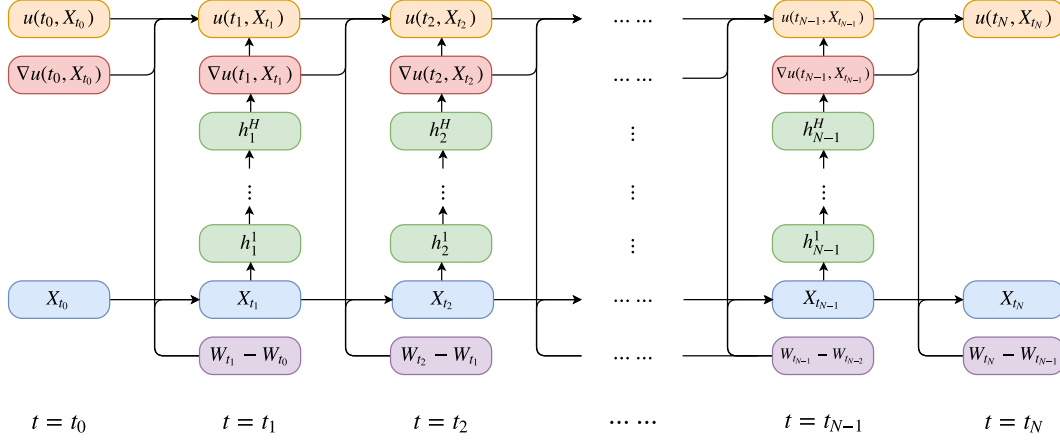

Figure 2: Network architecture for solving parabolic PDEs. Each column corresponds to a subnetwork at time $t = t_n$. The whole network has $(H + 1)(N - 1)$ layers in total that involve free parameters to be optimized simultaneously. Reprinted from [70].

From the viewpoint of machine learning, this neural network model has several interesting features.

1. It does not require us to generate training data beforehand. The paths $\{W_{t_n}\}_{0\leq n\leq N})$ play the role of the data and they are generated on the fly. For this reason, one can think of this model as a model with an infinite amount of data.

2. For the same reason, it is very natural to use stochastic gradient descent (SGD) to train the network.

3. The network has a very natural "residual neural network" structure embedded in the stochastic difference equations. For example:

$$\begin{aligned} &u(t_{n+1}, X_{t_{n+1}}) - u(t_n, X_{t_n}) \\ &\approx -f\big(t_n, X_{t_n}, u(t_n, X_{t_n}), \phi_n(X_{t_n})\big)\,\Delta t_n + (\phi_n(X_{t_n}))^{\mathrm{T}}\Delta W_n. \end{aligned} \tag{41}$$

3.3 Some numerical examples

Next we examine the effectiveness of the algorithms described above. We will discuss two examples: The first is a canonical benchmark problem, the linear-quadratic control

problem (LQG). The second is a nonlinear Black-Scholes model. We use the simplest implementation of Deep BSDE: Each subnetwork has 3 layers, with 1 input layer (d-dimensional), 2 hidden layers (both $d + 10$-dimensional), and d-dimensional output. We choose the rectifier function (ReLU) as the activation function and optimize with the Adam method [102].

We will report the mean and the standard deviation of the relative error from 5 independent runs with different random seeds.

LQG (linear quadratic Gaussian)

Consider the stochastic dynamic model in 100 dimension:

$$dX_t = 2\sqrt{\lambda}\, m_t\, dt + \sqrt{2}\, dW_t, \tag{42}$$

with cost functional:

$$J(\{m_t\}_{0 \leq t \leq T}) = \mathbb{E}\Big[\int_0^T \|m_t\|_2^2\, dt + g(X_T)\Big]. \tag{43}$$

The associated HJB equation is given by

$$\frac{\partial u}{\partial t} + \Delta u - \lambda \|\nabla u\|_2^2 = 0 \tag{44}$$

The solution to this HJB equation can be expressed as

$$u(t, x) = -\frac{1}{\lambda} \ln\Big(\mathbb{E}\Big[\exp\Big(-\lambda g(x + \sqrt{2}W_{T-t})\Big)\Big]\Big). \tag{45}$$

This formula can be evaluated directly using Monte Carlo. Therefore this problem serves as a good model for validating algorithms. The results from the Deep BSDE method is shown in Figure 3.
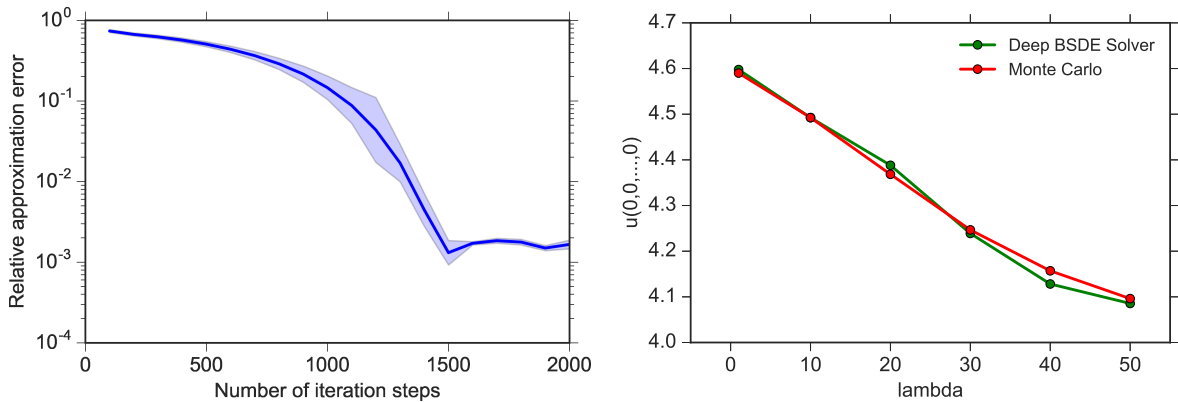

Figure 3: Left: Relative error of the Deep BSDE method for $u(t{=}0, x{=}(0, \ldots, 0))$ when $\lambda = 1$, which achieves 0.17% relative error in a run time of 330 seconds. Right: Optimal cost $u(t{=}0, x{=}(0, \ldots, 0))$ for different values of λ. The shaded area depicts the mean $\pm$ the standard deviation over five different random seeds. Reprinted from [70].

14

We see that the accuracy of the trained solution improves along the training curve before it saturates.

Black-Scholes equation with default risk

The pricing model for financial derivatives should take into account the whole basket of the underlies, which results in high dimensional PDEs. In addition, the classical Black-Scholes model can and should be augmented by some important factors in real markets, including the effect of default, transactions costs, uncertainties in the model parameters, etc. Taking into account these effects leads to nonlinear Black-Scholes type of models.

We study a particular case of the recursive valuation model with default risk [35, 15]. The underlying asset price moves as a geometric Brownian motion, and the possible default is modeled by the first jump time of a Poisson process. The claim value is modeled by the nonlinear Black-Scholes model with

$$f\big(t, x, u(t,x), \sigma^{\mathrm{T}}(t,x)\nabla u(t,x)\big) = -\,(1-\delta)\,Q(u(t,x))\,u(t,x) - R\,u(t,x). \qquad (46)$$

where Q is some nonlinear function. We will consider the fair price of an European claim based on 100 underlying assets conditional on no default having occurred yet. This leads to a problem with $d = 100$. Figure 4 presents the results of Deep BSDE and multilevel Picard for this nonlinear Black-Scholes equation for $d = 100$. Reported in the figure is the approximate solution at $t = 0$, $x = (100, \ldots, 100)$. For this problem we cannot find the "exact solution". Therefore we use the results of the two different methods to calibrate each other.
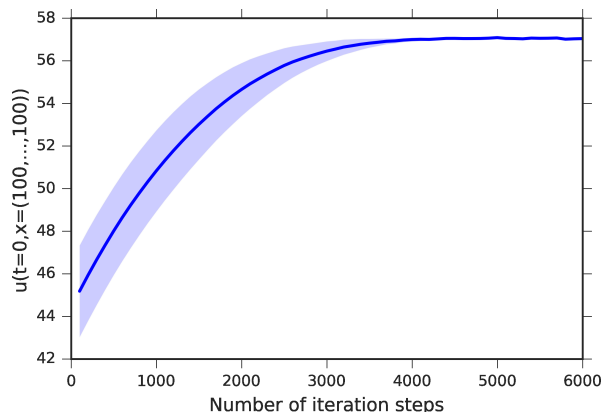

Figure 4: Approximation of $u(t{=}0, x{=}(100, \ldots, 100))$ as a function of the number of iteration steps. The Deep BSDE method achieves a relative error of size 0.46% in a runtime of 617 seconds. The shaded area depicts the mean $\pm$ the standard deviation over five different random seeds. Reprinted from [70].

3.4   Analysis of the Deep BSDE method

There is not yet a complete theory for the analysis of the Deep BSDE method. We will review the existing results that have been obtained so far. Here instead of bounding the cost required for reducing the error to certain tolerance ε, we bound the error associated

with certain hyper-parameters, such as the time step size Δt and the size of the neural network models. The basic strategy is to reduce the problem to bounding the generalization error for supervised learning [40]. In order to do that, we need to do the following: (1) Estimate the error in the time discretization. (2) Prove that the functions that need to be approximated using neural networks belong to the right function class and bound their norms in that function class. (3) Adapt the analysis for supervised learning problems to the current setting. For two-layer neural network models, the function class is the Barron space [39].

At this point, only step (1) has been accomplished.

Theorem 1 (A Posteriori Estimates [71]). *Under some assumptions, there exists a constant C, independent of h, d, and m, such that for sufficiently small h,*

$$
\begin{aligned}
&\sup_{t\in[0,T]} (\mathbb{E}|X_t - \hat{X}_t^\pi|^2 + \mathbb{E}|Y_t - \hat{Y}_t^\pi|^2) + \int_0^T \mathbb{E}|Z_t - \hat{Z}_t^\pi|^2 \, \mathrm{d}t \\
&\leq C[h + \mathbb{E}|g(X_T^\pi) - Y_T^\pi|^2],
\end{aligned}
\tag{47}
$$

where $\hat{X}_t^\pi = X_{t_i}^\pi$, $\hat{Y}_t^\pi = Y_{t_i}^\pi$, $\hat{Z}_t^\pi = Z_{t_i}^\pi$ for $t \in [t_i, t_{i+1})$.

Theorem 2 (Upper Bound of Optimal Loss [71]). *Under some assumptions, there exists a constant C, independent of h, d, and m, such that for sufficiently small h,*

$$
\begin{aligned}
&\mathbb{E}|g(X_T^\pi) - Y_T^\pi|^2 \\
&\leq C\Big\{h + \mathbb{E}|Y_0 - \mu_0^\pi(\xi)|^2 + \sum_{i=0}^{N-1} \mathbb{E}|\mathbb{E}[\tilde{Z}_{t_i}|X_{t_i}^\pi, Y_{t_i}^\pi] - \phi_i^\pi(X_{t_i}^\pi, Y_{t_i}^\pi)|^2 h\Big\},
\end{aligned}
\tag{48}
$$

where $\tilde{Z}_{t_i} = h^{-1}\mathbb{E}[\int_{t_i}^{t_{i+1}} Z_t \, \mathrm{d}t|\mathcal{F}_{t_i}]$. If b and σ are independent of Y, the term $\mathbb{E}[\tilde{Z}_{t_i}|X_{t_i}^\pi, Y_{t_i}^\pi]$ can be replaced with $\mathbb{E}[\tilde{Z}_{t_i}|X_{t_i}^\pi]$.

4 Control problems in high dimensions

One of the areas that high dimensional problems are often encountered is optimal control. In fact the term "curse of dimensionality" was first coined by Richard Bellman in the context of dynamic programming for control problems [13]. Regarding CoD, there is an important difference between open- and closed-loop controls that we now explain. fConsider the optimal control problem with a finite horizon T:

$$
\begin{cases}
\min_{u} & g(x(T)) + \int_0^T L(t, x(t), u(t))dt, \\
\text{subject to} & \dot{x}(t) = f(t, x, u), \\
& x(0) = x_0.
\end{cases}
\tag{49}
$$

Here $x : [0, T] \to \mathcal{X} \subseteq \mathbb{R}^d$ is the state, $u : [0, T] \to \mathcal{U} \subseteq \mathbb{R}^m$ is the control, $g : \mathcal{X} \to \mathbb{R}$ is the terminal cost, and $L : [0, T \times \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the running cost. For fixed x_0, the

problem above can be thought of as a two-point boundary value problem over the time interval $[0, T]$ and the optimal control can be sought in the form:

$$u = u^*(t, x^*(t)) \tag{50}$$

where x^* denotes the optimal path. We refer to [128] for a review of the numerical algorithms for solving this kind of two-point boundary value problems. In this case, CoD is not really an issue since the dimensionality of the independent variable is just 1. Controls of the form (50) is called an open-loop control. In this case, the optimal control is only known along the optimal path. Once the system deviates from the optimal path, one has to either recompute the optimal control or force the system back to the optimal path. In many applications, one prefers a closed-loop control or feedback control

$$u = u^*(t, x), \tag{51}$$

where the optimal control is known as every point in the state space. Closed-loop controls are functions of the state variable and this is where the CoD problem arises. To characterize open- and closed-loop controls, let

$$\tilde{H}(t, x, \lambda, u) := L(t, x, u) + \lambda^T f(t, x, u) \tag{52}$$

be the extended Hamiltonian associated with this control problem, and define

$$u^*(t, x; \lambda) = \arg \min_{u \in \mathcal{U}} H(t, x, \lambda, u). \tag{53}$$

Here λ is the co-state variable. An important result is that the solution to the optimal control problem satisfies Pontryagin's Minimum Principle:

$$\begin{cases} \dot{x}(t) = \dfrac{\partial \tilde{H}}{\partial \lambda} = f(t, x, u^*(t, x, \lambda)), \\[2mm] \dot{\lambda}(t) = -\dfrac{\partial \tilde{H}}{\partial x}(t, x, \lambda, u^*(t, x, \lambda)), \\[2mm] \dot{v}(t) = -L(t, x, u^*(t, x, \lambda)), \end{cases} \tag{54}$$

with the boundary conditions

$$x(0) = x_0, \ \lambda(T) = \nabla g(x(T)), \ v(T) = g(x(T)). \tag{55}$$

Denote by V the value function of the control problem:

$$V(t, x) := \inf_{u \in \mathcal{U}} \left\{ g(y(T)) + \int_t^T L(\tau, y, u) d\tau \right\}, \tag{56}$$

subject to $\dot{y}(\tau) = f(\tau, y, u)$ and $y(t) = x$. Define the Hamiltonian:

$$H^*(t, x, \lambda) := H(t, x, \lambda, u^*). \tag{57}$$

17

The HJB equation can be written as

$$V_t(t, x) + H^*(t, x, V_x) = 0 \tag{58}$$

with the terminal condition $V(T, x) = F(x)$. The co-state and the closed-loop optimal control is given in terms of the value function by

$$\lambda(t) = \nabla_x V(t, x(t)), \tag{59}$$

$$u^*(t, x) = \arg\min_{u \in \mathcal{U}} H(t, x, \nabla_x V, u). \tag{60}$$

To obtain an accurate approximation to the closed-loop control, we need to solve the control problem for a large set of initial conditions, if not all. The formulation (49) is for a single initial condition. To extend it to all initial conditions, we consider instead the problem:

$$\min_u \mathbb{E}_{x_0 \sim \mu} \left(g(x(T)) + \int_0^T L(t, x(t), u(t, x(t))) dt \right) \tag{61}$$

subject to $\dot{x}(t) = f(t, x(t), u(t, (t)), x(0) = x_0$. Here the optimization is over all possible policy functions u. One question that naturally arises is how we should choose the distribution μ for the initial condition. Clearly we are only interested in states whose value functions are not very big. Therefore one possible choice is the Gibbs distribution for the value function:

$$\mu = \frac{1}{Z} e^{-\beta V} \tag{62}$$

where Z is a normalization factor. β is a positive hyper-parameter.

Unlike the stochastic case for which the training data is obtained on-the-fly, here one needs to address the issue of data generation explicitly. The following strategy was proposed in [117, 97]:

- The two-point boundary value problem (54)-(55) is solved to obtain the training data.

- A neural network model is trained for the value function.

In practice, (54)-(55) is not an easy problem to solve, and it is important to look for a small yet representative training dataset. The following ideas were proposed and tested in [117, 97].

The first is called "warm start". The basic idea is to choose initializations for the iterative algorithms for (54)-(55) to help guarantee convergence. For example one can start with small values of T in which case the convergence of the iterative algorithms is much less of an issue. One can use simple extrapolations of these solutions on longer time intervals as initializations and obtain converged solutions on longer intervals. This process can be continued. In addition, once a reasonable approximation of the policy and value functions is obtained, one can use that to help initializing the two-point boundary value problem.

The second is to explore adaptive sampling. It has been explored in a similar context [147]. As for all adaptive algorithms, the key issue is an error indicator: The larger

18

the error, the more data are needed. [147] uses the variance of the predictions from an ensemble of similar machine learning models as the error indicator, A sophisticated error indicator that makes use of the variance of the gradient of the loss function was proposed in [117]. Another idea is to simply use the magnitude of the gradient of the value function as the error indicator.

5   Ritz, Galerkin, and least squares

The Ritz, Galerkin, and least squares formulations are among the most commonly used frameworks for designing numerical algorithms for PDEs. The Ritz formulation is based on a variational principle. The Galerkin formulation is based on the weak formulation of a PDE that involves both the trial and test functions. Least squares formulation is a very general approach for turning a PDE problem into a variational problem by minimizing the squared residual of the PDE. It has the advantage of being general and straightforward to think about. However, in classical numerical analysis, it is often the least preferred since the numerical problem obtained this way tends to be worse conditioned than the ones using Ritz or Galerkin formulation. Designing machine learning-based algorithms using Ritz and least square formulations is rather straightforward. Since there is a variational principle behind both the Ritz and least square formulations, one can simply replace the space of trial functions for these variational principles by the hypothesis space in machine learning models. Since machine learning is also a fundamentally optimization-based approach, the integration of machine learning with variational methods for PDEs is quite seamless. Indeed these were among the first set of ideas that were proposed for machine learning-based numerical algorithms for PDEs [22, 41, 132]. For the same reason, designing machine learning-based algorithms using the Galerkin formulation is a different matter, since Galerkin is not an optimization-based approach. Rather it is based on a weak formulation using test functions. The closest machine learning model to the Galerkin formulation is the Wasserstein GAN (WGAN) [1, 2]: In WGAN, the discriminator plays the role of the test function; the generator plays the role of the trial function.

5.1   The Deep Ritz method

The Deep Ritz method was proposed in [41]. Consider the variational problem [43]

$$\min_{u \in H} I(u) \tag{63}$$

where

$$I(u) = \int_{\Omega} \left(\frac{1}{2} |\nabla u(x)|^2 - f(x)u(x) \right) dx \tag{64}$$

and H is the set of admissible functions (also called trial function, here represented by u), f is a given function, representing external forcing to the system under consideration. It is understood that boundary conditions are incorporated into the definition of H. The Deep Ritz method consists of the following components:

1. Deep neural network-based representation of the trial function.

19

2. A numerical quadrature rule for the functional.

3. An algorithm for solving the final optimization problem.

Each component is relatively straightforward. One can take the usual neural network models to represent the trial function. In high dimensions one needs an effective Monte Carlo algorithm to discretize the integral in (64). The interplay between the discretization of the integral and the discretization of the trial function using neural network models is an interesting issue that requires further attention. Finally, SGD can be used naturally, similar to the situation in Deep BSDE: The integral in the functional (64) plays the role of the expectation in Deep BSDE. One notable issue is the choice of the activation function. ReLU activation does not perform well due to the discontinuity in its derivative. It has been observed that the activation function $\sigma_3(z) = \max(z, 0)$ performs much better than ReLU. More careful study is needed on this issue also. One feature of the Deep Ritz method that potentially makes it interesting even for low dimensional problems is that it is mesh-free and naturally adaptive. To examine this we consider the well-known crack problem: Computing the displacement around a crack. To this end, we consider the Poisson equation:

$$
\begin{aligned}
-\Delta u(x) &= 1, & x \in \Omega \\
u(x) &= 0, & x \in \partial\Omega
\end{aligned}
\tag{65}
$$

where $\Omega = (-1, 1) \times (-1, 1) \setminus [0, 1) \times \{0\}$. The solution to this problem suffers from the well-known "corner singularity" caused by the nature of the domain [134]. A simple asymptotic analysis shows that at the origin, the solution behaves as $u(x) = u(r, \theta) \sim r^{\frac{1}{2}} \sin \frac{\theta}{2}$ [134]. Models of this type have been extensively used to help developing and testing adaptive finite element methods. Here the essential boundary condition causes some problems. The simplest idea is to just use a penalty method and consider the modified functional

$$
I(u) = \int_{\Omega} \left(\frac{1}{2} |\nabla_x u(x)|^2 - f(x) u(x) \right) dx + \beta \int_{\partial\Omega} u(x)^2 ds.
\tag{66}
$$

An acceptable choice is $\beta = 500$. The results from the Deep Ritz method with 811 parameters in the neural network model and the finite difference method with $\Delta x_1 = \Delta x_2 = 0.1$ ($1,681$ degrees of freedom) are shown in Figure 5. More quantitative comparisons can be found in [41]. Of course adaptive numerical methods are very well developed for solving problems with corner singularities and even more general singular problems. Nevertheless, this example shows that Deep Ritz is potentially a naturally adaptive algorithm. There are also a number of problems that need to be addressed in future work:

1. The variational problem that results from Deep Ritz is usually not convex even when the original problem is.

2. At the present time, there are no consistent conclusions about the convergence rate.

3. The treatment of the essential boundary condition is not as simple as the traditional methods.

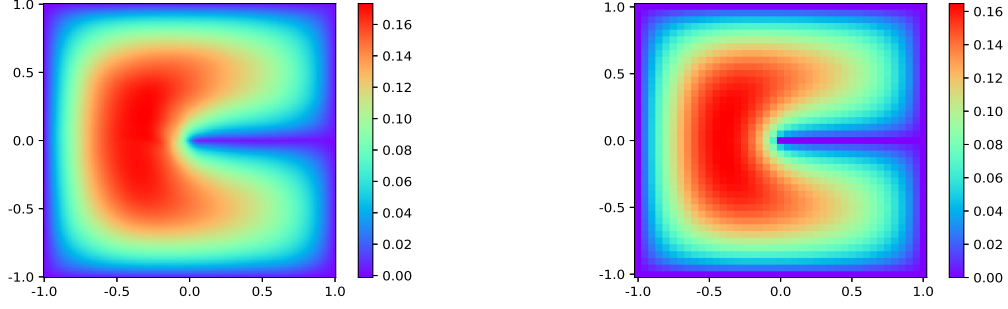Some analysis of the Deep Ritz method has been carried out in [116].

Figure 5: Solutions computed by two different methods. On the left is Deep Ritz with 811 parameters. On the right is the solution of the finite difference method on a uniform grid with 1681 parameters. Reprinted from [41].

5.2 The least square formulation

The least square approach was used in [22] for solving the dynamic Schrödinger equation and was subsequently developed more systematically in [132] (although [132] referred to it as Galerkin method). The basic idea is very simple: Solving the PDE

$$Lu = f \tag{67}$$

over a domain Ω in $\mathbb{R}^d$ can be formulated equivalently as solving the variational problem for the functional

$$J(u) = \int_\Omega \|Lu - f\|^2 \mu(dx) \tag{68}$$

where μ is a suitably chosen probability distribution on Ω. μ should be non-degenerate and readily sampled. With this, the least square formulation looks very similar to the Ritz formulation with J replacing the functional I.

5.3 The Galerkin formulation

The starting point of the Galerkin approximation is the weak form of (67):

$$a(u, \phi) = (Lu, \phi) = (f, \phi), u \in H_1, \phi \in H_2 \tag{69}$$

where H_1 and H_2 are the trial and test function spaces respectively, ϕ is an arbitrary test function in H_2, $(\cdot, \cdot)$ is the standard L^2 inner product for functions. Usually some integration by parts is applied. For example, if $L = -\Delta$, then except for boundary terms, one has

$$a(u, \phi) = (\nabla u, \nabla \phi) \tag{70}$$

Therefore this formulation only involves first order derivatives. The most important feature of the Galerkin formulation is that involves the test function. In this spirit, the Wasserstein GAN can also be regarded as an example of the Galerkin formulation. Given

21

a set of data $\{x_j, j = 1, 2, \cdots, n\}$ in $\mathbb{R}^d$ and a reference probability distribution ν^* on $\mathbb{R}^{d'}$, we look for the mapping G (the generator) from $\mathbb{R}^{d'}$ to $\mathbb{R}^d$, such that [2]

$$\int_{\mathbb{R}^{d'}} \phi(G(\boldsymbol{z}))\nu^*(d\boldsymbol{z}) = \frac{1}{n}\sum_{j=1}^{n}\phi(x_j) \tag{71}$$

for all Lipschitz functions ϕ. The test function ϕ is called the discriminator in this context. Like GAN, the most obvious reformulation of (69) is a min-max problem:

$$\min_{u \in H_1}\max_{\|\phi\|_{H_2}\leq 1}(a(u,\phi) - (f,\phi))^2. \tag{72}$$

Unfortunately this formulation is not easy to work with. The problems encountered are similar to those in WGAN. Despite this some very encouraging progress has been made and we refer to [143] for the details.

6 Multilevel Picard approximation methods for nonlinear PDEs

In the articles E et al. [37] and Hutzenthaler et al. [89] so-called fully history recursive multilevel Picard approximation methods have been introduced and analyzed (in the following we abbreviate *fully history recursive multilevel Picard* by MLP). The error analysis in the original article Hutzenthaler et al. [89] is restricted to semilinear heat PDEs with Lipschitz nonlinearities. By now in the scientific literature there are, however, a series of further articles on such MLP approximation methods (see [90, 7, 53, 6, 9, 86, 91, 38, 87]) which analyze, extend, or generalize the MLP approximation methods proposed in [37, 89] to larger classes of PDE problems such as semilinear Black-Scholes PDEs (see [90, 9]), semilinear heat PDEs with gradient dependent nonlinearities (see [86, 91]), semilinear elliptic PDE problems (see [6]), semilinear heat PDEs with non-Lipschitz continuous nonlinearities (see [7, 9]), and semilinear second-order PDEs with varying coefficient functions (see [90, 87]).

In the remainder of this section we sketch the main ideas of MLP approximation methods and to keep the presentations as easy as possible we restrict ourselves in the following presentations to semilinear heat PDEs with Lipschitz continuous nonlinearities with bounded initial values. The next result, Theorem 3 below, provides a complexity analysis for MLP approximations in the case of semilinear heat PDEs with Lipschitz continuous nonlinearities. Theorem 3 is strongly based on Hutzenthaler et al. [89, Theorem 1.1] and Beck et al. [7, Theorem 1.1].

Theorem 3. *Let* $T \in (0, \infty)$, $\Theta = \cup_{n \in \mathbb{N}}\mathbb{Z}^n$, *let* $f: \mathbb{R} \to \mathbb{R}$ *be Lipschitz continuous, for every* $d \in \mathbb{N}$ *let* $u_d \in C^{1,2}([0, T] \times \mathbb{R}^d, \mathbb{R})$ *be at most polynomially growing, assume for every* $d \in \mathbb{N}$, $t \in [0, T]$, $x \in \mathbb{R}^d$ *that*

$$(\tfrac{\partial}{\partial t}u_d)(t, x) = (\Delta_x u_d)(t, x) + f(u_d(t, x)), \tag{73}$$

let $(\Omega, \mathcal{F}, \mathbb{P})$ *be a probability space, let* $\mathcal{R}^\theta: \Omega \to [0, 1]$, $\theta \in \Theta$, *be independent* $\mathcal{U}_{[0,1]}$-*distributed random variables, let* $W^{d,\theta}: [0, T] \times \Omega \to \mathbb{R}^d$, $d \in \mathbb{N}$, $\theta \in \Theta$, *be independent*

standard Brownian motions, assume that $(\mathcal{R}^\theta)_{\theta \in \Theta}$ and $(W^{d,\theta})_{(d,\theta) \in \mathbb{N} \times \Theta}$ are independent, for every $d \in \mathbb{N}$, $s \in [0,T]$, $t \in [s,T]$, $x \in \mathbb{R}^d$, $\theta \in \Theta$ let $X^{d,\theta}_{s,t,x} \colon \Omega \to \mathbb{R}^d$ satisfy $X^{d,\theta}_{s,t,x} = x + \sqrt{2}(W^{d,\theta}_t - W^{d,\theta}_s)$, let $U^{d,\theta}_{n,M} \colon [0,T] \times \mathbb{R}^d \times \Omega \to \mathbb{R}$, $d, n, M \in \mathbb{N}_0$, $\theta \in \Theta$, satisfy for every $d, M \in \mathbb{N}$, $n \in \mathbb{N}_0$, $\theta \in \Theta$, $t \in [0,T]$, $x \in \mathbb{R}^d$ that

$$
U^{d,\theta}_{n,M}(t,x) = \sum_{k=1}^{n-1} \frac{t}{M^{n-k}} \left[\sum_{m=1}^{M^{n-k}} \left(f\left(U^{d,(\theta,k,m)}_{k,M}\left(t\mathcal{R}^{(\theta,k,m)}, X^{d,(\theta,k,m)}_{t\mathcal{R}^{(\theta,k,m)},t,x}\right)\right) \right. \right. \tag{74}
$$
$$
\left. \left. - f\left(U^{d,(\theta,-k,m)}_{k-1,M}\left(t\mathcal{R}^{(\theta,k,m)}, X^{d,(\theta,k,m)}_{t\mathcal{R}^{(\theta,k,m)},t,x}\right)\right)\right) \right] + \frac{\mathbb{1}_{\mathbb{N}}(n)}{M^n} \left[\sum_{m=1}^{M^n} \left(u_d(0, X^{d,(\theta,0,-m)}_{0,t,x}) + t\, f(0)\right)\right],
$$

and for every $d, M \in \mathbb{N}$, $n \in \mathbb{N}_0$ let $\mathfrak{C}_{d,n,M} \in \mathbb{N}_0$ be the number of function evaluations of f and $u_d(0,\cdot)$ and the number of realizations of scalar random variables which are used to compute one realization of $U^{d,0}_{n,M}(T,0) \colon \Omega \to \mathbb{R}$ (cf. [87, Corollary 4.4] for a precise definition). Then there exist $\mathfrak{N} \colon (0,1] \to \mathbb{N}$ and $c \in \mathbb{R}$ such that for all $d \in \mathbb{N}$, $\varepsilon \in (0,1]$ it holds that $\mathfrak{C}_{d,\mathfrak{N}_\varepsilon,\mathfrak{N}_\varepsilon} \leq cd^c\varepsilon^{-3}$ and $\left(\mathbb{E}\big[|U^{d,0}_{\mathfrak{N}_\varepsilon,\mathfrak{N}_\varepsilon}(T,0) - u_d(T,0)|^2\big]\right)^{1/2} \leq \varepsilon$.

In the following we add some comments on the statement in Theorem 3 above and we thereby also provide explanations for some of the mathematical objects which appear in Theorem 3.

Theorem 3 provides a complexity analysis for MLP approximations in the case of semilinear heat PDEs with Lipschitz continuous nonlinearities. In (74) in Theorem 3 the employed MLP approximations are specified. The MLP approximations in (74) aim to approximate the solutions of the PDEs in (73). The strictly positive real number $T \in (0,\infty)$ in Theorem 3 describes the time horizon of the PDEs in (73). The function $f \colon \mathbb{R} \to \mathbb{R}$ in Theorem 3 describes *the nonlinearity of the PDEs* in (73). For simplicity we restrict ourselves in Theorem 3 in this article to Lipschitz continuous nonlinearities which do only depend on the solution of the PDE but not on the time variable $t \in [0,T]$, not on the space variable $x \in \mathbb{R}^d$, and also not on the derivatives of the PDE solution. In the more general MLP analyses in the scientific literature (cf., e.g., [89, 90, 7, 53, 6, 9, 86, 87]) the nonlinearity of the PDE is allowed to depend on the time variable $t \in [0,T]$, on the space variable $x \in \mathbb{R}^d$, on the PDE solution, and also on the derivatives of the PDE solution (see [86]), and the nonlinearity of the PDE may also be not Lipschitz continuous (see [7, 9]).

The functions $u_d \colon [0,T] \times \mathbb{R}^d \to \mathbb{R}$, $d \in \mathbb{N}$, in Theorem 3 describe the exact solutions of the PDEs in (73). The linear differential operator on the right hand side of the PDE in (73) is just the Laplacian and Theorem 3 thus only applies to semilinear heat PDEs of the form (73) but the MLP analyses in the scientific literature also apply to PDEs with *much more general second-order differential operators* (cf., e.g., [87, 90]).

The approximation algorithm in (74) is a Monte-Carlo algorithm in the sense that it employs Monte-Carlo averages based on many independent identically distributed (i.i.d.) random variables. In the case of plain-vanilla standard Monte Carlo algorithms for linear PDEs the employed i.i.d. random variables are often indexed through the set of all natural numbers $\mathbb{N} = \{1,2,3,\dots\}$ where we have one random variable for each natural

number $n \in \mathbb{N}$. The approximation algorithm in (74) is somehow a *nonlinear Monte-Carlo algorithm* and in the case of such a nonlinear Monte-Carlo algorithm the situation is getting more complicated and, roughly speaking, we need more i.i.d. random variables and therefore, roughly speaking, also a larger index set[1]. More precisely, in the case of the MLP approximation algorithm in (74) we employ the set $\Theta = \cup_{n \in \mathbb{N}} \mathbb{Z}^n$ in Theorem 3 as the index set to introduce sufficiently many i.i.d. random variables. In particular, in Theorem 3 we use the family $\mathcal{R}^{\theta} \colon \Omega \to [0,1]$, $\theta \in \Theta$, of independent on $[0,1]$ continuous uniformly distributed random variables and the family $W^{d,\theta} \colon [0,T] \times \Omega \to \mathbb{R}^d$, $d \in \mathbb{N}$, $\theta \in \Theta$, of independent standard Brownian motions as random input sources for the MLP approximation algorithm in (74).

The natural numbers $\mathfrak{C}_{d,n,M} \in \mathbb{N}_0$, $d, M \in \mathbb{N}$, $n \in \mathbb{N}_0$, in Theorem 3 above aim to measure to *computational cost* for the MLP approximation algorithm in (74). Theorem 3 shows that there exists a function $\mathfrak{N} \colon (0,1] \to \mathbb{N}$ and a real number $c \in \mathbb{R}$ such that for all $d \in \mathbb{N}$, $\varepsilon \in (0,1]$ we have that the L^2-approximation error $\big(\mathbb{E} \big[|U_{\mathfrak{N}_\varepsilon, \mathfrak{N}_\varepsilon}^{d,0}(T,0) - u_d(T,0)|^2 \big] \big)^{1/2}$ between the MLP approximation $U_{\mathfrak{N}_\varepsilon, \mathfrak{N}_\varepsilon}^{d,0}(T,0)$ and the exact solution u_d of the PDE at the space-time point $t = T$, $x = 0$ is smaller or equal than the prescribed approximation accuracy ε with the computational cost $\mathfrak{C}_{d,\mathfrak{N}_\varepsilon, \mathfrak{N}_\varepsilon}$ for the MLP approximations being smaller or equal than $cd^c \varepsilon^{-3}$. The computational cost for the MLP approximation algorithm thus grows at most polynomially in the PDE dimension $d \in \mathbb{N}$ and at most cubically in the reciprocal ε^{-1} of the prescribed approximation accuracy $\varepsilon \in (0,1]$.

The more general MLP approximation [7, 89, 90, 53, 86, 87] in the scientific literature improve this statement in several ways. First, the main approximation results in the above named reference list allow the numerical approximation of the PDE solution not necessarily at the space point $x = 0$ but at much more general space points. Second, the main approximation results in the above named reference list provide explicit errors constants and explicit exponents in dependence on the constants in the assumptions for the involved functions. For instance, if the initial conditions of the PDEs under consideration are bounded functions, then Hutzenthaler et al. [89, Theorem 1.1] and Beck et al. [7, Theorem 1.1] even prove that the computation cost of the employed MLP approximation scheme *grows at most linearly in the PDE dimension*. Finally, most of the MLP approximation results in the scientific literature also prove that the computational cost of the considered MLP approximation scheme grows up to an arbitrarily small real number at most quadratically (instead of cubically as in Theorem 3 above) in the reciprocal ε^{-1} of the prescribed approximation accuracy $\varepsilon \in (0,1]$.

It should also be noted that MLP approximation schemes not only overcome the curse of dimensionality in the numerical approximation of parabolic PDEs but also in the case of *elliptic PDEs* with Lipschitz continuous nonlinearities (see Beck et al. [6]). Encouraging numerical simulations for MLP approximation schemes in the case of semilinear Black-Scholes PDEs, systems of semilinear PDEs, Allen-Cahn PDEs, and sine-Gordon type PDEs can be found in Becker et al. [9] (see also E et al. [38]).

In this article we do not provide a detailed proof for Theorem 3 but instead we refer, e.g., Hutzenthaler et al. [89, 87] for a detailed proof of Theorem 3. In addition, in the

---

[1]We remark that the set of all natural numbers $\mathbb{N}$ is a proper subset of the set $\Theta = \cup_{n \in \mathbb{N}} \mathbb{Z}^n$ in the sense that $\mathbb{N} \subsetneq \Theta$ but the set of all natural numbers $\mathbb{N}$ and the set Θ have, of course, the same cardinality

following we also briefly outline some of *the main ideas of the proof* of Theorem 3. The derivation and thereby also the mathematical analysis of the MLP approximation schemes is, roughly speaking, based on the following three steps.

(I) First, we reformulate the PDE under consideration (or, more generally, the computational problem under consideration) as a suitable stochastic fixed point equation with the unique fixed point solution of the stochastic fixed point equation being the unique solution of the PDE under consideration.

(II) Second, we approximate the unique fixed point of the stochastic fixed point equation by means of fixed point iterations according to the Banach fixed point theorem (which are referred to as Picard iterations in the context of temporal integral fixed point equations).

(III) Third, we recursively approximate the resulting fixed point iterations by means of suitable multilevel Monte-Carlo approximations resulting with the resulting Monte-Carlo approximations being full history recursive.

A key idea in the above derivation of the MLP approximation scheme is that the fixed point iterations often converge exceedingly quick, that is, with factorial convergence speed to the unique fixed point of the stochastic fixed point equation under consideration while efficient multilevel Monte Carlo approximations assure that the computation cost of the considered MLP approximation scheme grows not significantly larger than factorially. These facts made it possible to prove that MLP approximation schemes overcome the curse of dimensionality in the numerical approximation of a large class of semilinear PDEs.

Despite the great performance of deep learning-based approximation schemes in various numerical simulations, until today, MLP approximation schemes are, to the best of our knowledge, the only approximation schemes in the scientific literature for which *it has been proven that they do indeed overcome the curse of dimensionality* in the numerical approximation of semilinear PDEs with general time horizons.

7 Mathematical results for neural network approximations for PDEs

Until today, there is no complete rigorous mathematical analysis which proves (or disproves) the conjecture that there exists a deep learning-based approximation method which overcomes the curse of dimensionality in the numerical approximation of PDEs. However, there are now a few mathematical results in the scientific literature (see, e.g., [17, 42, 61, 64, 65, 88, 93, 104, 129, 82]) which prove that deep neural networks have the capacity to approximate solutions of PDEs without the curse of dimensionality.

In particular, in the article Grohs et al. [64] it has been proved that there exist neural networks which approximate solutions of linear Black-Scholes PDEs with the number of parameters of the neural networks growing at most polynomially in both the reciprocal $1/\varepsilon$ of the prescribed approximation accuracy $\varepsilon \in (0, \infty)$ and the PDE dimension $d \in \mathbb{N}$. The

articles [17, 42, 61, 65, 93, 104, 129, 82], in particular, extend the results in the article Grohs et al. [64] to more general linear PDEs and the article Hutzenthaler et al. [88] extends the results in the article Grohs et al. [64] to nonlinear heat PDEs with Lipschitz continuous nonlinearities. To better explain the results in the article Hutzenthaler et al. [88], we now present in the following result, Theorem 4 below, a special case of Hutzenthaler et al. [88, Theorem 1.1].

Theorem 4. *Let* $\rho\colon (\bigcup_{d\in\mathbb{N}} \mathbb{R}^d) \to (\bigcup_{d\in\mathbb{N}} \mathbb{R}^d)$ *satisfy for all* $d \in \mathbb{N}$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ *that* $\rho(x) = (\max\{x_1, 0\}, \ldots, \max\{x_d, 0\})$, *let* $\mathbf{N} = \bigcup_{L\in\mathbb{N}} \bigcup_{l_0,l_1,\ldots,l_L\in\mathbb{N}} (\mathsf{X}_{k=1}^{L}(\mathbb{R}^{l_k \times l_{k-1}} \times \mathbb{R}^{l_k}))$, *let* $\mathcal{R}\colon \mathbf{N} \to (\bigcup_{k,l\in\mathbb{N}} C(\mathbb{R}^k, \mathbb{R}^l))$ *and* $\mathcal{P}\colon \mathbf{N} \to \mathbb{N}$ *satisfy for all* $L \in \mathbb{N}$, $l_0, l_1, \ldots, l_L \in \mathbb{N}$, $\Phi = ((W_1, B_1), (W_2, B_2), \ldots, (W_L, B_L)) \in (\mathsf{X}_{k=1}^{L}(\mathbb{R}^{l_k \times l_{k-1}} \times \mathbb{R}^{l_k}))$, $x_0 \in \mathbb{R}^{l_0}$, $x_1 \in \mathbb{R}^{l_1}$, ..., $x_L \in \mathbb{R}^{l_L}$ *with* $\forall\, k \in \{1, 2, \ldots, L-1\}\colon x_k = \rho(W_k x_{k-1} + B_k)$ *that* $\mathcal{R}(\Phi) \in C(\mathbb{R}^{l_0}, \mathbb{R}^{l_L})$, $(\mathcal{R}(\Phi))(x_0) = W_L x_{L-1} + B_L$, *and* $\mathcal{P}(\Phi) = \sum_{k=1}^{L} l_k(l_{k-1}+1)$, *let* $T, \kappa \in (0, \infty)$, $(\mathfrak{g}_{d,\varepsilon})_{(d,\varepsilon)\in\mathbb{N}\times(0,1]} \subseteq \mathbf{N}$, *let* $f\colon \mathbb{R} \to \mathbb{R}$ *be Lipschitz continuous, let* $u_d \in C^{1,2}([0,T] \times \mathbb{R}^d, \mathbb{R})$, $d \in \mathbb{N}$, *and assume for all* $d \in \mathbb{N}$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, $\varepsilon \in (0,1]$, $t \in [0,T]$ *that* $\mathcal{R}(\mathfrak{g}_{d,\varepsilon}) \in C(\mathbb{R}^d, \mathbb{R})$, $\varepsilon|u_d(t,x)| + |u_d(0,x) - (\mathcal{R}(\mathfrak{g}_{d,\varepsilon}))(x)| \le \varepsilon\kappa d^\kappa(1 + \sum_{i=1}^{d}|x_i|^\kappa)$, $\mathcal{P}(\mathfrak{g}_{d,\varepsilon}) \le \kappa d^\kappa \varepsilon^{-\kappa}$, *and*

$$(\tfrac{\partial}{\partial t}u_d)(t,x) = (\Delta_x u_d)(t,x) + f(u_d(t,x)). \tag{75}$$

Then there exist $(\mathfrak{u}_{d,\varepsilon})_{(d,\varepsilon)\in\mathbb{N}\times(0,1]} \subseteq \mathbf{N}$ *and* $c \in \mathbb{R}$ *such that for all* $d \in \mathbb{N}$, $\varepsilon \in (0,1]$ *it holds that* $\mathcal{R}(\mathfrak{u}_{d,\varepsilon}) \in C(\mathbb{R}^d, \mathbb{R})$, $\mathcal{P}(\mathfrak{u}_{d,\varepsilon}) \le cd^c\varepsilon^{-c}$, *and*

$$\left[\int_{[0,1]^d} |u_d(T,x) - (\mathcal{R}(\mathfrak{u}_{d,\varepsilon}))(x)|^2\,dx\right]^{1/2} \le \varepsilon. \tag{76}$$

Theorem 4 is an immediate consequence of Hutzenthaler et al. [88, Theorem 1.1]. In the following we add some comments on the mathematical objects appearing in Theorem 4 above and, thereby, we also add some explanatory comments on the statement of Theorem 4.

Theorem 4 is a DNN approximation result with the activation functions in the DNNs being multidimensional rectifier functions described by the function $\rho\colon (\bigcup_{d\in\mathbb{N}} \mathbb{R}^d) \to (\bigcup_{d\in\mathbb{N}} \mathbb{R}^d)$ in Theorem 4 above. The set $\mathbf{N}$ in Theorem 4 above represents *the set of all neural networks*. The function $\mathcal{R}\colon \mathbf{N} \to (\bigcup_{k,l\in\mathbb{N}} C(\mathbb{R}^k, \mathbb{R}^l))$ maps neural networks to their realization function in the sense that for every $\Phi \in \mathbf{N}$ it holds that $\mathcal{R}(\Phi) \in (\bigcup_{k,l\in\mathbb{N}} C(\mathbb{R}^k, \mathbb{R}^l))$ is the realization function associated with the neural network Φ. The function $\mathcal{P}\colon \mathbf{N} \to \mathbb{N}$ counts the number of parameters of the neural networks in the sense that for every $\Phi \in \mathbf{N}$ it holds that $\mathcal{P}(\Phi) \in \mathbb{N}$ represents the number of real numbers which are used to uniquely describe the neural network Φ.

Theorem 4 demonstrates that the solutions of the PDEs in (75) can be approximated by DNNs without the curse of dimensionality. The real number $T \in (0, \infty)$ in Theorem 4 describes the time horizon of the PDEs in (75) above. The function $f\colon \mathbb{R} \to \mathbb{R}$ in Theorem 4 describes the nonlinearity in the PDEs in (75). It is assumed to be Lipschitz continuous in the sense that there exists $L \in \mathbb{R}$ such that for all $x, y \in \mathbb{R}$ it holds that $|f(x) - f(y)| \le L|x - y|$.

The real number $\kappa \in (0, \infty)$ in Theorem 4 is used to formulate *the regularity and the approximation assumptions* which we impose in Theorem 4. In particular, we assume in

Theorem 4 that the initial value functions of the PDEs in (75) can be approximated by DNNs without the curse of dimensionality. In Theorem 4 this approximation assumption is formulated by means of the family $(\mathfrak{g}_{d,\varepsilon})_{(d,\varepsilon)\in\mathbb{N}\times(0,1]} \subseteq \mathbf{N}$ of neural networks. More formally, in Theorem 4 we assume that there exist neural networks $\mathfrak{g}_{d,\varepsilon} \in \mathbf{N}$, $d \in \mathbb{N}$, $\varepsilon \in (0,1]$, which approximate the initial value functions $\mathbb{R}^d \ni x \mapsto u_d(t,x) \in \mathbb{R}$, $d \in \mathbb{N}$, without the curse of dimensionality. In particular, we observe that the assumption in Theorem 4 that for all $d \in \mathbb{N}$, $x = (x_1,\ldots,x_d) \in \mathbb{R}^d$, $\varepsilon \in (0,1]$, $t \in [0,T]$ it holds that $\varepsilon|u_d(t,x)| + |u_d(0,x) - (\mathcal{R}(\mathfrak{g}_{d,\varepsilon}))(x)| \leq \varepsilon\kappa d^\kappa(1 + \sum_{i=1}^d |x_i|^\kappa)$ assures that for all $d \in \mathbb{N}$, $x = (x_1,\ldots,x_d) \in \mathbb{R}^d$, $\varepsilon \in (0,1]$ it holds that $|u_d(0,x) - (\mathcal{R}(\mathfrak{g}_{d,\varepsilon}))(x)| \leq \varepsilon\kappa d^\kappa(1 + \sum_{i=1}^d |x_i|^\kappa)$ and this condition, in turn, ensures that for all $d \in \mathbb{N}$, $x \in \mathbb{R}^d$ it holds that $(\mathcal{R}(\mathfrak{g}_{d,\varepsilon}))(x)$ converges to $u_d(0,x)$ as ε converges to 0.

Moreover, we observe that the assumption in Theorem 4 that for all $d \in \mathbb{N}$, $\varepsilon \in (0,1]$ it holds that $\mathcal{P}(\mathfrak{g}_{d,\varepsilon}) \leq \kappa d^\kappa \varepsilon^{-\kappa}$ assures that the number of parameters of the neural networks $\mathfrak{g}_{d,\varepsilon} \in \mathbf{N}$, $d \in \mathbb{N}$, $\varepsilon \in (0,1]$, grows at most polynomially in both the reciprocal ε^{-1} of the prescribed approximation precision $\varepsilon \in (0,1]$ and the PDE dimension $d \in \mathbb{N}$.

Furthermore, we note that the assumption in Theorem 4 that for all $d \in \mathbb{N}$, $x = (x_1,\ldots,x_d) \in \mathbb{R}^d$, $\varepsilon \in (0,1]$, $t \in [0,T]$ it holds that $\varepsilon|u_d(t,x)| + |u_d(0,x) - (\mathcal{R}(\mathfrak{g}_{d,\varepsilon}))(x)| \leq \varepsilon\kappa d^\kappa(1 + \sum_{i=1}^d |x_i|^\kappa)$ demonstrates that for all $d \in \mathbb{N}$, $x = (x_1,\ldots,x_d) \in \mathbb{R}^d$, $t \in [0,T]$ it holds that $|u_d(t,x)| \leq \kappa d^\kappa(1 + \sum_{i=1}^d |x_i|^\kappa)$ and this condition, in turn, ensures that the solutions of the PDEs in (75) grow at most polynomially in both the space variable $x \in \mathbb{R}^d$ and the PDE dimension $d \in \mathbb{N}$. The condition that for all $d \in \mathbb{N}$, $x = (x_1,\ldots,x_d) \in \mathbb{R}^d$, $t \in [0,T]$ it holds that $|u_d(t,x)| \leq \kappa d^\kappa(1 + \sum_{i=1}^d |x_i|^\kappa)$ also ensures that the solutions of the PDEs in (75) are uniquely described by their initial value functions $\mathbb{R}^d \ni x \mapsto u_d(0,x) \in \mathbb{R}$, $d \in \mathbb{N}$ (cf., e.g., Beck et al. [8, Theorem 1.1]).

Roughly speaking, *the conclusion* of Theorem 4 assures that there exist neural networks $\mathfrak{u}_{d,\varepsilon} \in \mathbf{N}$, $d \in \mathbb{N}$, $\varepsilon \in (0,1]$, such that for all $d \in \mathbb{N}$, $\varepsilon \in (0,1]$ it holds that the L^2-approximation error $[\int_{[0,1]^d} |u_d(T,x) - (\mathcal{R}(\mathfrak{u}_{d,\varepsilon}))(x)|^2 \, dx]^{1/2}$ between the exact solution $u_d(T,x)$ of the PDE and its neural network approximation $(\mathcal{R}(\mathfrak{u}_{d,\varepsilon}))(x)$ is smaller or equal than the prescribed approximation accuracy ε while the numbers $\mathcal{P}(\mathfrak{u}_{d,\varepsilon})$, $d \in \mathbb{N}$, $\varepsilon \in (0,1]$, of parameters of the approximating neural networks $\mathfrak{u}_{d,\varepsilon} \in \mathbf{N}$, $d \in \mathbb{N}$, $\varepsilon \in (0,1]$, grow at most polynomially in both the PDE dimension $d \in \mathbb{N}$ and the reciprocal ε^{-1} of the prescribed approximation accuracy ε. We note that Theorem 4 above is a neural network approximation result for the solutions of the PDEs in (75) at the final time T on the d-dimensional hypercube $[0,1]^d$ but the more general neural network approximation result in Hutzenthaler et al. [88, Theorem 4.1] also provides neural networks approximations for solutions of PDEs on more general space regions.

In the next step we add some words on *the strategy of the proofs* of Theorem 4 above and Theorem 1.1 in Hutzenthaler et al. [88], respectively. Even though Theorem 4 above and Theorem 1.1 in Hutzenthaler et al. [88], respectively, are purely deterministic neural network approximation results, the proofs of Theorem 4 above and Theorem 1.1 in Hutzenthaler et al. [88], respectively, are strongly based on probabilistic arguments on a suitable artificial probability space. In particular, the proofs of Theorem 4 above and Theorem 1.1 in Hutzenthaler et al. [88], respectively, employ the fact in the following elementary lemma.

Lemma 1. *Let $\varepsilon \in (0, \infty)$, let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, and let $E \colon \Omega \to \mathbb{R}$ be a random variable with $(\mathbb{E}[|E|^2])^{1/2} \leq \varepsilon$. Then there exists $\omega \in \Omega$ such that $|E(\omega)| \leq \varepsilon$.*

The elementary statement in Lemma 1 follows, e.g., from Grohs et al. [64, Proposition 3.3]. Lemma 1 is employed in the proofs of Theorem 4 above and Theorem 1.1 in Hutzenthaler et al. [88], respectively, to construct an appropriate random realization with desired approximation properties on a suitable artifical probability space. To make it more concrete, the proofs of Theorem 4 above and Theorem 1.1 in Hutzenthaler et al. [88], respectively, consist, roughly speaking, of the following four steps (cf., e.g., [93, Section 1]):

(I) First, appropriate random neural networks are constructed on a suitable artificial probability space. These appropriate neural networks are random in the sense that the weights and the biases of these neural networks are random variables instead of deterministic real numbers. The random neural networks are appropriately constructed with the aim to appropriately approximate the solutions of the PDEs in (75).

(II) Second, it is proved that the realization functions of these random neural networks are in a suitable root mean square sense close to the solutions of the PDEs in (75) at the final time T.

(III) Third, it is proved that the numbers of parameters of these random neural networks grow at most polynomially in both the reciprocal ε^{-1} of the prescribed approximation accuracy $\varepsilon \in (0, 1]$ and the PDE dimension $d \in \mathbb{N}$. Here the approximation accuracy is measured in a suitable root mean square sense according to item (II) above.

(IV) Fourth, Lemma 1 is applied to suitable error random variables, which describe certain L^2-errors between the realization functions of the constructed random neural networks (see item (II) above) and the exact solutions of the PDEs in (76) at the final time T (cf. (76) in Theorem 4 above), to obtain the existence of a realization on the artifical probability space such that the error random variables evaluated at this realization are smaller or equal than the prescribed approximation accuracy ε. Combining the existence of such a realization on the artifical probability space with item (III) above then completes the proofs of Theorem 4 above and Theorem 1.1 in Hutzenthaler et al. [88], respectively.

Let us also add a few comments on the way how the appropriate random neural networks in item (I) above are designed and on the way how the statements sketched in items (II)–(III) above are proved. The main tool for items (I)–(III) above are MLP approximations (cf. Section 6 above). More formally, the random neural networks in item (I) above are designed so that their realization functions coincide with suitable MLP approximations and the statement in item (II) above is then proved by employing suitable root mean square error estimates for MLP approximations (cf. Hutzenthaler et al. [89, Theorem 3.5] and Theorem 3 above) and the statement in item (III) above is then proved by employing suitable cost estimates for neural networks and MLP approximations (cf. Hutzenthaler et al. [88, Sections 3.2–3.3]).

8   Conclusion

The progress reviewed here has opened up a host of new possibilities, both in theory and applications. In applications, it has been proved effective in finance, such as the pricing of financial derivatives [138, 10, 12, 11] and credit valuation adjustment [54]. It also opens up new possibilities in control theory, an area that has long been hindered by the curse of dimensionality problem. In fact, it is likely that control theory will be among the areas most impacted by the kind of ideas reviewed here.

Another interesting new problem is the mathematical study of high dimensional PDEs. The fact that we can compute their solutions rather efficiently even in very high dimensions means that the complexity of these solutions should not be very high. Can we quantify this in some way? In low dimensions, a huge amount of effort has gone into studying the regularity of solutions of PDEs. It seems that regularity is not the most important issue in high dimensions. Rather, it is the complexity that is more relevant. It would be very interesting to develop a complexity-based PDE theory in high dimensions. It is worth mentioning that in low dimensions, regularity is also a measure of complexity: The efficiency of approximating a target function by certain approximation scheme, say piecewise polynomial approximation, is often measured by the regularity of the target function.

An interesting topic untouched in this review is reinforcement learning. Formulated with the language we use here, reinforcement learning is all about solving the Bellman equation for the underlying Markov decision process [135]. However, in contrast to the ideas reviewed here, which make heavy use of the underlying model, reinforcement learning makes minimum use of the specifics of the model. At this moment, it is still quite unclear what the relative merits are between the ideas reviewed here and those of reinforcement learning. This is undoubtedly an interesting area for further work.

Acknowledgement

References

[1] ARJOVSKY, M., AND BOTTOU, L. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv: 170104862* (2017).

[2] ARJOVSKY, M., CHINTALA, S., AND BOTTOU, L. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning* (International Convention Centre, Sydney, Australia, 06–11 Aug 2017), D. Precup and Y. W. Teh, Eds., vol. 70 of *Proceedings of Machine Learning Research*, PMLR, pp. 214–223.

[3] BALLY, V., PAGES, G., ET AL. A quantization algorithm for solving multidimensional discrete-time optimal stopping problems. *Bernoulli 9*, 6 (2003), 1003–1049.

[4] BECK, C., BECKER, S., CHERIDITO, P., JENTZEN, A., AND NEUFELD, A. Deep splitting method for parabolic PDEs. *arXiv:1907.03452* (2019).

[5] BECK, C., E, W., AND JENTZEN, A. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science 29*, 4 (2019), 1563–1619.

[6] BECK, C., GONON, L., AND JENTZEN, A. Overcoming the curse of dimensionality in the numerical approximation of high-dimensional semilinear elliptic partial differential equations. *arXiv preprint arXiv:2003.00596* (2020).

[7] BECK, C., HORNUNG, F., HUTZENTHALER, M., JENTZEN, A., AND KRUSE, T. Overcoming the curse of dimensionality in the numerical approximation of Allen-Cahn partial differential equations via truncated full-history recursive multilevel Picard approximations. *arXiv preprint arXiv:1907.06729* (2019).

[8] BECK, C., HUTZENTHALER, M., AND JENTZEN, A. On nonlinear Feynman-Kac formulas for viscosity solutions of semilinear parabolic partial differential equations. *arXiv preprint arXiv:2004.03389* (2020).

[9] BECKER, S., BRAUNWARTH, R., HUTZENTHALER, M., JENTZEN, A., AND VON WURSTEMBERGER, P. Numerical simulations for full history recursive multilevel Picard approximations for systems of high-dimensional partial differential equations. *arXiv preprint arXiv:2005.10206* (2020).

[10] BECKER, S., CHERIDITO, P., AND JENTZEN, A. Deep optimal stopping. *Journal of Machine Learning Research 20*, 74 (2019), 1–25.

[11] BECKER, S., CHERIDITO, P., AND JENTZEN, A. Pricing and hedging American-style options with deep learning. *Journal of Risk and Financial Management 13*, 7 (2020), 158.

[12] BECKER, S., CHERIDITO, P., JENTZEN, A., AND WELTI, T. Solving high-dimensional optimal stopping problems using deep learning. *Minor revision requested from European Journal of Applied Mathematics, arXiv:1908.01602* (2019).

[13] BELLMAN, R. E. *Dynamic Programming.* Princeton University Press, 1957.

[14] BENDER, C., AND DENK, R. A forward scheme for backward SDEs. *Stochastic Processes and their Applications 117*, 12 (2007), 1793–1812.

[15] BENDER, C., SCHWEIZER, N., AND ZHUO, J. A primal–dual algorithm for BSDEs. *Mathematical Finance 27*, 3 (2017), 866–901.

[16] Berg, J., and Nyström, K. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing 317* (2018), 28–41.

[17] Berner, J., Grohs, P., and Jentzen, A. Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *arXiv preprint arXiv:1809.03062* (2018).

[18] Billaud-Friess, M., Macherey, A., Nouy, A., and Prieur, C. Stochastic methods for solving high-dimensional partial differential equations. In *International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing* (2018), Springer, pp. 125–141.

[19] Bouchard, B., and Touzi, N. Discrete-time approximation and Monte-Carlo simulation of backward stochastic differential equations. *Stochastic Processes and their Applications 111*, 2 (2004), 175–206.

[20] Briand, P., Labart, C., et al. Simulation of BSDEs by Wiener chaos expansion. *The Annals of Applied Probability 24*, 3 (2014), 1129–1171.

[21] Cai, Z., and Liu, J. Approximating quantum many-body wave functions using artificial neural networks. *Physical Review B 97*, 3 (2018), 035116.

[22] Carleo, G., and Troyer, M. Solving the quantum many-body problem with artificial neural networks. *Science 355*, 6325 (2017), 602–606.

[23] Carmona, R., and Laurière, M. Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games: I–the ergodic case. *arXiv preprint arXiv:1907.05980* (2019).

[24] Carmona, R., and Laurière, M. Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games: II–the finite horizon case. *arXiv preprint arXiv:1908.01613* (2019).

[25] Chan-Wai-Nam, Q., Mikael, J., and Warin, X. Machine learning for semi linear PDEs. *Journal of Scientific Computing 79*, 3 (2019), 1667–1712.

[26] Chang, D., Liu, H., and Xiong, J. A branching particle system approximation for a class of FBSDEs. *Probability, Uncertainty and Quantitative Risk 1*, 1 (2016), 1–34.

[27] Chen, Y., Lu, L., Karniadakis, G. E., and Dal Negro, L. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics Express 28*, 8 (2020), 11618–11633.

[28] Crisan, D., and Manolarakis, K. Probabilistic methods for semilinear partial differential equations. Applications to finance. *ESAIM: Mathematical Modelling and Numerical Analysis 44*, 5 (2010), 1107–1133.

[29] CRISAN, D., AND MANOLARAKIS, K. Solving backward stochastic differential equations using the cubature method: application to nonlinear pricing. *SIAM Journal on Financial Mathematics 3*, 1 (2012), 534–571.

[30] CRISAN, D., MANOLARAKIS, K., ET AL. Second order discretization of backward SDEs and simulation with the cubature method. *The Annals of Applied Probability 24*, 2 (2014), 652–678.

[31] CRISAN, D., MANOLARAKIS, K., AND TOUZI, N. On the Monte Carlo simulation of BSDEs: An improvement on the Malliavin weights. *Stochastic Processes and their Applications 120*, 7 (2010), 1133–1158.

[32] DARBON, J., AND OSHER, S. Algorithms for overcoming the curse of dimensionality for certain Hamilton–Jacobi equations arising in control theory and elsewhere. *Research in the Mathematical Sciences 3*, 1 (2016), 19.

[33] DELARUE, F., MENOZZI, S., ET AL. A forward–backward stochastic algorithm for quasi-linear PDEs. *The Annals of Applied Probability 16*, 1 (2006), 140–184.

[34] DOCKHORN, T. A discussion on solving partial differential equations using neural networks. *arXiv preprint arXiv:1904.07200* (2019).

[35] DUFFIE, D., SCHRODER, M., AND SKIADAS, C. Recursive valuation of defaultable securities and the timing of resolution of uncertainty. *The Annals of Applied Probability 6*, 4 (1996), 1075–1090.

[36] E, W., HAN, J., AND JENTZEN, A. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics 5*, 4 (2017), 349–380.

[37] E, W., HUTZENTHALER, M., JENTZEN, A., AND KRUSE, T. Multilevel Picard iterations for solving smooth semilinear parabolic heat equations. *arXiv preprint arXiv:1607.03295* (2016).

[38] E, W., HUTZENTHALER, M., JENTZEN, A., AND KRUSE, T. On multilevel Picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. *Journal of Scientific Computing 79*, 3 (2019), 1534–1571.

[39] E, W., MA, C., AND WU, L. Barron spaces and the compositional function spaces for neural network models. *arXiv preprint arXiv:1906.08039* (2019).

[40] E, W., MA, C., AND WU, L. Machine learning from a continuous viewpoint. *arXiv preprint arXiv:1912.12777* (2019).

[41] E, W., AND YU, B. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics 6*, 1 (2018), 1–12.

[42] ELBRÄCHTER, D., GROHS, P., JENTZEN, A., AND SCHWAB, C. DNN expression rate analysis of high-dimensional PDEs: Application to option pricing. *arXiv preprint arXiv:1809.07669* (2018).

[43] EVANS, L. C. *Partial Differential Equations*, vol. 19. American Mathematical Soc., 2010.

[44] FAHIM, A., TOUZI, N., WARIN, X., ET AL. A probabilistic numerical method for fully nonlinear parabolic PDEs. *The Annals of Applied Probability 21*, 4 (2011), 1322–1364.

[45] FAN, Y., AND YING, L. Solving inverse wave scattering with deep learning. *arXiv preprint arXiv:1911.13202* (2019).

[46] FAN, Y., AND YING, L. Solving electrical impedance tomography with deep learning. *Journal of Computational Physics 404* (2020), 109119.

[47] FARAHMAND, A.-M., NABI, S., AND NIKOVSKI, D. N. Deep reinforcement learning for partial differential equation control. In *2017 American Control Conference (ACC)* (2017), IEEE, pp. 3120–3127.

[48] FISHER, R. A. The wave of advance of advantageous genes. *Annals of Eugenics 7*, 4 (1937), 355–369.

[49] FUJII, M., TAKAHASHI, A., AND TAKAHASHI, M. Asymptotic expansion as prior knowledge in deep learning method for high dimensional BSDEs. *Asia-Pacific Financial Markets 26*, 3 (2019), 391–408.

[50] GEISS, C., AND LABART, C. Simulation of BSDEs with jumps by Wiener chaos expansion. *Stochastic Processes and their Applications 126*, 7 (2016), 2123–2162.

[51] GILES, M. B. Multilevel Monte Carlo path simulation. *Operations Research 56*, 3 (2008), 607–617.

[52] GILES, M. B. Multilevel Monte Carlo methods. *Acta Numerica 24* (2015), 259.

[53] GILES, M. B., JENTZEN, A., AND WELTI, T. Generalised multilevel Picard approximations. *arXiv preprint arXiv:1911.03188* (2019).

[54] GNOATTO, A., PICARELLI, A., AND REISINGER, C. Deep xVA solver–a neural network based counterparty credit risk management framework. *arXiv preprint arXiv:2005.02633* (2020).

[55] GOBET, E., AND LABART, C. Solving BSDE with adaptive control variate. *SIAM Journal on Numerical Analysis 48*, 1 (2010), 257–277.

[56] GOBET, E., AND LEMOR, J.-P. Numerical simulation of BSDEs using empirical regression methods: theory and practice. *arXiv preprint arXiv:0806.4447* (2008).

[57] GOBET, E., LEMOR, J.-P., WARIN, X., ET AL. A regression-based Monte Carlo method to solve backward stochastic differential equations. *The Annals of Applied Probability 15*, 3 (2005), 2172–2202.

[58] GOBET, E., LÓPEZ-SALAS, J. G., TURKEDJIEV, P., AND VÁZQUEZ, C. Stratified regression Monte-Carlo scheme for semilinear PDEs and BSDEs with large scale parallelization on GPUs. *SIAM Journal on Scientific Computing 38*, 6 (2016), C652–C677.

[59] GOBET, E., AND TURKEDJIEV, P. Linear regression MDP scheme for discrete backward stochastic differential equations under general conditions. *Mathematics of Computation 85*, 299 (2016), 1359–1391.

[60] GOBET, E., TURKEDJIEV, P., ET AL. Approximation of backward stochastic differential equations using Malliavin weights and least-squares regression. *Bernoulli 22*, 1 (2016), 530–562.

[61] GONON, L., GROHS, P., JENTZEN, A., KOFLER, D., AND ŠIŠKA, D. Uniform error estimates for artificial neural network approximations for heat equations. *arXiv preprint arXiv:1911.09647* (2019).

[62] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems* (2014), pp. 2672–2680.

[63] GOUDENÈGE, L., MOLENT, A., AND ZANETTE, A. Variance reduction applied to machine learning for pricing Bermudan/American options in high dimension. *arXiv preprint arXiv:1903.11275* (2019).

[64] GROHS, P., HORNUNG, F., JENTZEN, A., AND VON WURSTEMBERGER, P. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *arXiv preprint arXiv:1809.02362* (2018).

[65] GROHS, P., JENTZEN, A., AND SALIMOVA, D. Deep neural network approximations for Monte Carlo algorithms. *arXiv preprint arXiv:1908.10828* (2019).

[66] GUO, W., ZHANG, J., AND ZHUO, J. A monotone scheme for high-dimensional fully nonlinear PDEs. *The Annals of Applied Probability 25*, 3 (2015), 1540–1580.

[67] HAN, J., AND E, W. Deep learning approximation for stochastic control problems. *Deep Reinforcement Learning Workshop, NIPS* (2016).

[68] HAN, J., AND HU, R. Deep fictitious play for finding Markovian Nash equilibrium in multi-agent games. In *Proceedings of The First Mathematical and Scientific Machine Learning Conference (MSML)* (2020), vol. 107, pp. 221–245.

[69] HAN, J., HU, R., AND LONG, J. Convergence of deep fictitious play for stochastic differential games. *arXiv preprint arXiv:2008.05519* (2020).

[70] HAN, J., JENTZEN, A., AND E, W. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences 115*, 34 (2018), 8505–8510.

[71] HAN, J., AND LONG, J. Convergence of the deep BSDE method for coupled FBSDEs. *Probability, Uncertainty and Quantitative Risk 5*, 1 (2020), 1–33.

[72] HAN, J., LU, J., AND ZHOU, M. Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion Monte Carlo like approach. *Journal of Computational Physics* (2020).

[73] HAN, J., ZHANG, L., AND E, W. Solving many-electron Schrödinger equation using deep neural networks. *Journal of Computational Physics 399* (2019), 108929.

[74] HEINRICH, S. Monte Carlo complexity of global solution of integral equations. *Journal of Complexity 14*, 2 (1998), 151–175.

[75] HEINRICH, S. Multilevel Monte Carlo methods. In *International Conference on Large-Scale Scientific Computing* (2001), Springer, pp. 58–67.

[76] HEINRICH, S., AND SINDAMBIWE, E. Monte Carlo complexity of parametric integration. *Journal of Complexity 15*, 3 (1999), 317–341.

[77] HENRY-LABORDÈRE, P. Counterparty risk valuation: a marked branching diffusion approach. *arXiv:1203.2369* (2012).

[78] HENRY-LABORDÈRE, P. Deep Primal-Dual Algorithm for BSDEs: Applications of Machine Learning to CVA and IM. *Available at SSRN 3071506* (2017).

[79] HENRY-LABORDERE, P., OUDJANE, N., TAN, X., TOUZI, N., WARIN, X., ET AL. Branching diffusion representation of semilinear PDEs and Monte Carlo approximation. In *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques* (2019), vol. 55, Institut Henri Poincaré, pp. 184–210.

[80] HENRY-LABORDÈRE, P., TAN, X., AND TOUZI, N. A numerical algorithm for a class of BSDEs via the branching process. *Stochastic Processes and their Applications 124*, 2 (2014), 1112–1140.

[81] HERMANN, J., SCHÄTZLE, Z., AND NOÉ, F. Deep neural network solution of the electronic Schrödinger equation. *arXiv preprint arXiv:1909.08423* (2019).

[82] HORNUNG, F., JENTZEN, A., AND SALIMOVA, D. Space-time deep neural network approximations for high-dimensional partial differential equations. *arXiv preprint arXiv:2006.02199* (2020).

[83] HUANG, M., CAINES, P. E., AND MALHAMÉ, R. P. Large-population cost-coupled LQG problems with nonuniform agents: individual-mass behavior and decentralized ϵ-Nash equilibria. *IEEE Transactions on Automatic Control 52*, 9 (2007), 1560–1571.

[84] HUANG, M., MALHAMÉ, R. P., AND CAINES, P. E. Large population stochastic dynamic games: closed-loop McKean-Vlasov systems and the Nash certainty equivalence principle. *Communications in Information and Systems 6*, 3 (2006), 221–252.

[85] HURÉ, C., PHAM, H., AND WARIN, X. Some machine learning schemes for high-dimensional nonlinear PDEs. *arXiv:1902.01599* (2019).

[86] HUTZENTHALER, M., JENTZEN, A., AND KRUSE, T. Overcoming the curse of dimensionality in the numerical approximation of parabolic partial differential equations with gradient-dependent nonlinearities. *arXiv preprint arXiv:1912.02571* (2019).

[87] HUTZENTHALER, M., JENTZEN, A., KRUSE, T., AND NGUYEN, T. A. Multilevel Picard approximations for high-dimensional semilinear second-order PDEs with Lipschitz nonlinearities. *arXiv preprint arXiv:2009.02484* (2020).

[88] HUTZENTHALER, M., JENTZEN, A., KRUSE, T., AND NGUYEN, T. A. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *SN Partial Differential Equations and Applications 1* (2020), 1–34.

[89] HUTZENTHALER, M., JENTZEN, A., KRUSE, T., NGUYEN, T. A., AND VON WURSTEMBERGER, P. Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations. *arXiv preprint arXiv:1807.01212, accepted in Proceedings of Royal Society A* (2020).

[90] HUTZENTHALER, M., JENTZEN, A., AND VON WURSTEMBERGER, P. Overcoming the curse of dimensionality in the approximative pricing of financial derivatives with default risks. *Electronic Journal of Probability* (2019).

[91] HUTZENTHALER, M., AND KRUSE, T. Multilevel Picard approximations of high-dimensional semilinear parabolic differential equations with gradient-dependent nonlinearities. *SIAM Journal on Numerical Analysis 58*, 2 (2020), 929–961.

[92] JACQUIER, A. J., AND OUMGARI, M. Deep PPDEs for rough local stochastic volatility. *arXiv preprint arXiv:1906.02551* (2019).

[93] JENTZEN, A., SALIMOVA, D., AND WELTI, T. A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. *arXiv preprint arXiv:1809.07321* (2018).

[94] JI, S., PENG, S., PENG, Y., AND ZHANG, X. Three algorithms for solving high-dimensional fully-coupled FBSDEs through deep learning. *IEEE Intelligent Systems* (2020).

[95] JIANG, D. R., AND POWELL, W. B. An approximate dynamic programming algorithm for monotone value functions. *Operations Research 63*, 6 (2015), 1489–1511.

[96] JIANYU, L., SIWEI, L., YINGJIAN, Q., AND YAPING, H. Numerical solution of elliptic partial differential equation using radial basis function neural networks. *Neural Networks 16*, 5-6 (2003), 729–734.

[97] KANG, W., GONG, Q., AND NAKAMURA-ZIMMERER, T. Algorithms of data development for deep learning and feedback design. *arXiv preprint arXiv:1912.00492* (2019).

[98] KARATZAS, I., AND SHREVE, S. E. *Brownian Motion and Stochastic Calculus.* Springer New York, 1998.

[99] KHOO, Y., LU, J., AND YING, L. Solving for high-dimensional committor functions using artificial neural networks. *Research in the Mathematical Sciences 6*, 1 (2019), 1.

[100] KHOO, Y., LU, J., AND YING, L. Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics* (2020), 115.

[101] KHOO, Y., AND YING, L. SwitchNet: a neural network model for forward and inverse scattering problems. *SIAM Journal on Scientific Computing 41*, 5 (2019), A3182–A3201.

[102] KINGMA, D., AND BA, J. Adam: a method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)* (2015).

[103] KOLMOGOROV, A., PETROVSKII, I., AND PISCUNOV, N. A study of the equation of diffusion with increase in the quantity of matter, and its application to a biological problem. *Moscow University Bulletin of Mathematics*, 1 (1937), 1–26.

[104] KUTYNIOK, G., PETERSEN, P., RASLAN, M., AND SCHNEIDER, R. A theoretical analysis of deep neural networks and parametric PDEs. *arXiv preprint arXiv:1904.00377* (2019).

[105] LABART, C., AND LELONG, J. A parallel algorithm for solving BSDEs. *Monte Carlo Methods and Applications 19*, 1 (2013), 11–39.

[106] LAGARIS, I. E., LIKAS, A., AND FOTIADIS, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks 9*, 5 (1998), 987–1000.

[107] LASRY, J.-M., AND LIONS, P.-L. Jeux champ moyen. I. Le cas stationnaire. *C. R. Math. Acad. Sci. Paris 9* (2006), 619–625.

[108] LASRY, J.-M., AND LIONS, P.-L. Jeux champ moyen. II. Horizon fini et contrle optimal. *C. R. Math. Acad. Sci. Paris 10* (2006), 679–684.

[109] Lasry, J.-M., and Lions, P.-L. Mean field games. *Japanese Journal of Mathematics 2* (2007), 229–260.

[110] Lee, H., and Kang, I. S. Neural algorithm for solving differential equations. *Journal of Computational Physics 91*, 1 (1990), 110–131.

[111] Lin, A. T., Fung, S. W., Li, W., Nurbekyan, L., and Osher, S. J. APAC-Net: Alternating the population and agent control via two neural networks to solve high-dimensional stochastic mean field games. *arXiv preprint arXiv:2002.10113* (2020).

[112] Luo, D., and Clark, B. K. Backflow transformations via neural networks for quantum many-body wave functions. *Physical Review Letters 122*, 22 (2019), 226401.

[113] Lye, K. O., Mishra, S., and Ray, D. Deep learning observables in computational fluid dynamics. *Journal of Computational Physics* (2020), 109339.

[114] Magill, M., Qureshi, F., and de Haan, H. Neural networks trained to solve differential equations learn general representations. In *Advances in Neural Information Processing Systems* (2018), pp. 4071–4081.

[115] McKean, H. P. Application of Brownian motion to the equation of Kolmogorov-Petrovskii-Piskunov. *Communications on Pure and Applied Mathematics 28*, 3 (1975), 323–331.

[116] Müller, J., and Zeinhofer, M. Deep Ritz revisited. *arXiv preprint arXiv:1912.03937* (2019).

[117] Nakamura-Zimmerer, T., Gong, Q., and Kang, W. Adaptive deep learning for high dimensional Hamilton-Jacobi-Bellman equations. *arXiv preprint arXiv:1907.05317* (2019).

[118] Nüsken, N., and Richter, L. Solving high-dimensional Hamilton-Jacobi-Bellman PDEs using neural networks: perspectives from the theory of controlled diffusions and measures on path space. *arXiv preprint arXiv:2005.05409* (2020).

[119] Oksendal, B. *Stochastic Differential Equations: An Introduction with Applications.* Springer Science & Business Media, 2013.

[120] Pardoux, É., and Peng, S. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In *Stochastic partial differential equations and their applications (Charlotte, NC, 1991)*, vol. 176 of *Lecture Notes in Control and Inform. Sci.* Springer, Berlin, 1992, pp. 200–217.

[121] Pardoux, É., and Tang, S. Forward-backward stochastic differential equations and quasilinear parabolic PDEs. *Probab. Theory Related Fields 114*, 2 (1999), 123–150.

[122] PFAU, D., SPENCER, J. S., MATTHEWS, A. G. D. G., AND FOULKES, W. Ab-initio solution of the many-electron Schrödinger equation with deep neural networks. *arXiv preprint arXiv:1909.02487* (2019).

[123] PHAM, H. Feynman-Kac representation of fully nonlinear PDEs and applications. *Acta Mathematica Vietnamica 40*, 2 (2015), 255–269.

[124] PHAM, H., PHAM, H., AND WARIN, X. Neural networks-based backward scheme for fully nonlinear PDEs. *arXiv preprint arXiv:1908.00412* (2019).

[125] RAISSI, M. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research 19*, 1 (2018), 932–955.

[126] RAISSI, M. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010* (2018).

[127] RAISSI, M., PERDIKARIS, P., AND KARNIADAKIS, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics 378* (2019), 686–707.

[128] RAO, A. V. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences 135*, 1 (2009), 497–528.

[129] REISINGER, C., AND ZHANG, Y. Rectified deep neural networks overcome the curse of dimensionality for nonsmooth value functions in zero-sum games of nonlinear stiff systems. *arXiv preprint arXiv:1903.06652* (2019).

[130] RUSZCZYNSKI, A., AND YAO, J. A dual method for evaluation of dynamic risk in diffusion processes. *arXiv preprint arXiv:1701.06234* (2017).

[131] RUTHOTTO, L., OSHER, S. J., LI, W., NURBEKYAN, L., AND FUNG, S. W. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proceedings of the National Academy of Sciences* (2020).

[132] SIRIGNANO, J., AND SPILIOPOULOS, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics 375* (2018), 1339–1364.

[133] SKOROKHOD, A. V. Branching diffusion processes. *Theory of Probability & Its Applications 9*, 3 (1964), 445–449.

[134] STRANG, G., AND FIX, G. J. *An Analysis of the Finite Element Method*. Prentice-Hall, 1973.

[135] SUTTON, R. S., AND BARTO, A. G. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[136] UCHIYAMA, T., AND SONEHARA, N. Solving inverse problems in nonlinear PDEs by recurrent neural networks. In *IEEE International Conference on Neural Networks* (1993), IEEE, pp. 99–102.

[137] VON PETERSDORFF, T., AND SCHWAB, C. Numerical solution of parabolic equations in high dimensions. *ESAIM: Mathematical Modelling and Numerical Analysis 38*, 1 (2004), 93–127.

[138] WANG, H., CHEN, H., SUDJIANTO, A., LIU, R., AND SHEN, Q. Deep learning-based BSDE solver for LIBOR market model with application to bermudan swaption pricing and hedging. *arXiv preprint arXiv:1807.06622* (2018).

[139] WARIN, X. Variations on branching methods for non linear PDEs. *arXiv preprint arXiv:1701.07660* (2017).

[140] WARIN, X. Monte Carlo for high-dimensional degenerated semi linear and full non linear PDEs. *arXiv preprint arXiv:1805.05078* (2018).

[141] WARIN, X. Nesting Monte Carlo for high-dimensional non-linear PDEs. *Monte Carlo Methods and Applications 24*, 4 (2018), 225–247.

[142] WATANABE, S. On the branching process for Brownian particles with an absorbing boundary. *Journal of Mathematics of Kyoto University 4*, 2 (1965), 385–398.

[143] ZANG, Y., BAO, G., YE, X., AND ZHOU, H. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics* (2020), 109409.

[144] ZHANG, D., GUO, L., AND KARNIADAKIS, G. E. Learning in modal space: Solving time-dependent stochastic PDEs using physics-informed neural networks. *SIAM Journal on Scientific Computing 42*, 2 (2020), A639–A665.

[145] ZHANG, D., LU, L., GUO, L., AND KARNIADAKIS, G. E. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics 397* (2019), 108850.

[146] ZHANG, J. A numerical scheme for BSDEs. *The Annals of Applied Probability 14*, 1 (2004), 459–488.

[147] ZHANG, Y., WANG, H., CHEN, W., ZENG, J., ZHANG, L., WANG, H., AND E, W. DP-GEN: A concurrent learning platform for the generation of reliable deep learning based potential energy models. *Computer Physics Communications* (2020), 107206.