Check for updates

Received: 2 February 2021 | Revised: 8 April 2021 | Accepted: 13 April 2021 | Published on: 28 May 2021

DOI: 10.1002/gamm.202100006

ORIGINAL PAPER

GAMM Mitteilungen

Three ways to solve partial differential equations with neural networks — A review

Jan Blechschmidt | **Oliver G. Ernst**

Department of Mathematics, TU Chemnitz, Saxony, Germany

Correspondence
Jan Blechschmidt, Reichenhainer Str. 41, 09126 Chemnitz, Germany.
Email: jan.blechschmidt@math.tu-chemnitz.de

Funding information
the German Federal Ministry Education and Research (BMBF), Grant/Award Number: 05M20OCA

Abstract

Neural networks are increasingly used to construct numerical solution methods for partial differential equations. In this expository review, we introduce and contrast three important recent approaches attractive in their simplicity and their suitability for high-dimensional problems: physics-informed neural networks, methods based on the Feynman–Kac formula and methods based on the solution of backward stochastic differential equations. The article is accompanied by a suite of expository software in the form of Jupyter notebooks in which each basic methodology is explained step by step, allowing for a quick assimilation and experimentation. An extensive bibliography summarizes the state of the art.

KEYWORDS

backward differential equation, curse of dimensionality, Feynman–Kac, Hamilton–Jacobi–Bellman equations, neural networks, partial differential equation, PINN, stochastic process

1 | INTRODUCTION

The spectacular successes of neural networks in machine learning tasks such as computer vision, natural speech processing and game theory as well as the prospect of harnessing the computing power of specialized hardware such as Google's Tensor Processing Units and Apple's Neural Engine designed to efficiently execute neural networks has led the scientific community to investigate their suitability also for high performance computing tasks. The result is now an exciting new research field known as *scientific machine learning*, where techniques such as deep neural networks and statistical learning are applied to classical problems of applied mathematics. In this expository survey our intention is to provide an accessible introduction to recent developments in the field of numerical solution of linear and nonlinear partial differential equations (PDEs) using techniques from machine learning and artificial intelligence.

After decades of research on the numerical solution of PDEs, manifold challenges remain. One that applies to essentially all classical discretization schemes is that they suffer from the *curse of dimensionality* first formulated by Bellman in the 1950s in the context of optimal control problems [10]. In its simplest manifestation (see [147] for a more extensive discussion) this notion states that doubling of the number of degrees of freedom in each of d coordinate directions increases the solution complexity (at least) by a factor of 2^d. In a similar spirit, the number of degrees of freedom when discretizing a 100-dimensional PDE with only 10 nodes in each coordinate direction exceeds the estimated number of atoms in the universe (around 10^{80}) by several orders of magnitude. One might think that equations in such high dimensions have little practical relevance, but they are common in mathematical finance and portfolio optimization where the spatial

This is an open access article under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.
© 2021 The Authors. *GAMM - Mitteilungen* published by Wiley-VCH GmbH.

dimension is determined by the number of financial assets in the market. Other areas prone to high-dimensional PDE problems include stochastic control, differential games and quantum physics. The challenge of solving high-dimensional PDEs has been taken up in a number of papers, and are addressed in particular in Section 3 for linear Kolmogorov PDEs and in Section 4 for semilinear PDEs in nondivergence form. Another impetus for the development of data-driven solution methods is the effort often necessary to develop tailored solution methods for different kinds of nonlinear PDEs. This will play a particular role in Section 2.

Neural networks offer attractive approximation capabilities for highly nonlinear functions. Their compositional nature contrasts with the more conventional additive form of trial functions in linear function spaces in which PDE solution approximations are constructed by Galerkin, collocation or finite volume methods. Their computational parametrization through statistical learning and large-scale optimization methods using modern hardware, software systems and algorithms are making them increasingly amenable for solving nonlinear and high-dimensional PDEs.

PDE solvers based on (deep) neural networks typically cannot compete with classical numerical solution methods in low to moderate dimensions—in particular as solving an algebraic equation is generally simpler than solving the highly nonlinear large-scale optimization problems associated with neural network training. Moreover, they currently lack the mature error analysis that has been established for traditional numerical methods. In addition, many specialized methods have been developed over the years for specific problems, often incorporating constraints or physical assumptions directly into the approximations. On the other hand, the ease with which methods such as the *physics-informed neural networks* to be discussed below can be applied to essentially any differential equation makes them attractive for rapid prototyping when efficiency and high accuracy are not the principal concern.

While we aim to provide a useful overview, research activity in this area is incredibly intense and impossible to cover exhaustively. Therefore, we have decided to present three approaches that have generated a lot of interest in recent years in detail in Sections 2–4. Further scientific machine learning methods for solving PDEs are collected in Section 5. Additionally, we want to draw some attention to another recent overview [9] which contains many references, in particular works focusing on the solution of PDEs in high-dimensions.

A unique feature of this paper is a collection of accompanying Jupyter notebooks that contain sample Python implementations of the methods reviewed in Sections 2 to 4 with detailed comments and explanations as well as a number of numerical experiments. The notebooks are freely available from the GitHub repository https://github.com/janblechschmidt/PDEsByNNs and can even be executed in *Google Collaboratory* directly in a web browser with no need for local installations. Of course, the reader may also download and run the notebooks on her local machine.

The remainder of the paper is organized as follows: Section 2 discusses physics-informed neural networks, a straightforward and flexible approach for leveraging machine learning technology on challenging nonlinear PDE problems. Section 3 and 4 are devoted to recent methods based on the long-established link between PDEs and stochastic processes, which for high dimensions makes approximations based on sampling attractive due to their dimension independence. Here neural networks on dedicated hardware can make the sample-based training very efficient. Section 5 provides an outlook to related developments in this area followed by a concluding Section 6.

2 | PHYSICS-INFORMED NEURAL NETWORKS

The flexibility of deep neural networks as a universal technique for function approximation comes at the price of a large number of parameters to be determined in the supervised learning phase, and therefore typically demands a large volume of training data. Physics-informed neural networks (PINNs) are a scientific machine learning technique for solving PDE problems in the *small data* setting, meaning only the PDE problem data is available rather than a large number of value pairs of the independent and dependent variables. PINNs generate approximate solutions to PDEs by training a neural network to minimize a loss function consisting of terms representing the misfit of the initial and boundary conditions along the boundary of the space-time domain as well as the PDE residual at selected points in the interior. While precursors of this approach date back to the early 1990s [100,101,103,149], the term PINN as well as a surge of ensuing research activity was initiated by the two-part report [156,157] subsequently published in [159].

We describe the PINN approach for approximating the solution $u : [0, T] \times \mathcal{D} \to \mathbb{R}$ of an evolution equation

$$\partial_t u(t, x) + \mathcal{N}[u](t, x) = 0, \qquad (t, x) \in (0, T] \times \mathcal{D}, \tag{1a}$$

$$u(0, x) = u_0(x), \qquad x \in \mathcal{D}, \tag{1b}$$

where $\mathcal{N}$ is a nonlinear differential operator acting on u, $\mathcal{D} \subset \mathbb{R}^d$ a bounded domain, T denotes the final time and $u_0 : \mathcal{D} \to \mathbb{R}$ the prescribed initial data. Although the methodology allows for different types of boundary conditions, we restrict our discussion to the inhomogeneous Dirichlet case and prescribe

$$u(t,x) = u_b(t,x), \quad (t,x) \in (0,T] \times \partial\mathcal{D}, \tag{1c}$$

where $\partial\mathcal{D}$ denotes the boundary of the domain $\mathcal{D}$ and $u_b : (0,T] \times \partial\mathcal{D} \to \mathbb{R}$ the given boundary data. The method constructs a neural network approximation $u_\theta(t,x) \approx u(t,x)$ of the solution of (1), where $u_\theta : [0,T] \times \mathcal{D} \to \mathbb{R}$ denotes a function realized by a neural network with parameters θ.

In contrast to other learning-based methods that try to infer the solution by a purely data-driven approach, that is, by fitting a neural network to a number of state-value pairs $\{(t_i, x_i, u(t_i, x_i))\}_{i=1}^N$, PINNs take the underlying PDE (the "physics") into account. Taking advantage of modern machine learning software environments, which provide automatic differentiation capabilities for functions realized by neural networks, the approximate solution u_θ is differentiated with respect to the time and space variables, which allows the residual of the nonlinear PDE (1a) to be evaluated at a set of collocation points. In this way, the physics encoded in the differential equation is made available for a loss function measuring the extent to which the PDE problem (1) is satisfied by u_θ.

While the focus of other methods employing neural networks for solving PDEs is on mitigating the curse of dimensionality in high dimensions, the strength of PINNs lies in their flexibility in that they can be applied to a great variety of challenging PDEs, whereas classical numerical approximations typically require tailoring to the specifics of a particular PDE. In particular, this includes problems from computational physics that are notoriously hard to solve with classical numerical approaches due to, for example, strong nonlinearities, convection dominance or shocks, see also the last paragraph in Section 2.4. A further challenge that can be addressed by this approach is the regime with a small number of data samples, which is common for physical experiments since the acquisition of new data samples is often expensive.

In [156] the authors introduce the PINN methodology for solving nonlinear PDEs and demonstrate its efficiency for the Schrödinger, Burgers and Allen–Cahn equations. The focus of the second part [157] lies in the simultaneous solution of a nonlinear PDE of the form (1a) and the identification of corresponding unknown parameters λ which enter the nonlinear part of the differential equation. This problem setting has been studied within the regime of Gaussian processes in [153,154,168]. For both problem settings, the authors discuss, depending on the type of data available, a time-continuous and time-discrete approach. We discuss these methods next.

2.1 | Continuous time approach

The continuous time approach for the parabolic PDE (1) as described in [156] is based on the (strong) residual of a given neural network approximation $u_\theta : [0,T] \times \mathcal{D} \to \mathbb{R}$ of the solution u with respect to (1a)

$$r_\theta(t,x) := \partial_t u_\theta(t,x) + \mathcal{N}[u_\theta](t,x). \tag{2}$$

The neural network class considered here are *multilayer feed-forward neural networks*, sometimes known as *multilayer perceptrons*. Such networks are compositions of alternating affine linear $W^\ell \cdot + b^\ell$ and nonlinear functions $\sigma^\ell(\cdot)$ called activations, that is,

$$u_\theta(z) := W^L \sigma^L(W^{L-1}\sigma^{L-1}(\cdots \sigma^1(W^0 z + b^0) \cdots) + b^{L-1}) + b^L,$$

where W^ℓ and b^ℓ are weight matrices and bias vectors, and $z = [t,x]^T$. This highly nonlinear compositional structure of the approximating function u_θ forms the core of many neural network-based machine learning methods, and has been found to possess remarkably good approximation properties in many applications.

In general, training a neural network, that is, determining the (typically large number of) parameters θ, using gradient-based optimization methods [23,58,63,167] such as stochastic gradient descent [23], the Adam optimizer [95], or AdaGrad [42], requires the derivative of u_θ with respect to its unknown parameters W^ℓ and b^ℓ. To incorporate the PDE residual (2) into the loss function to be minimized, PINNs require a further differentiation to evaluate the differential operators $\partial_t u_\theta$ and $\mathcal{N}[u_\theta]$. Thus the PINN term r_θ shares the same parameters as the original network $u_\theta(t,x)$, but

respects the "physics" of (1a). Both types of derivatives can be easily obtained by automatic differentiation [4] with current state-of-the-art machine learning libraries, for example, TensorFlow [1] or PyTorch [141]. In Example 1, we show how such a PINN can be derived explicitly for the one-dimensional time-dependent eikonal equation.

The PINN approach for the solution of the PDE (1) now proceeds by minimization of the loss functional

$$\phi_\theta(X) := \phi_\theta^r(X^r) + \phi_\theta^0(X^0) + \phi_\theta^b(X^b), \tag{3}$$

where X denotes the collection of training data and the loss function ϕ_θ contains the following terms:

- the mean squared residual

$$\phi_\theta^r(X^r) := \frac{1}{N_r}\sum_{i=1}^{N_r}\left|r_\theta\left(t_i^r, x_i^r\right)\right|^2$$

  in a number of collocation points $X^r := \{(t_i^r, x_i^r)\}_{i=1}^{N_r} \subset (0,T] \times \mathcal{D}$, where r_θ is the physics-informed neural network (2),
- the mean squared misfit with respect to the initial and boundary conditions

$$\phi_\theta^0(X^0) := \frac{1}{N_0}\sum_{i=1}^{N_0}\left|u_\theta\left(t_i^0, x_i^0\right) - u_0\left(x_i^0\right)\right|^2 \quad \text{and} \quad \phi_\theta^b(X^b) := \frac{1}{N_b}\sum_{i=1}^{N_b}\left|u_\theta\left(t_i^b, x_i^b\right) - u_b\left(t_i^b, x_i^b\right)\right|^2$$

  in a number of points $X^0 := \{(t_i^0, x_i^0)\}_{i=1}^{N_0} \subset \{0\} \times \mathcal{D}$ and $X^b := \{(t_i^b, x_i^b)\}_{i=1}^{N_b} \subset (0,T] \times \partial\mathcal{D}$, where u_θ is the neural network approximation of the solution $u : [0,T] \times \mathcal{D} \to \mathbb{R}$.

We note that the training data X consists entirely of time-space coordinates. Moreover, individual weighting of each loss term in (3) may help improve the convergence of the scheme, see for example [163].

2.1.1 | Example: Burgers equation

To illustrate the PINN approach we consider the one-dimensional Burgers equation on the spatial domain $\mathcal{D} = [-1,1]$

$$\begin{aligned} \partial_t u + u\,\partial_x u - (0.01/\pi)\,\partial_{xx} u &= 0, && (t,x) \in (0,1] \times (-1,1), \\ u(0,x) &= -\sin(\pi\, x), && x \in [-1,1], \\ u(t,-1) = u(t,1) &= 0, && t \in (0,1]. \end{aligned} \tag{4}$$

This PDE arises in various disciplines such as traffic flow, fluid mechanics and gas dynamics, and can be derived from the Navier–Stokes equations, see [3]. We assume that the collocation points X^r as well as the points for the initial and boundary data X^0 and X^b are generated by random sampling from a uniform distribution. Although uniformly distributed data are sufficient in our experiments, the authors of [156] employed a space-filling Latin hypercube sampling strategy [174]. Our numerical experiments indicate that this strategy slightly improves the observed convergence rate, but for simplicity the code examples accompanying this paper employ uniform sampling throughout.

We choose training data of size $N_0 = N_b = 50$ and $N_r = 10\,000$. In this example, adopted from [156], we assume a deep neural network of the following structure: the input is scaled elementwise to lie in the interval $[-1,1]$, followed by 8 fully connected layers each containing 20 neurons and each followed by a hyperbolic tangent activation function and one output layer. This setting results in a network containing 3021 trainable parameters (first hidden layer: $2 \times 20 + 20 = 60$; seven intermediate layers: each $20 \times 20 + 20 = 420$; output layer: $20 \times 1 + 1 = 21$).

The loss functional (3) can be minimized by a number of algorithms, our accompanying code implements gradient descent-based algorithms as well as a variant of the limited-memory Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [109] which was also used in the numerical experiments in [156]. Although currently the majority of neural networks are trained with gradient descent-based methods, BFGS is a quasi-Newton algorithm also often employed for scientific machine learning tasks.

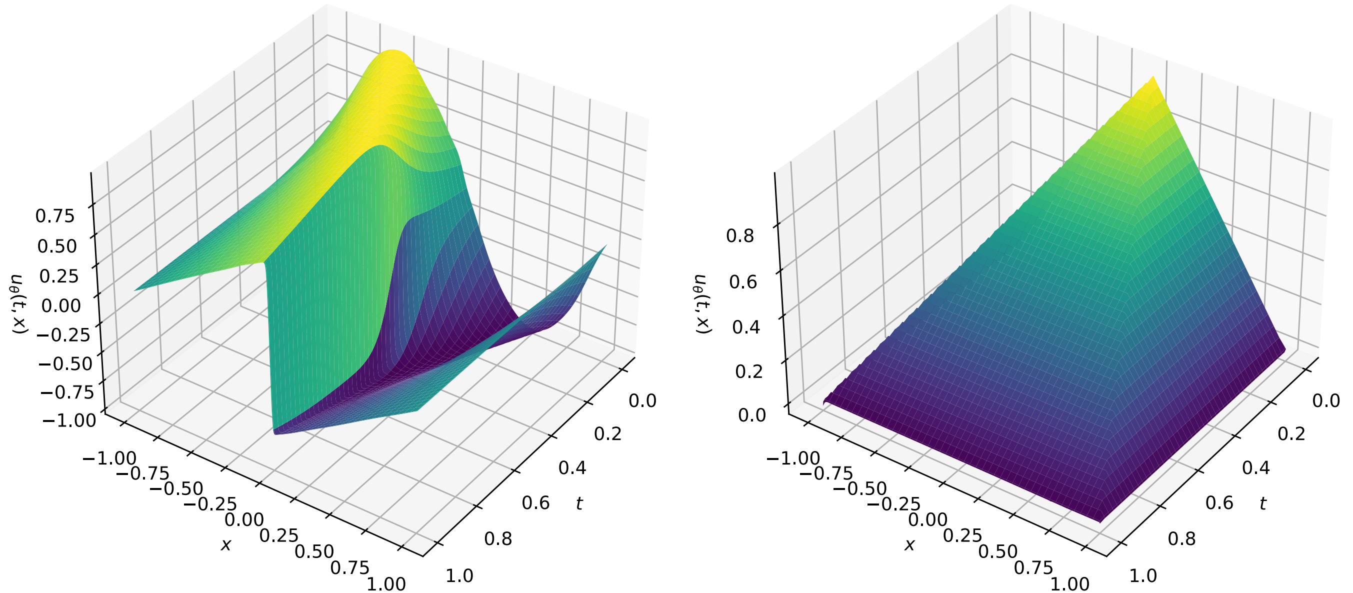

FIGURE 1 Left: PINN approximation u_θ of the solution of Burgers equation (4). The shock formation at around $t = 0.4$ is clearly visible. Right: Approximate solution u_θ of the eikonal equation (5) with sharp edges at $t = |x|$. Both examples are implemented in the accompanying Jupyter Notebook `PINN_Solver.ipynb`

The left panel of Figure 1 shows the approximate solution of the Burgers equation (4) after 5000 training epochs with the Adam optimizer and learning rate[1] $\delta(n) = 0.01\ \mathbf{1}_{\{n<1000\}} + 0.001\ \mathbf{1}_{\{1000\le n<3000\}} + 0.00\ \mathbf{1}_{\{3000\le n\}}$ which decays in a piecewise constant fashion.

2.1.2 | Example: Eikonal equation

As a second example we consider the one-dimensional eikonal equation backward in time on the domain $\mathcal{D} = [-1, 1]$

$$\begin{aligned} -\partial_t u(t,x) + |\nabla u|(t,x) &= 1, && (t,x) \in [0,T) \times [-1,1], \\ u(T,x) &= 0, && x \in [-1,1], \\ u(t,-1) = u(t,1) &= 0, && t \in [0,T). \end{aligned} \tag{5}$$

Note that the partial differential equation in (5) can be equally written as a Hamilton–Jacobi–Bellman equation, viz

$$-\partial_t u(t,x) + \sup_{|c|\le 1}\{c\ \nabla u(t,x)\} = 1, \qquad (t,x) \in [0,T) \times [-1,1],$$

which characterizes the solution of an optimal control problem seeking to minimize the distance from a point (t,x) to the boundary $[0,T] \times \partial\mathcal{D} \cup \{T\} \times \mathcal{D}$. As is easily verified, the solution is given by $u(t,x) = \min\{1-t, 1-|x|\}$. The fact that (5) runs backward in time is in accordance with its interpretation as the optimality condition of a control problem. Note that (5) is transformed into a forward evolution problem (1a) by the change of variables $\hat{t} = T - t$.

The neural network model chosen for this particular problem can be simpler. We decided to use only two hidden layers with 20 neurons in each, resulting in 501 unknown parameters (first hidden layer: $2 \times 20 + 20 = 60$; one intermediate layer: $20 \times 20 + 20 = 420$; output layer: $20 \times 1 + 1 = 21$). To account for the lack of smoothness of the solution, we choose a nondifferentiable activation function, although the hyperbolic tangent function seems to be able to approximate the kinks in the solution sufficiently well. Here, we decided to use the *leaky rectified linear unit (leaky ReLU)* activation

[1] The chosen learning rates used in the Adam optimizer in this section are not based on any hyperparameter optimization but were selected in a way that ensured stable and reliable results.

TABLE 1 Number of successful attempts to learn the solution of the eikonal equation (5) for different network architectures for 10 randomly initialized sets of training data with $N_r = 2000$, $N_0 = 25$ and $N_b = 50$

	One hidden layer			Two hidden layers		
Activation	**3 neurons**	**10 neurons**	**25 neurons**	**3 neurons**	**10 neurons**	**25 neurons**
ReLU	0	3	8	0	5	3
Leaky ReLU	0	9	10	1	7	10

Note: An attempt is considered successful if it achieves a training loss below the threshold $\phi_\theta(X) < 10^{-10}$.

function [114]

$$\sigma(z) = \begin{cases} z & \text{if } z \geq 0, \\ 0.1\, z & \text{otherwise,} \end{cases}$$

which displays a nonvanishing gradient when the unit is not active, that is, when $z < 0$. The approximate solution after $N_{\text{epochs}} = 10\,000$ epochs of training with the Adam optimizer [95] and a piecewise constant learning rate

$$\delta(n) = 0.1\mathbf{1}_{\{n<3000\}} + 0.01\ \mathbf{1}_{\{3000\leq n<7000\}} + 0.001\ \mathbf{1}_{\{7000\leq n\}} \tag{6}$$

is displayed in the right panel of Figure 1. Noting that the explicit solution of the eikonal equation is a piecewise linear function on a convex polyhedral domain, closer inspection yields the closed-form expression

$$u(t,x) = \texttt{ReLU}(x+1) - \texttt{ReLU}(x+t) - \texttt{ReLU}(x-t),$$

which can be represented exactly by a neural network with one hidden layer containing three neurons. In order to study the capability of the PINN approach combined with the Adam optimizer to recover the solution of this problem we conducted an experiment for which we counted the number of successful attempts to train the model to achieve a training loss below the threshold $\phi_\theta(X) < 10^{-10}$. Otherwise, when a maximum number of iterations of 100 000 was reached, the algorithm had most often converged to a local minimum and no further decrease of the loss could be expected. We compared the activation functions leaky `ReLU` (slope 0.1 for negative values) and standard `ReLU` (zero slope for negative values) on a set of different network architectures for 10 uniformly drawn sets of training data with $N_r = 2000$, $N_0 = 25$ and $N_b = 50$ with learning rate as given in (6). Table 1 shows the absolute number of successes among 10 independent runs, indicating clearly that the leaky `ReLU` outperforms standard `ReLU` in this case.

We conclude this section with the explicit derivation of a PINN for a neural network with a single hidden layer.

Example 1. For the one-dimensional eikonal equation (5) the PDE residual is obtained as

$$r(t,x) := -\partial_t u(t,x) + |\nabla u|(t,x) - 1.$$

For simplicity we consider a single hidden layer neural network with only three neurons, resulting in the solution approximation

$$u_\theta(t,x) = U\sigma\left(W\begin{bmatrix} t \\ x \end{bmatrix} + b\right) + c$$

with unknown weight matrices $U \in \mathbb{R}^1$, $W \in \mathbb{R}^{3\times 2}$ and bias vectors $b \in \mathbb{R}^3$, $c \in \mathbb{R}^1$, and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ acting componentwise on its input. We further abbreviate the values of the hidden layer by $z = W[t,x]^T + b$. The chain rule now yields the partial derivatives

$$\partial_t u_\theta(t,x) = U\ \text{diag}(\sigma'(z))\ W_{:,1} \qquad \text{and} \qquad \partial_x u_\theta(t,x) = U\ \text{diag}(\sigma'(z))\ W_{:,2}$$

where $\text{diag}(\sigma'(z))$ denotes the matrix with diagonal entries $\sigma'(z)$ and $W_{:,j}$ denotes the jth column of the matrix W. This allows us to compute the residual (the actual physics-informed neural network):

$$r_\theta(t,x) = -U\ \text{diag}(\sigma'(z))\ W_{:,1} + \left|U\ \text{diag}(\sigma'(z))\ W_{:,2}\right| - 1.$$

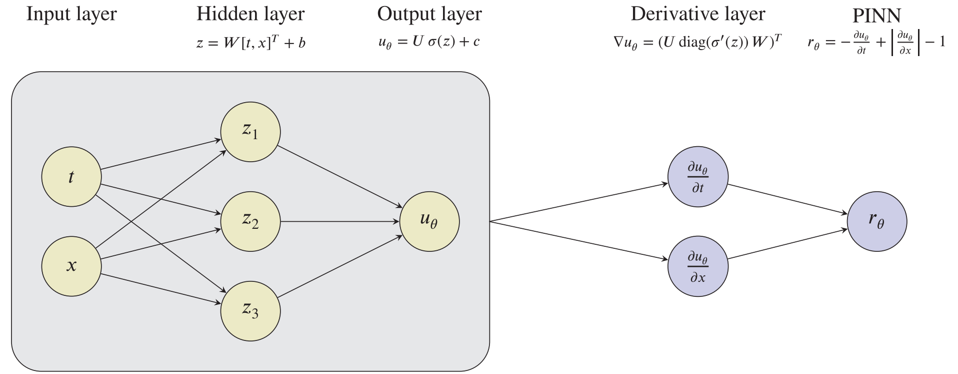

FIGURE 2 Illustration of a neural network with a single hidden layer (yellow). Complete network includes the physics-informed neural network r_θ for the one-dimensional eikonal equation (5) derived from the spatial and temporal derivatives of u_θ

We observe that the residual again possesses the structure of a more complicated neural network mapping $(t,x) \mapsto r(t,x)$. The neural network employed in this example is illustrated in Figure 2.

2.2 | Discrete time approach

In contrast to the continuous time approach, the discrete time variant does not incorporate physical information through a set of collocation points, but does so by semi-discretization via Runge–Kutta time-stepping [67]. Specifically, assuming the solution is known at time $t^n \in [0,T)$, this method assumes the availability of N_n solution data points $X^n := \{(t^n, x^{n,k}, u^{n,k})\}_{k=1}^{N_n}$ together with boundary data at the domain boundaries. To continue the solution to t^{n+1}, we employ a Runge–Kutta method with q stages

$$
\begin{aligned}
u^{n+c_i} &= u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], \qquad\qquad i = 1, \dots, q, \\
u^{n+1} &= u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}],
\end{aligned}
\tag{7}
$$

where $u^{n+c_j} \approx u(t^n + c_j \Delta t, \cdot)$ for $j = 1, \dots, q$. Depending on the coefficients a_{ij}, b_j, c_j, this represents either an explicit or implicit Runge–Kutta scheme.

While the neural network in the continuous approach approximates the mapping $(t,x) \mapsto u(t,x)$, the discrete-time variant instead approximates $x \mapsto (u^{n+c_1}(x), \dots, u^{n+c_q}(x), u^{n+1}(x))$, that is, the solution $u(t,x)$ at the $q+1$ stage values. Once sufficiently trained, $u(t^{n+1},x) \approx u^{n+1}(x)$ can be used as the initial data for the next step. Thus, subsequent steps can proceed analogously.

To be more precise, we establish the link between our data set $\{(t^n, x^{n,k}, u^{n,k})\}_{k=1}^{N_n}$, the PDE solution at time t^{n+1} and the unknown stages u^{n+c_i}, $i = 1, \dots, q$ of the Runge–Kutta scheme (7), which should hold for all $x \in \mathcal{D}$, and in particular for all data samples $(x^{n,k}, u^{n,k})$. This results after a rearrangement of the terms in

$$
\begin{aligned}
r^i(x^{n,k}, u^{n,k}) &:= u^{n+c_i}(x^{n,k}) - u^{n,k} + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}](x^{n,k}) \approx 0, \qquad\qquad i = 1, \dots, q, \\
r^{q+1}(x^{n,k}, u^{n,k}) &:= u^{n+1}(x^{n,k}) - u^{n,k} + \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}](x^{n,k}) \approx 0.
\end{aligned}
$$

These identities are then used to learn the unknown mapping $x \mapsto (u^{n+c_1}(x), \dots, u^{n+c_q}(x), u^{n+1}(x))$ by minimizing the loss functional, specified here with homogeneous Dirichlet boundary data

$$
\phi(X^n) := \sum_{k=1}^{N_n} \sum_{j=1}^{q+1} |r^j(x^{n,k}, u^{n,k})|^2 + \sum_{i=1}^{q} \left(|u^{n+c_i}(-1)|^2 + |u^{n+c_i}(+1)|^2 \right) + |u^{n+1}(-1)|^2 + |u^{n+1}(-1)|^2
$$

The numerical experiments presented in [156] employ a 500-stage Runge–Kutta scheme that advances from initial to final time in a single time step. The option of using Runge–Kutta methods of extremely high-order rather than small time steps is presented as an attractive feature of this approach, as the task of stage computation for stiff problems requiring implicit integration schemes are passed on to the neural network optimization. Together with the simplicity of the algorithm and the possibility of choosing large time steps of high order, the numerical results in [156] suggest that the method is capable of handling a variety of nonlinearities and boundary conditions.

2.3 | Parameter identification setting

The PINN approach is easily modified to also determine unknown parameters in a general nonlinear partial differential equation. As an example, consider the PDE

$$\partial_t u(t,x) + \mathcal{N}^{\lambda}[u](t,x) = 0, \qquad (t,x) \in (0,T] \times \mathcal{D}, \tag{8}$$

with $\mathcal{N}^{\lambda}$ a nonlinear partial differential operator depending on a parameter $\lambda \in \mathbb{R}^m$. Here, we consider only the continuous time framework introduced in Section 2.1, and refer to [157] for the discrete time variant.

The parameter identification setting as introduced in [157] assumes a set of data $X_d := \{t_i^d, x_i^d, u_i^d\}_{i=1}^{N_d}$, where $u_i^d \approx u(t_i^d, x_i^d)$ are (possibly noisy) observations of the solution of problem (8) in order to identify the unknown parameter λ. This training data is then used twofold in a new loss function: in a mean squared misfit term and also in a mean squared residual term:

$$\phi(X_d) = \frac{1}{N_d} \sum_{i=1}^{N_d} |u_\theta(t_i^d, x_i^d) - u_i^d|^2 + \frac{1}{N_d} \sum_{i=1}^{N_d} |r_\theta(t_i^d, x_i^d)|^2.$$

Here, we consider a slightly modified procedure: In addition to the initial values, boundary and collocation data X introduced in Section 2.1, we treat the (possibly noisy) observations X_d of the solution of problem (8) in the same way as Dirichlet boundary conditions, which can be enforced via an additional loss function term

$$\phi^d(X_d) := \frac{1}{N_d} \sum_{i=1}^{N_d} \left| u_\theta \left(t_i^d, x_i^d\right) - u_i^d \right|^2,$$

added to the loss functional (3).

The unknown parameter λ can be learned through training in the same way as the unknown weight matrices W^ℓ and bias vectors b^ℓ by automatic differentiation of the loss function ϕ with respect to λ. Indeed, the modifications necessary for including the dependence of the PDE on an unknown parameter require merely a few lines of code, as can be seen in the accompanying Jupyter notebook `PINN_Solver.ipynb`.

In our example we consider the parametric eikonal equation

$$-\partial_t u(t,x) + |\nabla u|(t,x) = \lambda^{-1} \tag{9}$$

with homogeneous final time and boundary conditions and unknown parameter $\lambda > 0$. Its explicit solution is given by $u^*(t,x) = \lambda^{-1} \min\{1-t, 1-|x|\}$. The numerical results for $\lambda^* = 3$ after 10 000 training epochs with the Adam optimizer, a piecewise constant learning rate (6) for a neural network consisting of one hidden layer with 20 neurons and leaky ReLU activation function are shown in Figure 3.

2.4 | Summary and extensions

Physics-informed neural networks can be used to solve nonlinear partial differential equations. While the continuous-time approach approximates the PDE solution on a time-space cylinder, the discrete time approach exploits the parabolic structure of the problem to semi-discretize the problem in time in order to evaluate a Runge–Kutta method. A major advantage of this approach is that it is data-efficient in the sense that it does not require a large number of training

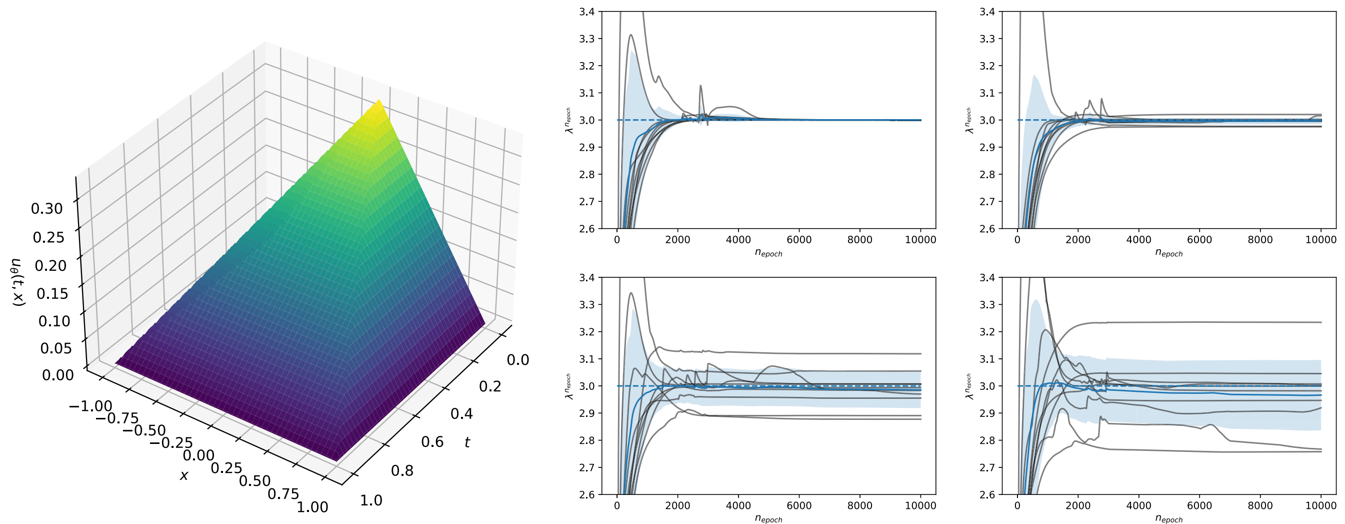

FIGURE 3 Left: One approximate solution u_θ of the parametric eikonal equation (9). Right: Ten evolutions of the estimated parameters $\lambda^{n_{\text{epochs}}}$ for $n_{\text{epochs}} = 1, \dots, 10\,000$ (gray) with $N_d = 500$ noisy measurements $u_i^d = u(t_i^d, x_i^d) + \varepsilon\eta$ with $\eta \sim \mathrm{N}(0, 1)$ for different noise levels $\varepsilon = 0.0, 0.01, 0.05, 0.1$ (from upper left to lower right) together with its mean (solid blue) and one standard deviation around the mean (shaded area). The different paths are a result of the random initialization of the parameters in the neural network as well as randomly drawn data X and X_d

samples, which may be difficult to obtain in physical experiments. Indeed, besides the information on the initial time and spatial boundary, no further knowledge of solution values is required.

In contrast to the method described in Section 4, the PINN approach is based on a single neural network to characterize the solution on the entire time-space cylinder $[0, T] \times \overline{D}$. We note that the focus of the approach does not lie in the solution of high-dimensional problems but rather in challenging physics features including shocks, convection dominance, and so on. Another advantage of this approach is that the value of the loss function can be interpreted as a measure of accuracy of the approximation, and thus can be used as a stopping criterion during training. We further recall that all derivatives required in the derivation of PINNs (2) can be computed by the chain rule and evaluated by means of automatic differentiation [4].

A similar physics-constrained approach based on convolutional encoder-decoder neural networks for solving PDEs with random data is developed in [183]. Parametrized and locally adaptive activation functions to improve the learning rate in connection with PINNs are explored in [89] and [90], resp. Rigorous estimates on the generalization error of PINNs in the context of inverse problems and data assimilation are given in [126]. XPINNS (eXtended PINNS) are introduced in [88] as a generalization of PINNS involving multiple neural networks allowing for parallelization in space and time via domain decomposition, see also [70] for a recent review on machine learning approaches in domain decomposition. The converse task of learning a nonlinear differential equation from given observations using neural networks is addressed in [151].

In addition, PINNs have been applied successfully in a wide range of applications, including fluid dynamics [113,115,117,160,177], continuum mechanics and elastodynamics [66,132,162], inverse problems [91,121], fractional advection–diffusion equations [135], stochastic advection–diffusion–reaction equations [34], stochastic differential equations [179] and power systems [127]. Finally, we mention that Gaussian processes as an alternative to neural networks for approximating complex multivariate functions have also been studied extensively for solving PDEs and inverse problems [136,155,158,164]. While PINNs have been found to work essentially out of the box in many of these references, as was the case for the examples in Section 2.1.1, they may require problem-specific adaptations, particularly when accuracy of efficiency is a consideration. An example is a clustering of the interior collocation points to improve the resolution near a shock when solving the Euler equations in [117].

3 | LINEAR PDES IN HIGH DIMENSIONS: THE FEYNMAN–KAC FORMULA

The appeal of the PINN approach of the previous section lies in its simplicity as well as its versatility in applying to a large range of PDE problems. The neural network-based approaches presented in this and the next section are aimed at solving PDE problems posed on high-dimensional domains, one of the unsolved problems of numerical analysis. These problems stem from important applications such as derivative valuation in financial portfolios, the Schrödinger equation in the quantum many-body problem or the Hamilton–Jacobi–Bellman equation in optimal control problems. The methods described below are based on the connection between PDEs and stochastic processes, established already in the pioneering work of Bachelier, Einstein, Smoluchowski and Langevin on financial markets, heat diffusion and the kinetic theory of gases (see [54,169] for fascinating accounts) and made explicit in the Feynman–Kac formula [93].

In this section and the next, we consider the solution by neural network methods of a class of partial differential equations which arise as the *backward Kolmogorov equation* of stochastic processes known as *Itô diffusions* as proposed in [6]. We begin with linear parabolic second-order partial differential equations in nondivergence form

$$\begin{aligned} \partial_t u(t,x) + \frac{1}{2}\sigma\sigma^T(t,x) : \nabla^2 u(t,x) + \mu(t,x)\cdot\nabla u(t,x) &= 0, && (t,x)\in[0,T)\times\mathbb{R}^d, \\ u(T,x) &= g(x), && x\in\mathbb{R}^d, \end{aligned} \tag{10}$$

and subsequently move to more general PDEs. We consider the pure Cauchy problem, allowing the state variable x to vary throughout $\mathbb{R}^d$. Here, $d\in\mathbb{N}$ is the spatial dimension, $\nabla u(t,x)$ and $\nabla^2 u(t,x)$ denote the gradient and Hessian of the function u, respectively, the colon symbol denotes the Frobenius inner product of $d\times d$ matrices, that is, $A : B = \sum_{i,j=1}^d a_{ij}\, b_{ij}$, and the dot symbol the Euclidean inner product on $\mathbb{R}^d$. Let the coefficient functions $\mu : [0,T]\times\mathbb{R}^d \to \mathbb{R}^d$ (drift) and $\sigma : [0,T]\times\mathbb{R}^d \to \mathbb{R}^{d\times d}$ (diffusion) be globally Lipschitz continuous. Due to the stochastic process connection, (10) is posed as a *final time problem* with prescribed data at time $t=T$ given by the function $g : \mathbb{R}^d \to \mathbb{R}$. The simple change of variables $t\mapsto T-t$ yields the more familiar initial value form

$$\begin{aligned} \partial_t u(t,x) - \frac{1}{2}\sigma\sigma^T(t,x) : \nabla^2 u(t,x) - \mu(t,x)\cdot\nabla u(t,x) &= 0, && (t,x)\in(0,T]\times\mathbb{R}^d, \\ u(0,x) &= g(x), && x\in\mathbb{R}^d. \end{aligned} \tag{11}$$

Equations in nondivergence form like the backward Kolmogorov equation (10) with leading term $\sigma\sigma^T(t,x) : \nabla^2 u(t,x)$ typically arise in the context of stochastic differential equations due to the Itô formula, see [52,87,165]. Such problems play a central role in mathematical finance, for example, in the valuation of complex financial products as well as in stochastic optimal control problems and the solution of second-order Hamilton–Jacobi–Bellman equations [145,172], where the nondivergence form of the differential operator is again due to the stochastic influence. Equations of nondivergence type (10) also arise in the numerical solution of highly nonlinear second-order PDEs that have been linearized, for example, when applying Newton's method. Typical examples include the Monge-Ampère equation [11,26,51]. Classical and strong solutions of problems in nondivergence form are analyzed in [55, Ch. 6, 9]. In contrast to nondivergence PDEs, many problems in applied mathematics arise in divergence form consisting of an operator with leading term $\nabla\cdot[\widetilde{A}(t,x)\nabla u(t,x)]$. Given sufficient smoothness, each operator in divergence form can be brought into nondivergence form by setting $A(t,x)=\widetilde{A}(t,x)$ and subtracting the row-wise divergence $\nabla\cdot\widetilde{A}(t,x)$ from the first-order term. Even if $\widetilde{A}$ is smooth, however, this may result in strongly dominating convection in the resulting equation, introducing further challenges.

Following [6], the method reviewed here can be used to construct approximate solutions of a Kolmogorov PDE (10) or (11) at a fixed time on some bounded domain of interest $D\subset\mathbb{R}^d$. Similar to the technique reviewed in Section 2, a neural network is employed to approximate this solution. The authors of [6] applied their method to a number of examples including the heat equation, the Black-Scholes option pricing equation and others with particular emphasis on the accurate and fast solution in *high dimensions*. Classical numerical approximation schemes for Kolmogorov partial differential equations are numerous, and include finite difference approximations [25,97,98], finite element methods [21,27,55,130], numerical schemes based on Monte-Carlo methods [56,59,60,64], as well as approximations based on a discretization of the underlying stochastic differential equations (SDEs) [76,96]. Establishing a link of the proposed method to the classical approaches, which might be highly accurate and efficient in up to three dimensions, it shares also similarity to Monte-Carlo methods since it relies on the connection between PDEs and SDEs in the form of the

Feynman–Kac theorem and uses a discrete approximation of the SDE associated with equation (10). The reviewed method shares many ideas published in a number of papers, in particular there is a strong connection to [43,69] where the Deep BSDE solver, to be presented in detail in Section 4, is introduced.

In [92] it is proven that deep neural networks are able to overcome the curse of dimensionality for linear backward Kolmogorov PDEs with constant diffusion and nonlinear drift coefficients. In particular, it is shown that the number of parameters in the neural network grows at most polynomially in both the dimension of the PDE $(d+1)$ and the reciprocal of the desired approximation accuracy. We note, however, that training a neural network in general is known to be an NP-hard problem, [170, Sec. 20.5].

3.1 | The Feynman–Kac formula

The method reviewed here [6] is based on the Feynman–Kac formula for Kolmogorov PDEs which connects the solution of the PDE (10) and the expectation of a stochastic process. In order to understand the method fully, we recall the link between PDEs and SDEs formally in this section; for a thorough treatment we refer to [133,165].

In a nutshell, the Feynman–Kac theorem states that for every $(t,x) \in [0,T] \times \mathbb{R}^d$ the solution $u(t,x)$ of the Kolmogorov backward equation (10) can be expressed as the conditional expectation of a stochastic process $\{X_s\}_{s\in[t,T]}$ starting at $X_t = x$, that is,

$$u(t,x) = \mathbb{E}[g(X_T)|X_t = x]. \tag{12}$$

Here, $g : \mathbb{R}^d \to \mathbb{R}$ is the final time prescribed in (10) and $\mathbb{E}[\cdot|X_t = x]$ denotes expectation conditioned on $X_t = x$. One immediate consequence is that, for all $x \in \mathbb{R}^d$, we have

$$u(T,x) = \mathbb{E}[g(X_T)|X_T = x] = g(x). \tag{13}$$

Another implication that can be obtained by the law of iterated conditional expectation is that for all $s \in [t,T]$

$$u(t,x) = \mathbb{E}[u(s,X_s)|X_t = x]. \tag{14}$$

We assume that we are given a filtered probability space $(\Omega, \mathcal{F}, \mathbb{P}; \mathbb{F})$ equipped with the filtration $\mathbb{F} = \{\mathcal{F}_t\}_{t\in[0,T]}$ induced by a d-dimensional Brownian motion $\{W_t\}_{t\in[0,T]}$. The stochastic process $\{X_s\}_{s\in[t,T]}$ can be characterized as the solution of the stochastic differential equation (SDE)

$$X_s = x + \int_t^s \mu(\tau, X_\tau)\,\mathrm{d}\tau + \int_t^s \sigma(\tau, X_\tau)\,\mathrm{d}W_\tau. \tag{15}$$

Assuming Lipschitz conditions on the coefficients μ and σ, a pathwise unique strong solution[2] to (15) always exists, where μ and σ are the coefficients in (10). Note that the second integral in (15) is an Itô integral, that is, a particular type of stochastic integral. We refer to [148,165] for details concerning stochastic analysis and SDEs in general.

Given a strong solution of (15) $\{X_s\}_{s\in[t,T]}$ and a real-valued function $v \in C^{1,2}([0,T) \times \mathbb{R}^d; \mathbb{R}) \cap C^0([0,T] \times \mathbb{R}^d; \mathbb{R})$ applying Itô's formula [87,165], a generalization of the chain rule for (in generally nondifferentiable) stochastic processes, gives that for any $s \in [t,T]$

$$v(s,X_s) = v(t,x) + \int_t^s \partial_t v(\tau,X_\tau)\,\mathrm{d}\tau + \int_t^s \nabla v(\tau,X_\tau)\cdot \mathrm{d}X_\tau + \frac{1}{2}\int_t^s \nabla^2 v(\tau,X_\tau) : \sigma\sigma^T(\tau,X_\tau)\,\mathrm{d}\tau, \tag{16}$$

which, upon substituting $\mathrm{d}X_\tau$ by its definition (15), becomes

$$= v(t,x) + \int_t^s \left(\partial_t v + \frac{1}{2}\nabla^2 v : \sigma\sigma^T + \nabla v \cdot \mu\right)(\tau,X_\tau)\,\mathrm{d}\tau + \int_t^s \nabla v \cdot \sigma(\tau,X_\tau)\,\mathrm{d}W_\tau. \tag{17}$$

[2] Pathwise uniqueness means that if $\{X_s\}_{s\in[t,T]}$ and $\{Y_s\}_{s\in[t,T]}$ are both solutions of (15), then $\mathbb{P}(X_s = Y_s \forall s \in [t,T]) = 1$.

Since this is valid for any $s \in [t, T)$, it holds in particular for $s = t + h$ with $h > 0$, which gives

$$v(t+h, X_{t+h}) = v(t,x) + \int_t^{t+h} \left(\partial_t v + \frac{1}{2}\nabla^2 v : \sigma\sigma^T + \nabla v \cdot \mu\right)(\tau, X_\tau)\,\mathrm{d}\tau + \int_t^{t+h} \nabla v \cdot \sigma(\tau, X_\tau)\,\mathrm{d}W_\tau.$$

Setting $v = u$ given by the expression (14) for $s = t + h$, we obtain

$$\begin{aligned} 0 &= \mathbb{E}\left[\int_t^{t+h} \left(\partial_t u + \frac{1}{2}\nabla^2 u : \sigma\sigma^T + \nabla u \cdot \mu\right)(\tau, X_\tau)\,\mathrm{d}\tau + \int_t^{t+h} \nabla u \cdot \sigma(\tau, X_\tau)\,\mathrm{d}W_\tau \Big| X_t = x\right] \\ &= \mathbb{E}\left[\int_t^{t+h} \left(\partial_t u + \frac{1}{2}\nabla^2 u : \sigma\sigma^T + \nabla u \cdot \mu\right)(\tau, X_\tau)\,\mathrm{d}\tau \Big| X_t = x\right], \end{aligned}$$

where we have used the fact that the stochastic integral is a continuous local martingale and therefore its conditional expectation vanishes. Dividing by $h > 0$ and taking the limit as h goes to zero yields, by the mean-value theorem,

$$\partial_t u(t,x) + \frac{1}{2}\sigma\sigma^T(t,x) : \nabla^2 u(t,x) + \mu(t,x) \cdot \nabla u(t,x) = 0 \qquad \forall (t,x) \in [0,T) \times \mathbb{R}^d,$$

confirming that the function given by the Feynman–Kac formula (14) solves PDE (10).

3.2 | Methodology

A number of numerical methods for high-dimensional PDEs have used the Feynman–Kac connection relating PDEs and SDEs in combination with the slow but dimension-independent convergence of Monte Carlo integration [12,13,24,32], and many more are listed in [5]. The method from [6] reviewed here adds a neural network representation of the PDE solution which is trained in the course of Monte Carlo sampling. It yields an approximation of the solution $u = u(t,\cdot) : \mathcal{D} \to \mathbb{R}$ of the final time problem (10) restricted to a bounded domain of interest $\mathcal{D} \subset \mathbb{R}^d$ at a selected time $t \in [0, T]$. In the following we discuss the methodology in detail for specifically $t = 0$.

3.2.1 | Generation of training data

Similar to the PINN method discussed in Section 2, the method to solve backward Kolmogorov equations does not require any approximate or exact solution values. Instead, it relies on the generation of a large amount of training data based on the stochastic process connected to the PDE (10).

To be more precise, we consider training data $\{(x^i, y^i)\}_{i=1}^{n_{\text{data}}}$. Here, the *input* or *independent variable* x is sampled randomly from $X \sim \mathsf{U}(\mathcal{D})$, which ensures that it covers the domain of interest $\mathcal{D}$ sufficiently well if sampled many times. The random *output* (*target variable*) y is defined as a function of x by $Y := g(X_T)$, where X_T is the final value of the stochastic process $\{X_t\}_{t\in[0,T]}$ starting at $X_0 = x$ and evolving according to the SDE

$$X_t = x + \int_0^t \mu(s, X_s)\,\mathrm{d}s + \int_0^t \sigma(s, X_s)\,\mathrm{d}W_s, \qquad 0 \le t \le T. \tag{18}$$

We distinguish two cases:

In cases where the distribution of X_T is explicitly known, we can draw sample pairs (x, y) directly. For example, in the case of a scaled Brownian motion whose dynamics is characterized by $\mu(t,x) \equiv 0$ and $\sigma(t,x) \equiv \sigma$, the solution of (18) is given by

$$X_t = x + \sigma\, W_t,$$

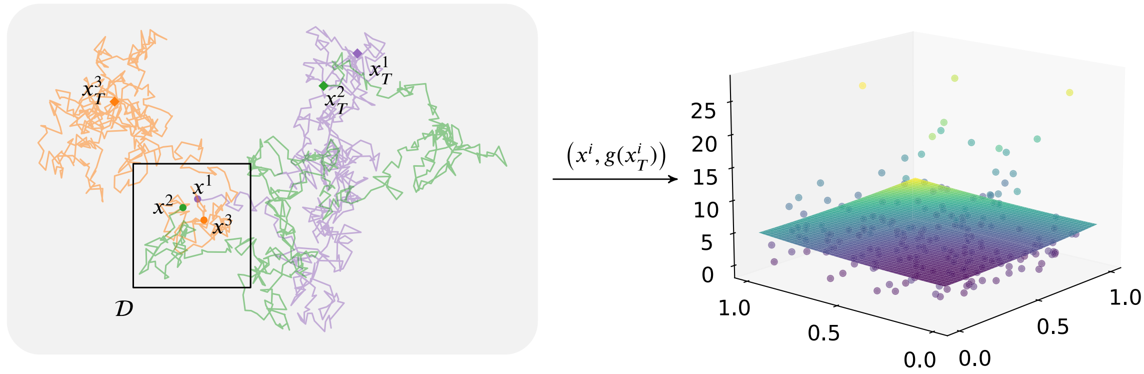

FIGURE 4 Illustration of the data generation process described in Section 3.2. The left panel shows sample paths originating from three different starting values x^i sampled from $\mathcal{D}$ for $\sigma \equiv \sqrt{2}I_{d\times d}$. The right panel shows the exact solution surface $u(0, x)$ along with a number of data pairs $\{(x^i, y^i)\} = \{(x^i, g(x_T^i))\}$ seen to exhibit a large variation around the solution

where W_t is a path of a standard d-dimensional Brownian motion. Since $W_t \sim \mathsf{N}(0, t\, I_{d\times d})$, we may simply draw $X \sim \mathsf{U}(\mathcal{D})$ and set $Y := g(X + \sigma\, \sqrt{T}\, \xi)$, where $\xi \sim \mathsf{N}(0, I_{d\times d})$ is a random variable with a d-variate standard normal distribution. Processes for which an explicit distribution is known include Gaussian processes (e.g., Brownian motion, Ornstein–Uhlenbeck processes), geometric Brownian motion and Cox–Ingersoll–Ross processes.

When an explicit distribution of X_t at $t = T$ is not available, we may approximate the continuous-time process $\{X_t\}_{t\in[0,T]}$ by generating approximate sample paths using numerical SDE solvers such as the *Euler–Maruyama scheme*

$$\widetilde{X}_{n+1} := \widetilde{X}_n + \mu(t_n, \widetilde{X}_n)\,(t_{n+1} - t_n) + \sigma(t_n, \widetilde{X}_n)\,(W_{t_{n+1}} - W_{t_n}), \qquad \widetilde{X}_0 := x, \tag{19}$$

where $\widetilde{X}_n \approx X_{t_n}$ is a discrete-time stochastic process approximating X_t at points $0 = t_0 < t_1 < \ldots < t_N = T$ and x is a realization of $X \sim \mathsf{U}(\mathcal{D})$. Note that the increment of a Brownian motion $(W_{t_{n+1}} - W_{t_n}) \sim \mathsf{N}(0, (t_{n+1} - t_n)I_{d\times d})$ is normally distributed. Finally, we set $Y := g(\widetilde{X}_N)$. Strong convergence results for the Euler–Maruyama scheme [96,119] ensure that $\widetilde{X}_n \to X_{t_n}$ as $N \to \infty$ and $\sup_n |t_n - t_{n-1}| \to 0$.

Generating training data via sample paths in this way yields an arbitrary number of easily obtained data pairs (x^i, y^i) with x^i sampled uniformly over $\mathcal{D}$ and y^i resulting from the final data g evaluated at the final state X_T of a trajectory $\{X_s\}_{s\in[0,T]}$ starting at $X_0 = x^i$. One has to bear in mind, however, that these individual measurements may vary strongly, in particular for large end times T and diffusion coefficients σ. The training of the neural network $u_\theta : \mathcal{D} \to \mathbb{R}$ in this way amounts to least squares fitting of u_θ to a point cloud formed by the data pairs $\{(x^i, y^i)\}_{i=1}^{n_{\text{data}}}$. This is illustrated in Figure 4. The left panel shows sample paths originating from three different starting values x^i sampled from $\mathcal{D} = [0, 1]^2$ for $\sigma \equiv \sqrt{2}I_{d\times d}$. Although all processes start within $\mathcal{D}$, they evolve in $\mathbb{R}^d$ according to the SDE (18) and ultimately leave the domain. As a consequence, this method of learning the mapping $u_\theta(0, x)$, $x \in \mathcal{D}$, does not require the formulation of artificial truncation boundary conditions along $\partial\mathcal{D}$ as is the case for conventional discretization methods for PDEs on unbounded domains. The right panel shows the exact solution surface $u(0, x)$ along with a number of data pairs $\{(x^i, y^i)\} \subset \mathbb{R}^2 \times \mathbb{R}$ seen to exhibit a large variation around the solution. Despite the presence of substantial noise in the solution samples, there is no danger of *overfitting* for this method as long as sufficiently many data pairs generated and the training is not restricted to a fixed small number of samples. This poses no restriction as the generation of new trajectories and hence solution samples are very inexpensive and allows for an essentially unlimited supply. This is particularly true when the distribution of X_T is explicitly known and therefore no numerical path integration is necessary as in the examples given below.

3.2.2 | Neural network approximation

Similar to the PINN approach discussed in Section 2, the unknown solution of the PDE (10) at a fixed time, here $t = 0$, is approximated by a (single) neural network. We denote this approximation by $u_\theta : \mathcal{D} \to \mathbb{R}$, where θ collects again all unknown parameters of the network.

The training of the model amounts to a simple regression task. Given a batch of training data $\{(x^i, y^i)\}_{i=1}^{n_{\text{batch}}}$, the objective is to minimize the mean squared error

$$\frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} |y^i - u_\theta(x^i)|^2,$$

which corresponds from the perspective of the underlying stochastic process to the minimization of

$$\mathbb{E}[|g(X_T) - u_\theta(x)|^2]$$

where X_T is the solution of the SDE (15) starting in $X_0 = x$. This may be viewed as a discrete approximation of a continuous problem, for which it is shown in [6, Prop. 2.7] that, under suitable assumptions, there exists a unique continuous function $u^* : \mathcal{D} \to \mathbb{R}$ such that

$$\mathbb{E}\left[|g(X_T) - u^*(x)|^2\right] = \inf_{v \in C(\mathcal{D};\mathbb{R})} \mathbb{E}\left[|g(X_T) - v(x)|^2\right]. \tag{20}$$

Furthermore, it holds for every $x \in \mathcal{D}$ that $u^*(x) = u(0, x)$.

The network proposed in [6], which is also employed in our numerical tests in Section 3.2.3, has the structure

```
Input ⇝ BN ⇝ (Dense ⇝ BN ⇝ TanH) ⇝ (Dense ⇝ BN ⇝ TanH) ⇝ Dense ⇝ BN ⇝ Output
```

where the notation is as follows:

- `BN` indicates a *batch normalization* step [86], which is a technique of normalizing each training mini-batch within the network architecture to make the model less sensitive in terms of proper weight initialization and allows for larger step sizes and faster training. This is effected by additional parameters that scale and shift the neurons that enter the `BN` layer componentwise. These parameters are learned in the same way as all unknown parameters in the neural network, for example, by a mini-batch gradient descent type algorithm.
- `Dense` indicates a fully connected layer *without* bias term, that is, a matrix-vector product with a learnable weight matrix. Due to the subsequent shifting during the `BN` layer, a bias term can be omitted since its effect would be canceled.
- `TanH` indicates the application of the componentwise hyperbolic tangent activation function.

The network is trained with the Adam optimizer [95], a variant of the stochastic gradient descent method based on an adaptive estimation of first-order and second-order moments to improve the speed of convergence. An explanatory walkthrough of the implementation of the complete algorithm is given in the accompanying Jupyter notebook `Feynman_Kac_Solver.ipynb`.

3.2.3 | Example: Heat equation

In this section, we want to solve the heat equation in d dimensions by means of the solver proposed in [6] and consider the initial value problem

$$\begin{aligned} \partial_t u(t,x) &= \Delta u(t,x) & (t,x) &\in (0,T] \times \mathbb{R}^d \\ u(0,x) &= \|x\|^2 & x &\in \mathbb{R}^d, \end{aligned} \tag{21}$$

where $\Delta u = \sum_{i=1}^d \partial^2 u / \partial x_i^2$ denotes the Laplacian of u. One can easily verify that the solution is given by

$$u(t,x) = \|x\|^2 + 2\ t\ d.$$

We tested two different step size strategies: a decaying piecewise constant learning rate with step sizes $\delta(n) = 10^{-3}\,\mathbf{1}_{\{n \le 250\,000\}} + 10^{-4}\,\mathbf{1}_{\{250\,000 < n \le 500\,000\}} + 10^{-5}\,\mathbf{1}_{\{500\,000 < n\}}$ as was employed in [6] and an exponentially decaying rate with

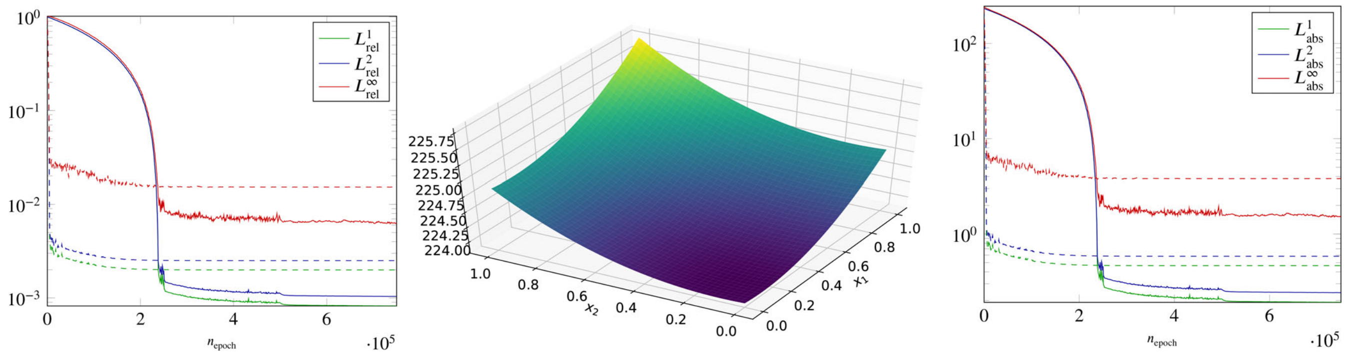

FIGURE 5 Evolution of relative (left) and absolute (right) errors for a decaying piecewise constant learning rate (solid) and an exponentially decaying rate (dashed) for the 100-dimensional heat equation (21), estimated by means of the Monte-Carlo method in order to approximate the integrals in dimension 100 with one million samples. Center: Two-dimensional slice through the approximate solution $(x_1, x_2) \mapsto u_\theta(x_1, x_2, 0.5, \ldots, 0.5)$

step sizes $\delta(n) = 0.1 \times 10^{-n/100\ 000}$. The remainder of the parameters are chosen as in [6]. We fixed the number of neurons in the two hidden layers to 200 independent of the dimension. Figure 5 shows the evolution of the absolute and relative approximation errors[3] on $\mathcal{D}$ for the 100-dimensional heat equation.

In our numerical experiments with the heat equation (21) we observed that the quality of the final approximation depends heavily on the chosen learning rate, that is, the step sizes used in the gradient method. A comparison between the evolutions of the relative and absolute errors for the two aforementioned learning rate strategies is displayed in Figure 5, together with a two-dimensional slice through the 100-dimensional solution. Together with Table 2, this indicates that it seems to be better to stay conservative and take smaller steps from the beginning on. Shown are errors for the two step size scenarios at $n_{\text{epochs}} = 100\,000$ and $n_{\text{epochs}} = 750\,000$. Although the errors decrease faster in the beginning for the exponentially decaying step sizes that start with larger steps, the errors seems to saturate at a higher level. This might be due to the algorithm settling into some local minimum. For the decaying piecewise constant learning rate, Figure 5 shows two distinct phases of error decay: While the first phase until approximately epoch number 250 000 is characterized by an accelerating decay of the errors probably due to mainly shifting the solution slowly towards the image range (200 to 300) of the solution, the second phase decays at a much slower rate which might correspond to the reduction rate of the Monte Carlo error. The exponentially decaying learning rate decays much faster in the beginning but settles at a higher absolute and relative error.

We also observe that it seems to be difficult to improve the achievable relative and absolute errors, see Table 3. In this example no SDE time-stepping is necessary, as the end of the sample paths X_T can be drawn directly. In particular, this incurs no discretization error.

A general recommendation on how to select the neural network architecture and parameter selection could be part of further research. This however, is a problem prevalent in many fields of scientific machine learning, see [144] for a discussion on selecting deep ReLU network architectures. Nevertheless, one has to bear in mind that problems in such a high spatial dimension have been considered absolutely infeasible for a long time in terms of numerical approximations. In particular, for problems in financial mathematics where derivatives, for example, options, often depend on a basket of more than 100 underlying risky assets (which determine the spatial dimension of the pricing PDE), the importance of having a feasible algorithm cannot be denied. Note that the accompanying code includes as a second example an option pricing problem.

3.3 | Linear parabolic PDEs in general form

The Feynman–Kac formula may be extended to the full class of linear parabolic equations, see [94, Ch. 5 Theorem 7.6]. Specifically, adding a zeroth order term with nonnegative potential $r : [0, T] \times \mathbb{R}^d \to [0, \infty)$ as well as a source term f :

[3] All errors shown in the plots are approximated by Monte-Carlo estimation with one million samples.

TABLE 2 Absolute and relative approximation errors for the d-dimensional heat equation (21)

Experiment	Dim	$L^1_{\text{rel}}(D)$	$L^2_{\text{rel}}(D)$	$L^\infty_{\text{rel}}(D)$	$L^1_{\text{abs}}(D)$	$L^2_{\text{abs}}(D)$	$L^\infty_{\text{abs}}(D)$	Time
Exp. decay $n_{\text{epoch}} = 250\,000$	10	1.53×10^{-3}	1.97×10^{-3}	3.09×10^{-2}	3.60×10^{-2}	4.65×10^{-2}	8.58×10^{-1}	896.45
	50	2.14×10^{-3}	2.70×10^{-3}	2.61×10^{-2}	2.49×10^{-1}	3.14×10^{-1}	3.34×10^{0}	940.53
	100	1.97×10^{-3}	2.47×10^{-3}	1.35×10^{-2}	4.59×10^{-1}	5.76×10^{-1}	3.35×10^{0}	1038.80
Exp. decay $n_{\text{epoch}} = 750\,000$	10	1.49×10^{-3}	1.91×10^{-3}	3.07×10^{-2}	3.49×10^{-2}	4.51×10^{-2}	8.52×10^{-1}	2692.69
	50	2.12×10^{-3}	2.67×10^{-3}	2.61×10^{-2}	2.47×10^{-1}	3.12×10^{-1}	3.34×10^{0}	2830.50
	100	1.96×10^{-3}	2.45×10^{-3}	1.32×10^{-2}	4.56×10^{-1}	5.73×10^{-1}	3.28×10^{0}	3129.64
Piecewise decay $n_{\text{epoch}} = 250\,000$	10	2.40×10^{-3}	3.01×10^{-3}	1.30×10^{-2}	5.60×10^{-2}	7.02×10^{-2}	3.45×10^{-1}	1122.72
	50	1.59×10^{-3}	2.00×10^{-3}	1.10×10^{-2}	1.85×10^{-1}	2.33×10^{-1}	1.25×10^{0}	1153.85
	100	1.44×10^{-3}	1.81×10^{-3}	9.66×10^{-3}	3.36×10^{-1}	4.21×10^{-1}	2.26×10^{0}	1203.94
Piecewise decay $n_{\text{epoch}} = 750\,000$	10	5.90×10^{-4}	7.43×10^{-4}	4.29×10^{-3}	1.37×10^{-2}	1.73×10^{-2}	9.19×10^{-2}	3405.13
	50	7.76×10^{-4}	9.87×10^{-4}	7.15×10^{-3}	9.04×10^{-2}	1.15×10^{-1}	8.66×10^{-1}	3493.81
	100	8.20×10^{-4}	1.04×10^{-3}	6.24×10^{-3}	1.91×10^{-1}	2.42×10^{-1}	1.47×10^{0}	3659.94

TABLE 3 Absolute and relative errors of the 100-dimensional heat equation with decaying piecewise constant learning rate for three different neural network architectures after 750 000 training epochs

Experiment	$L^1_{\text{rel}}(D)$	$L^2_{\text{rel}}(D)$	$L^\infty_{\text{rel}}(D)$	$L^1_{\text{abs}}(D)$	$L^2_{\text{abs}}(D)$	$L^\infty_{\text{abs}}(D)$	Time
$n_{layers} = 2$, $n_{neuron} = 200$	8.20×10^{-4}	1.04×10^{-3}	6.24×10^{-3}	1.91×10^{-1}	2.42×10^{-1}	1.47×10^{0}	3659.94
$n_{layers} = 3$, $n_{neuron} = 300$	7.75×10^{-4}	9.79×10^{-4}	6.64×10^{-3}	1.81×10^{-1}	2.28×10^{-1}	1.51×10^{0}	4704.55
$n_{layers} = 4$, $n_{neuron} = 400$	7.24×10^{-4}	9.15×10^{-4}	5.43×10^{-3}	1.69×10^{-1}	2.13×10^{-1}	1.30×10^{0}	8483.34

$[0,T]\times\mathbb{R}^d \to \mathbb{R}$, the final time problem (10) becomes

$$\begin{aligned} \partial_t u(t,x) + \frac{1}{2}\sigma\sigma^T(t,x) : \nabla^2 u(t,x) + \mu(t,x)\cdot\nabla u(t,x) - r(t,x)\,u(t,x) + f(t,x) = 0, \qquad & (t,x)\in[0,T)\times\mathbb{R}^d, \\ u(T,x) = g(x), \qquad & x\in\mathbb{R}^d. \end{aligned} \tag{22}$$

A sufficiently smooth solution of (22) admits the Feynman–Kac representation

$$u(t,x) = \mathbb{E}\left[\int_t^T \mathrm{e}^{-\int_t^\tau r(v,X_v)\,\mathrm{d}v} f(\tau,X_\tau)\,\mathrm{d}\tau + \mathrm{e}^{-\int_t^T r(v,X_v)\,\mathrm{d}v}\, g(X_T) | X_t = x\right] \qquad \forall (t,x)\in[0,T]\times\mathbb{R}^d, \tag{23}$$

which simplifies to (12) for $f\equiv 0$ and $r\equiv 0$.

Algorithmically, this can be considered within the same framework as discussed in Section 3.2. In particular, it does not change the generation of samples of the stochastic process $\{X_s\}_{s\in[0,T]}$. In the case of a discrete approximation $\{\widetilde{X}_n\}_{n=0}^N$ generated by the Euler–Maruyama scheme (19), a simple approximation of the corresponding output variable Y can be given by

$$Y = \sum_{n=0}^{N-1} \widetilde{R}_n\, f(t_n,\widetilde{X}_n)\,(t_{n+1}-t_n) + \widetilde{R}_N\, g(\widetilde{X}_N) \tag{24}$$

with

$$\widetilde{R}_n := \exp\left(-\sum_{j=0}^{n-1} r(t_j,\widetilde{X}_j)\,(t_{j+1}-t_j)\right) = \widetilde{R}_{n-1}\ \exp\big(-r(t_{n-1},\widetilde{X}_{n-1})\,(t_n-t_{n-1})\big), \qquad \widetilde{R}_0 := 1.$$

Here, $\widetilde{R}_n$ is a discrete approximation of the term $\exp\left(-\int_0^{t_n} r(v, X_v)\, \mathrm{d}v\right)$. In the case of a space-independent or even constant potential function $r(t,x)$, this can be simplified, for example, $\widetilde{R}_n = e^{-r\ t_n}$ in the case of a constant potential $r(t,x) = r$. The discrete approximation (24) can then be used to generate training samples $\{(x^i, y^i)\}_{i=1}^{n_{\text{data}}}$ and train a neural network $u_\theta : \mathcal{D} \to \mathbb{R}$ which approximates the solution of the PDE (22) in the domain of interest $\mathcal{D}$ at time $t = 0$.

An alternative formulation of (23) can be obtained by means of the concept of *killed* stochastic processes, see [133, Sec. 8.2] or [173, Ch. 15]. Such a process $(\widehat{X}_t)_{t\in[0,T]}$ behaves exactly like the process $\{X_t\}_{t\in[0,T]}$, but becomes undefined or "killed" at a certain random (killing) time ζ, after which the process $\widehat{X}_t$ is assigned a so-called "coffin state". Here, ζ is an exponentially distributed random time with "killing rate" $r(t,x)$. Thus, it can be shown, see [133, Sec. 8.2], that the solution of the parabolic PDE (22) admits the representation

$$u(t,x) = \mathbb{E}\left[\int_t^T f(\tau, \widehat{X}_\tau)\, \mathrm{d}\tau + g(\widehat{X}_T) | \widehat{X}_t = x\right] \qquad \forall (t,x) \in [0,T] \times \mathbb{R}^d. \tag{25}$$

Finally, we mention that boundary conditions can be incorporated into the PDE-SDE framework by considering certain kinds of stochastic processes. For example, in the case of a linear parabolic PDE as in (22) but posed on a bounded spatial domain $\mathcal{O}$ in place of $\mathbb{R}^d$, the appropriate concept is that of *stopped processes*, which evolve according to the SDE (18) in $\mathcal{O}$ and are stopped as soon as they hit the parabolic boundary $(0,T) \times \partial\mathcal{O} \cup \{T\} \times \overline{\mathcal{O}}$ where $\overline{\mathcal{O}}$ denotes the closure of $\mathcal{O}$. For further details, see [24,41,124,139,178] and the references therein.

3.4 | Summary and extensions

The approach discussed in this section can be used to solve backward Kolmogorov equations in high-dimensions. It is based on the Feynman–Kac connection between SDEs and PDEs and can be implemented efficiently using TensorFlow and other scientific machine learning software environments without deeper knowledge since it reduces, in essence, to a regression problem where the data is sampled either directly or via SDE time-stepping methods such as the Euler–Maruyama scheme.

In [17], a similar technique is proposed for the solution of *parametric* linear Kolmogorov PDEs. Again, this methodology generates training data by sampling; the employed neural networks, however, are based on a multilevel architecture with residual connections.

4 | SEMILINEAR PDES IN HIGH DIMENSIONS

In this section we extend the methodology of Section 3 to solving *semilinear* PDEs obtained by allowing the lower-order terms in (10) and (22) to depend nonlinearly on the solution and its gradient. This results in the final value problem

$$\begin{aligned} \partial_t u(t,x) + \frac{1}{2}\sigma\sigma^T(t,x) : \nabla^2 u(t,x) + \mu(t,x)\cdot\nabla u(t,x) + f(t,x,u(t,x),\sigma^T(t,x)\nabla u(t,x)) = 0, \qquad & (t,x) \in [0,T) \times \mathbb{R}^d, \\ u(T,x) = g(x), \qquad & x \in \mathbb{R}^d, \end{aligned} \tag{26}$$

with drift μ, diffusion σ and final data g as before. The function $f : [0,T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ containing lower order terms can depend in a general way on the independent variables t, x as well as on the solution $u(t,x)$ and its transformed gradient $(\sigma^T\nabla)u(t,x)$. The nondivergence form of the leading-order term as well as the specific dependence on $\sigma^T\nabla u$ again result from the connection between PDEs and stochastic processes. As we will see in Section 4.1, the presence of these dependencies requires extending the numerical solution method to include additional approximating stochastic processes for ∇u.

Problems of the form (26) arise in physics in the form of, for example, the Allen–Cahn, Burgers or reaction–diffusion equations; in finance, for example, for pricing derivatives with default risk [28,36,48]; and stochastic control problems, see [145]. The method discussed below is an extension to that presented in Section 3 in that it is also based on the PDE-SDE connection, but in this case it is the correspondence of nonlinear PDEs with *backward stochastic differential equations (BSDEs)* [94,165]. In the linear case discussed in Section 3 the approximation of the solution u at time $t = 0$ is based on a neural network approximation of the mapping $u(0,\cdot) : \mathcal{D} \to \mathbb{R}$, the Feynman–Kac representation $u(0,x) = \mathbb{E}[g(X_T)|X_0 = x]$ for $x \in \mathcal{D}$ and generating a large number of sample paths of the stochastic process $\{X_t\}_{t\in[0,T]}$

determined by (15) to approximate the conditional expectation and train the model. Using the theory of BSDEs, it is possible to treat nonlinearities of the type contained in (26).

The specific method presented here was proposed in [43,69] and is based on earlier work [68]. Again, the focus lies on solving high-dimensional problems and overcoming one source of the curse of dimensionality [10]: a high-dimensional state space (large d). In recent years, a number of approaches have been proposed for mitigating or overcoming the curse of dimensionality in solving high-dimensional PDEs. In the meantime, a number of theoretical results indicate this may indeed be possible; an (incomplete) list is given in Section 5. In [85] it is proven that deep ReLU networks, that is, neural networks with multiple hidden layers and the rectified linear unit activation function, are in theory able to overcome the curse of dimensionality for certain kinds of the semilinear parabolic equations with nonlinearities which do not involve the gradient. This is similar to the linear case [92]. In particular, it can be shown that the number of parameters in the neural network grows at most polynomially in both the dimension of the PDE ($d+1$) and the reciprocal of the desired approximation accuracy. Note however, that training a neural network in general is a NP-hard problem, [[170], Sec. 20.5]. The proof relies on full history recursive multilevel Picard approximations, see also [8,47].

The approach discussed below can be used to construct an approximate solution of the semilinear problem (26) at a fixed point in time over a bounded domain of interest $\mathcal{D} \subset \mathbb{R}^d$ by sampling the initial point x uniformly on $\mathcal{D}$ as in Section 3.2. For simplicity, however, we consider the problem of determining the solution at a specific point in space and time, that is, to determine $u(0,x)$ for fixed $x \in \mathbb{R}^d$.

4.1 | Theoretical background

As in Section 3, we consider a time-evolution $\{X_t\}_{t\in[0,T]}$ in state space $\mathbb{R}^d$ driven by the *forward* SDE

$$X_t = x + \int_0^t \mu(s,X_s)\,\mathrm{d}s + \int_0^t \sigma(s,X_s)\,\mathrm{d}W_s \tag{27}$$

starting at $x \in \mathbb{R}^d$, with underlying probability space $(\Omega, \mathcal{F}, \mathbb{P}; \mathbb{F})$ with filtration $\mathbb{F} = \{\mathcal{F}_t\}_{t\in[0,T]}$ induced by a d-dimensional Brownian motion $\{W_t\}_{t\in[0,T]}$. In Section 3.1 we concluded from Itô's formula in (16)–(17) that, given a sufficiently smooth function $v : [0,T] \times \mathbb{R}^d \to \mathbb{R}$, the dynamics of the *value process* $Y_t := v(t, X_t)$ is governed by the SDE (now written in differential notation)

$$\mathrm{d}Y_t = \left(\partial_t v + \frac{1}{2}\sigma\sigma^T : \nabla^2 v + \nabla v \cdot \mu\right)(t,X_t)\,\mathrm{d}t + \left(\sigma^T \nabla v\right)(t,X_t)\cdot \mathrm{d}W_t. \tag{28}$$

As in Section 3.1 we now assume a sufficiently smooth solution u of (26) to exist, set $v = u$ in (28), and introduce a third stochastic process $Z_t := \left(\sigma^T \nabla u\right)(t, X_t)$ to obtain

$$\mathrm{d}Y_t = -f(t,X_t,Y_t,Z_t)\,\mathrm{d}t + Z_t\cdot \mathrm{d}W_t, \qquad Y_T = g(X_T).$$

This SDE with final condition $Y_T = g(X_T)$ inherited from (26) is known as the *BSDE associated with* (26) and reads, in integral notation, as

$$Y_t = g(X_T) + \int_t^T f(s,X_s,Y_s,Z_s)\,\mathrm{d}s - \int_t^T Z_s\cdot\,\mathrm{d}W_s. \tag{29}$$

Under suitable regularity assumptions on the functions μ, σ, f and g, the SDEs (27) and (29) possess a unique solution (X_t, Y_t, Z_t) and the link to the nonlinear PDE is given by a generalization of the Feynman–Kac formula which states that for all $t \in [0,T]$ there holds $\mathbb{P}$-a.s. that

$$Y_t = u(t,X_t) \quad \text{and} \quad Z_t = \left(\sigma^T\,\nabla u\right)(t,X_t). \tag{30}$$

In view of the analogy to (12) and (23), the identities (30) are sometimes referred to as the *nonlinear Feynman–Kac representation* [[145], Sec. 6.3]. The system consisting of (27) and (29) is called a *forward-backward stochastic differential equation*

(FBSDE) [137,138,140]. We note that the forward SDE (27) does not depend on Y_t and Z_t, and can thus be solved independently. As a result, the desired solution value $u(0,x)$ can now be found by solving the FBSDE and evaluating Y_0 in (30). We refer to [[182], Ch. 7] for a general account on the solvability of FBSDEs.

The difference to the procedure described in Section 3.3 is that the solution of the value process $\{Y_s\}_{s\in[0,T]}$ is now more involved due to the nonlinear term f and its dependence on $u(t,x)$ and $(\sigma^T\nabla)u(t,x)$.

4.2 | Deep BSDE solver

The algorithm termed *deep BSDE solver* in [69] constructs an approximation to a solution value $u(0,x)$ of the PDE (26) by way of solving the associated FBSDE (27), (29), yielding $u(0,x)=Y_0$ as summarized in Section 4.1. We now proceed to show how this is achieved using neural networks.

Starting with a discretization of the time domain $[0,T]$ into N equidistant intervals with steps $0=t_0<t_1<\cdots<t_N=T$ and step size $\Delta t=T/N$, we generate approximate sample paths of the continuous time process $\{X_t\}_{t\in[0,T]}$ using the Euler–Maruyama scheme for the forward SDE (27) which yields the discrete time process

$$\widetilde{X}_{n+1}=\widetilde{X}_n+\mu(t_n,\widetilde{X}_n)\,(t_{n+1}-t_n)+\sigma(t_n,\widetilde{X}_n)\,(W_{t_{n+1}}-W_{t_n})\quad\text{with}\quad \widetilde{X}_0=x. \tag{31}$$

In the same way, we construct sample paths for the backward SDE (29) as

$$\widetilde{Y}_{n+1}=\widetilde{Y}_n-f(t_n,\widetilde{X}_n,\widetilde{Y}_n,\widetilde{Z}_n)\,(t_{n+1}-t_n)+\widetilde{Z}_n\cdot(W_{t_{n+1}}-W_{t_n})\quad\text{with}\quad \widetilde{Y}_N=g(\widetilde{X}_N). \tag{32}$$

Note that the increments of the Brownian motion $(W_{t_{n+1}}-W_{t_n})\sim \mathsf{N}(0,(t_{n+1}-t_n)I_{d\times d})$ are the same in (31) and (32).

The algorithm can be summarized by the following steps:

1. Simulate paths of the discrete state space process $\{\widetilde{X}_n\}_{n=0}^N$ and the corresponding increments of the Brownian motion $\{W_{t_{n+1}}-W_{t_n}\}_{n=0}^{N-1}$ according to the time-stepping scheme (31).
2. Simulate paths of the discrete value process $\{\widetilde{Y}_n\}_{n=0}^N$ according to the time-stepping scheme (32). Closer inspection reveals that (32) contains unknown quantities necessary to carry out the time-stepping: $\widetilde{Y}_0$, which is an approximation of $u(0,x)$ as well as $\widetilde{Z}_n$ for $n=0,\ldots,N-1$, which are approximations of $(\sigma^T\nabla u)(t_n,\widetilde{X}_n)$. These quantities are obtained by training a neural network.

   The quantities $\widetilde{Y}_0\approx u(0,x)$ and $\widetilde{Z}_0\approx(\sigma^T\nabla u)(0,x)$ are treated as individual parameters—both needed only in the point $(0,x)$—and are learned in the course of training. The remaining quantities $\widetilde{Z}_n$, $n=1,\ldots,N-1$ are approximated by neural networks which realize the mapping $x\mapsto(\sigma^T\nabla u)(t_n,x)$ for $n=1,\ldots,N-1$. All neural network parameters to be learned are collected in

   $$\theta=\left(\theta_{u_0},\theta_{\nabla u_0},\theta_{\nabla u_1},\ldots,\theta_{\nabla u_{N-1}}\right),$$

   where $\theta_{u_0}\in\mathbb{R}$, $\theta_{\nabla u_0}\in\mathbb{R}^d$ and $\theta_{\nabla u_n}\in\mathbb{R}^{\rho_n}$ and ρ_n is the number of unknown parameters in the neural network realizing the mapping $x\mapsto(\sigma^T\nabla u)(t_n,x)$ for $n=1,\ldots,N-1$.
3. Since $\widetilde{Y}_N$ should approximate $u(T,\widetilde{X}_N)=g(\widetilde{X}_N)$ according to (32) the network is trained to minimize the mean squared error (MSE) between $\widetilde{Y}_N$ and $g(\widetilde{X}_N)$. For a batch of m simulated pairs $(\widetilde{X}_N,\widetilde{Y}_N)$, this results in the loss function

   $$\phi_\theta(\widetilde{X}_N,\widetilde{Y}_N):=\frac{1}{m}\sum_{i=1}^m\left[\widetilde{Y}_N^i-g(\widetilde{X}_N^i)\right]^2,$$

   where Y_N^i is the output of the neural network. Automatic differentiation of ϕ_θ with respect to the unknowns θ is then employed to obtain the gradient $\nabla_\theta\phi_\theta$, which is then used by an optimization routine, for example, some variant of the stochastic gradient descent method. Note that the same considerations with regard to overfitting as noted at the end of Section 3.2.1 in connection with the Feynman–Kac solver apply here.

The complete network structure is illustrated in Figure 6. The architecture of the subnetworks realizing the mapping $x\mapsto(\sigma^T\nabla u)(t_n,x)$ used in the numerical experiments described below are taken to be the same as in [69], where they are

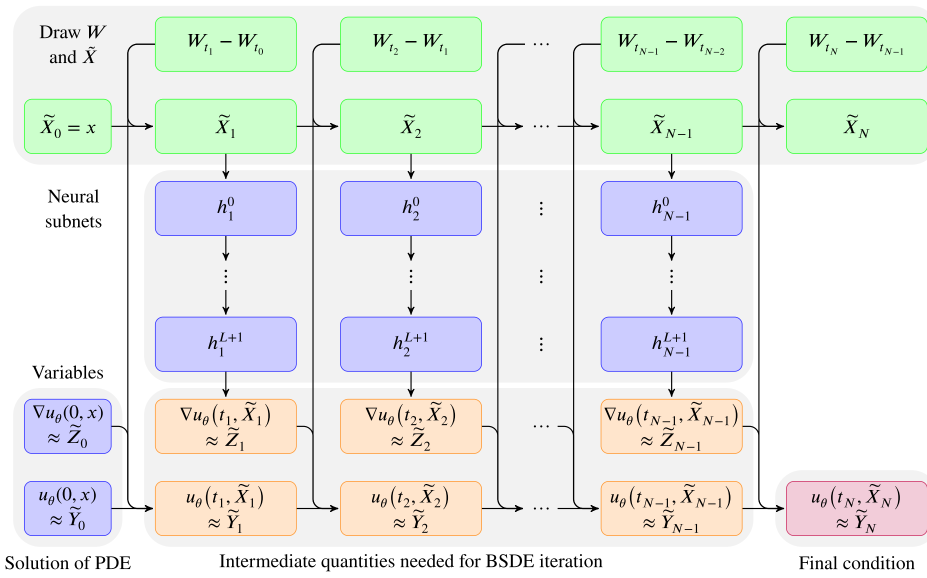

FIGURE 6 Illustration of the complete deep BSDE solver model adapted from [43,69] in the case $\sigma = I_{d\times d}$. The two upper rows express the evolution of the forward process $\{X\}_{n=0}$ starting at $\widetilde{X}_0 = x$ (*green*). The unknown parameters for $u_\theta(0,x)$ and $\nabla u_\theta(0,x)$ (*blue, left*) as well as the parameters in the neural network approximating $\widetilde{Z}_n$, $n = 1, \dots, N-1$ (*blue, center*) are learned by training. The intermediate values $\widetilde{Y}_n$ and $\widetilde{Z}_n$, $n = 1, \dots, N-1$ (*orange*) are needed to establish the link between the desired PDE solution value $u(0,x) \approx \widetilde{Y}_0$ with the given final value $\widetilde{Y}_N = g(\widetilde{X}_N)$ (*red*).

given by

$$\texttt{Input} \rightsquigarrow \texttt{BN} \rightsquigarrow (\texttt{Dense} \rightsquigarrow \texttt{BN} \rightsquigarrow \texttt{ReLU}) \rightsquigarrow (\texttt{Dense} \rightsquigarrow \texttt{BN} \rightsquigarrow \texttt{ReLU}) \rightsquigarrow \texttt{Dense} \rightsquigarrow \texttt{BN} \rightsquigarrow \texttt{Output} \tag{33}$$

Here, `BN` stands for *batch normalization*, `Dense` denotes a fully connected layer without bias term and activation, and `ReLU` denotes the application of the componentwise rectified linear unit activation function $\texttt{ReLU}(x) = \max\{0, x\}$. In terms of the layers in Figure 6 this means the following: first, the inputs $\widetilde{X}_n \in \mathbb{R}^d$ are scaled and shifted componentwise by batch normalization, resulting in $h_n^1 := \texttt{BN}_n^1(\widetilde{X}_n)$; second, the outputs from the first layer are processed by the subsequent block $h_n^2 := \texttt{ReLU}\big(\texttt{BN}_n^2(W_n^2 h_n^1)\big)$ followed by block $h_n^3 := \texttt{ReLU}\big(\texttt{BN}_n^3(W_n^3 h_n^2)\big)$; finally, the output is multiplied by another matrix W_n^4 and batch normalized once more, giving $h_n^4 := \texttt{BN}_n^4(W_n^4 h_n^3) \approx \widetilde{Z}_n$.

To implement the model in TensorFlow [1] all that is needed is to provide a routine that realizes the interaction between the known and unknown quantities and respects the time-stepping scheme (32). In the following, we discuss two examples. An implementation of the methodology for both examples is given in the accompanying Jupyter notebook `DeepBSDE_Solver.ipynb`.

4.3 | Example: Linear-quadratic Gaussian control

We consider the linear-quadratic Gaussian control problem as discussed in [43, Sec. 4.3] [69], and [2]. The goal is to control a stochastic process $\{X_t\}_{t\in[0,T]}$ governed by the SDE

$$X_t = x + 2\int_0^t m_s \,\mathrm{d}s + \sqrt{2}\int_0^t \mathrm{d}W_s$$

with a control $m_t \in \mathbb{R}^d$ entering as the drift term. The solution of the control problem is characterized by the value function, that is, the function $u : [0,T] \times \mathbb{R}^d \to \mathbb{R}$ that gives the minimal expected sum of accumulated running cost and final cost over all admissible control processes[4] from time t onward starting at x:

$$u(t,x) = \min_{\{m_s\}_{s\in[t,T]}} \mathbb{E}\left[\int_t^T \|m_s\|^2 \,\mathrm{d}s + g(X_T) | X_t = x\right] \tag{34}$$

[4]In this setting, an $\mathbb{R}^d$-valued control process $\{m_s\}_{s\in[t,T]}$ is admissible if its value at time s is based only on the information available up to time s. To be precise, the process m_s has to be progressively measurable with respect to the underlying filtration $\mathbb{F}$; see [145,182] for further details.

TABLE 4 Shown are the mean and standard deviations of $u_\theta(0,x)$ and the relative error $|u_\theta(0,x) - u^*|/u^*$, resp., with $u^* \approx 4.5901$ (determined via Monte-Carlo sampling), as well as the mean computation time over 5 consecutive runs with randomly initialized parameters θ after $n_{\text{epochs}} = 2000$ training epochs

Experiment	Mean $u(0,x)$	Std.-dev. $u(0,x)$	Mean relative error	Std.-dev. relative error	Mean time (s)
Simple $(L=0)$	4.6000	1.48×10^{-3}	2.15×10^{-3}	3.23×10^{-4}	3.44
Reference $(L=2)$	4.5989	9.71×10^{-4}	1.91×10^{-3}	2.12×10^{-4}	83.91
$L=3$	4.5991	1.19×10^{-3}	1.95×10^{-3}	2.60×10^{-4}	135.77
$L=5$	4.5983	1.72×10^{-3}	1.77×10^{-3}	3.75×10^{-4}	363.14

The function $g : \mathbb{R}^d \to \mathbb{R}$ is the prescribed final data. The Hamilton–Jacobi–Bellman equation associated with the stochastic control problem is given by the nonlinear PDE

$$\begin{aligned} \partial_t u(t,x) + \Delta u(t,x) + \min_m \left\{ 2\, m \cdot \nabla u(t,x) + \|m\|^2 \right\} &= 0, && (t,x) \in [0,T) \times \mathbb{R}^d, \\ u(T,x) &= g(x), && x \in \mathbb{R}^d. \end{aligned} \tag{35}$$

Note that this equation is purely deterministic. As easily verified, the minimum is attained at $m = -\nabla u$. Inserting this optimal control into the HJB equation (35) yields the semilinear PDE

$$\begin{aligned} \partial_t u(t,x) + \Delta u(t,x) - \|\nabla u(t,x)\|^2 &= 0, && (t,x) \in [0,T) \times \mathbb{R}^d, \\ u(T,x) &= g(x), && x \in \mathbb{R}^d. \end{aligned} \tag{36}$$

The formulation (34) reveals that the PDE solution u is the value function of a stochastic control problem, the control m_s is the negative gradient of the solution u which plays the role of a policy function in a reinforcement learning approach to solve the stochastic control problem [15,63,102,175]. This connection to stochastic control problems provided the original motivation for the deep BSDE method [43,68].

We solve this equation in dimension $d = 100$ for drift coefficient $\mu \equiv 0$, diffusion coefficient $\sigma \equiv \sqrt{2} I_{d\times d}$, reaction term $f(t,x,y,z) = -1/2\ \|z\|^2$ and final time $T = 1$ with prescribed data $g(x) = \log\left(1/2\ (1 + \|x\|^2)\right)$ using the algorithm described in Section 4.2 to approximate the solution value $u(0,x)$ for $x = 0 \in \mathbb{R}^d$. We note that the solution to this control problem can be obtained explicitly via a Cole–Hopf transformation, see for example [33], and is given by the formula $u(t,x) = -\log\left(\mathbb{E}\left[\exp\left(-g\left(x + \sqrt{2} W_{T-t}\right)\right)\right]\right)$. This can be used as a reference solution.

The results for 4 different experimental configurations are presented in Table 4. All experiments employ the Adam optimizer [95] with constant learning rate $\delta = 0.01$ as used in [43], the number of training epochs set to $n_{\text{epochs}} = 2000$ and batch size $n_{\text{batch}} = 64$. The setup in the second row labeled *Reference* uses the same configuration as employed in [43, Sec. 4.3], that is, $N = 20$ discrete time steps, and the network architecture as shown in (33) containing two stacks of layers of the form

$$\texttt{Dense} \rightsquigarrow \texttt{BN} \rightsquigarrow \texttt{ReLU} \tag{37}$$

with 110 neurons in each layer. The *Simple* configuration contains no such layer stack (37), and uses only $N = 1$ time step, which explains the fast computation. In setting *L=3*, we increased the number of hidden layer stacks (37) to three, the number of time steps to $N = 30$ and the number of neurons in each layer to 200. In setting *L=5*, we increased the number of hidden layer stacks (37) to five, the number of time steps to $N = 50$ and the number of neurons in each layer to 300.

The results in Table 4 suggests that for the solution of the linear-quadratic Gaussian control problem (36) all models display similar performance. It is surprising that even the *Simple* model taking less than 4 s total computation time provides essentially the same approximation quality as the more complex models. This is in line with the findings in [2] that it appears difficult to further decrease the relative errors using the proposed methodology. On the other hand, a decrease in relative approximation error when increasing the number of hidden layers was observed in another example given in [69]. The convergence behavior of this method seems to call for further research.

TABLE 5 Shown are the mean and standard deviations of $u_\theta(0,x)$ and the relative error $|u_\theta(0,x) - u^*|/u^*$, resp., with $u^* \approx 0.052802$ (taken from [43], calculated by a branching-diffusion method), as well as the mean computation time over 5 consecutive runs with randomly initialized parameters θ after $n_{\text{epochs}} = 4000$ training epochs

Experiment	Mean $u(0,x)$	Std.-dev. $u(0,x)$	Mean relative error	Std.-dev. relative error	Mean time (s)
Simple ($L=0$)	0.055816	4.55×10^{-5}	5.71×10^{-2}	8.62×10^{-4}	6.71
Reference ($L=2$)	0.052990	1.21×10^{-4}	3.63×10^{-3}	2.18×10^{-3}	157.54
$L=3$	0.052900	2.56×10^{-4}	4.02×10^{-3}	3.28×10^{-3}	308.45
$L=5$	0.052726	1.18×10^{-4}	2.27×10^{-3}	1.40×10^{-3}	637.18

4.4 | Example: Allen–Cahn equation

As a second example, we solve the Allen–Cahn equation with a double-well potential [43, Sec. 4.2] [50,69], that is, the semilinear reaction–diffusion equation

$$u_t(t,x) + \Delta u(t,x) + u(t,x) - u^3(t,x) = 0$$
$$u(T,x) = \left(2 + \frac{2}{5}\,\|x\|^2\right)^{-1}.$$

The results for the approximation of $u(0,x)$ for $x = 0 \in \mathbb{R}^d$ with $d = 100$ and $T = 1$ are displayed in Table 5. The training was carried out over $n_{\text{epochs}} = 4000$ epochs with the Adam optimizer [95] with a constant step size $\delta = 5\times10^{-4}$ for the same set of network configurations as used in Section 4.3 For this experiment, the difference between the *Simple* and more complex models is clearly visible. However, the *Simple* model yields again a rough approximation of the solution within only 7 s. Again, the decrease of the relative error is quite small for deeper and wider neural networks with more time steps, similar to our findings in Section 4.3. We note that the accompanying Jupyter notebook `DeepBSDE_Solver.ipynb` contains the Burgers-type PDE from [43, Sec. 4.5] as a third example.

4.5 | Summary and extensions

We have described the deep BSDE solver presented and developed in [43,69] for the solution of semilinear PDEs (26). Note, however, that the solver can also be used to solve BSDEs directly (without taking care of any PDE).

In [5], the deep BSDE solver considered in this section is extended to fully nonlinear PDEs of second-order. Here, neural networks are employed to approximate the second-order derivatives of u at a finite number of time steps, from which approximations of the gradients $\nabla u(t_n,\cdot)$ and the function values $u(t_n,\cdot)$ can be derived, similar to (32). The method relies on the connection between fully nonlinear second-order PDEs and second-order BSDEs [35].

The technique described in [7] is closely related and applies operator splitting techniques to derive a learning approach for the solution of parabolic PDEs in up to 10 000 spatial dimensions. In contrast to the deep BSDE method, however, the PDE solution at some discrete time snapshots is approximated by neural networks directly.

Another extension of the deep BSDE solver is considered in [31] where the authors employ a number of adaptations to the proposed methodology in order to improve the convergence properties of the algorithm, for example, by substituting the activation functions, removing some of the batch normalization layers and using only one instead of $N-2$ neural networks to approximate the scaled gradients of the solution $(\sigma^T\nabla u)(t_n,x)$ for $n=1,\dots,N-1$. Furthermore, residual connections are added and more elaborative *long short-term memory (LSTM)* neural networks are employed. Similarly, the authors in [53] consider the use of asymptotic expansion as prior knowledge in order to improve the accuracy and speed of convergence of the deep BSDE solver.

In [71], an extension based on a primal-dual solution method for BSDEs using neural networks and a dual formulation of stochastic control problems is discussed, see also [72]. An approach that uses the associated FBSDE to train a neural network to learn the solution of a semilinear PDE is discussed in [152].

5 | EXTENSIONS AND RELATED WORK

Beyond the three approaches discussed in detail in Sections 2 to 3, the rapidly developing discipline of scientific machine learning has brought forth a number of promising approaches for solving PDEs beyond the capabilities of conventional numerical methods. In this final section, we want to give a brief and necessarily incomplete overview over some recent developments.

Before we provide more references concerning neural network-based solution approaches for differential equations we list some results concerning general approximation properties of neural networks. Early work from the 1990s is now considered foundational, for example, [37,78,79,122,146]. Beginning around 2016, the spectacular successes of machine learning systems in computer vision, natural language processing and other areas prompted renewed efforts to establish a mathematically rigorous foundation for, in particular, deep feedforward neural networks [14,22,49,61,62,75,99,118,123,128,134,142-144,180,181]. We draw particular attention to a number of publications that rigorously establish that certain neural network architectures are theoretically able to overcome the curse of dimensionality for various linear and nonlinear PDEs, cf [8,18,65,81,82,84,85,92].

There are a number of criteria for classifying machine learning-based PDE solvers, among these mesh-free versus fixed mesh methods, stochastic versus deterministic methods or high-dimensional versus low-dimensional methods. While most of the investigated models can be considered mesh-free, we also mention some approaches that rely on an underlying and a priori known fixed mesh structure of the domain of the differential equations, cf [100,101,103,116,120,161,166].

A method termed *deep Galerkin method (DGM)* is proposed in [171] and is applied to the solution of nonlinear second-order parabolic equations. It is similar to the PINN approach discussed in Section 2 in that a neural network is used to approximate the PDE solution and the network is trained by minimizing a residual of the strong solution. The methods are aimed at high-dimensional problems, however, and a Monte Carlo method rather than automatic differentiation is used to compute second derivatives. A similar approach for solving high-dimensional random PDEs by training a neural network on the strong or weak residual is given in [129]. In [16] a deep neural network approximation to the solution of linear PDEs is constructed using the strong residual of the PDE as a loss function, similar to the PINN reviewed in Section 2.

In [40], an approach for solving a certain kind of high-dimensional first-order Hamilton–Jacobi equations is proposed based the Hopf formula [77] whose computational expense behaves polynomially in the spatial dimension. In subsequent work, first-order Hamilton–Jacobi equations in high dimension are considered in [38,39] based on classes of neural networks that exactly encode the viscosity solutions of these equations.

Further approaches based on the multilevel decomposition of Picard approximations and on full-history recursive multilevel Picard approximations [8,45-47,57,83] of type (26) have been successfully applied in high dimensions as well. Other directions of research that deal with high-dimensional PDEs are branching diffusion processes [73,74].

Another research area for the solution of PDEs is based on multi-scale *deep neural networks (DNNs)*, cf [30,104,110,176]. In a recent *Nature* publication multiscale DNNs are employed for diagnosing Alzheimer's disease [111]. Based on phase shift DNNs [29], considers the efficient solution of high-frequency wave equations.

In [2,80] the authors introduce and compare a number of neural network-based algorithms applied to stochastic control problems, nonlinear PDEs and BSDEs, incl. the example discussed in Section 4.3.

While the approaches discussed so far employ neural networks to learn mappings between finite-dimensional Euclidean spaces, the methodologies proposed in [20,105-107,112,131] aim to infer mappings between function spaces, known as *neural operators*. These mesh-free and infinite-dimensional operators require no prior knowledge of the underlying PDE but rely on a set of training data in the form of observations.

A general procedure based on data-driven machine learning to accelerate existing numerical methods for the solution of partial and ordinary differential equations is presented in [125].

A method to solve variational problems by means of scientific machine learning is proposed in [44], termed the *deep Ritz method* by the authors. The method relies on a reformulation of variational problems as an energy minimization problem. Boundary conditions are enforced weakly by the addition of a penalty term to the energy functional, for example

$$\min_{u \in H} \int_{\Omega} \left(\frac{1}{2} |\nabla u(x)| - u(x) \right) \mathrm{d}x + \beta \int_{\partial\Omega} u(s)^2 \mathrm{d}s \tag{38}$$

in the case of a Poisson problem with homogeneous boundary conditions, where H is a set of admissible functions and β is a penalty parameter used to enforce the boundary conditions. The proposed methodology relies on three key ideas:

the set of admissible functions H is represented by a (deep) neural network; the integrals in the energy functional (38) are approximated by Monte-Carlo sampling; and the neural network is trained through a stochastic gradient descent type algorithm on mini-batches. An extension to this approach is given in [108] termed the *deep Nitzsche method*.

Finally, we want to draw attention to the software package `NeuralPDE.jl` [150] written in the programming language Julia [19]. It is available at https://github.com/SciML/NeuralPDE.jl and features the solution of PDEs by PINNs, forward-backward SDEs for parabolic PDEs as well as deep-learning based solvers for optimal stopping time problems and Kolmogorov backward equations.

6 | CONCLUSION

The methods reviewed in this paper illustrate the versatility of machine learning-based algorithms for the solution of PDEs and represent the currently most promising approaches. While PINNs (Section 2) are, as of the writing of this survey, best suited for low-dimensional but complex nonlinear PDEs, the methods based on the Feynman–Kac theorem in Section 3 and BSDEs in Section 4 promise to extend current simulation capabilities when employed for high-dimensional linear and semi-linear parabolic problems in nonvariational form, for which classical approaches are infeasible due to the curse of dimensionality. As deep learning continues to grow rapidly in terms of methodological, theoretical and algorithmic advances, we believe that the field of machine learning-based solution methods of PDEs promises to remain an exciting research field in the years ahead.

ACKNOWLEDGMENTS

Funding statement: Open Access funding enabled and organized by Projekt DEAL. WOA institution: Technische Universitaet Chemnitz. Blended DEAL: Projekt DEAL.

REFERENCES

[1] M. Abadi et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.
[2] A. Bachouch et al., *Deep neural networks algorithms for stochastic control problems on finite horizon: Numerical applications*, arXiv:1812.05916, 2020.
[3] C. Basdevant et al., Spectral and finite difference solutions of the Burgers equation, Comput. & Fluids **14** (1986), 23–41.
[4] A. G. Baydin et al., Automatic differentiation in machine learning: A survey, J. Mach. Learn. Res. **18** (2018), 43 Paper No. 153.
[5] C. Beck and A. Jentzen, Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations, J. Nonlinear Sci. **29** (2019), 1563–1619.
[6] C. Beck et al., *Solving stochastic differential equations and Kolmogorov equations by means of deep learning*, arXiv:1806.00421, 2018.
[7] C. Beck et al., *Deep splitting method for parabolic PDEs*, arXiv:1907.03452, 2019.
[8] C. Beck et al., Overcoming the curse of dimensionality in the numerical approximation of Allen–Cahn partial differential equations via truncated full-history recursive multilevel Picard approximations, J. Numer. Math. **28** (2020), 197–222.
[9] C. Beck et al., *An overview on deep learning-based approximation methods for partial differential equations*, arXiv:2012.12348, 2020.
[10] R. Bellman, *Dynamic programming*, Princeton University Press, Princeton, NJ, 1957.
[11] J.-D. Benamou, B. D. Froese, and A. M. Oberman, Two numerical methods for the elliptic Monge-Ampère equation, M2AN. Math. Modell. Numer. Anal. **44** (2010), 737–758.
[12] C. Bender and R. Denk, A forward scheme for backward SDEs, Stochastic Process. Appl. **117** (2007), 1793–1812.
[13] C. Bender, N. Schweizer, and J. Zhuo, A primal–dual algorithm for BSDEs, Math. Finance **27** (2017), 866–901.
[14] P. Beneventano et al., *High-dimensional approximation spaces of artificial neural networks and applications to partial differential equations*, arXiv:2012.04326, 2020.
[15] Y. Bengio, *Learning deep architectures for AI*, Now Publishers Inc., 2009.
[16] J. Berg and K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing **317** (2018), 28–41.
[17] J. Berner, M. Dablander, and P. Grohs, Numerically solving parametric families of high-dimensional Kolmogorov partial differential equations via deep learning, Adv. Neural Inform. Process. Syst. **33** (2020).
[18] J. Berner, P. Grohs, and A. Jentzen, 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, SIAM J. Math. Data Sci. **2** (2020), 631–657.
[19] J. Bezanson et al., Julia: A fresh approach to numerical computing, SIAM Rev. **59** (2017), 65–98.
[20] K. Bhattacharya et al., *Model reduction and neural networks for parametric PDEs*, arXiv:2005.03180, 2020.
[21] J. Blechschmidt, R. Herzog, and M. Winkler, Error estimation for second-order partial differential equations in nonvariational form, Numer. Methods Partial Differential Equations (2020).
[22] H. Bolcskei et al., Optimal approximation with sparsely connected deep neural networks, SIAM J. Math. Data Sci. **1** (2019), 8–45.

[23] L. Bottou, F. E. Curtis, and J. Nocedal, Optimization methods for large-scale machine learning, SIAM Rev. **60** (2018), 223–311.
[24] B. Bouchard and N. Touzi, Discrete-time approximation and Monte-Carlo simulation of backward stochastic differential equations, Stochastic Process. Appl. **111** (2004), 175–206.
[25] M. J. Brennan and E. S. Schwartz, Finite difference methods and jump processes arising in the pricing of contingent claims: A synthesis, J. Financ. Quant. Anal. (1978), 461–474.
[26] S. C. Brenner and M. Neilan, Finite element approximations of the three dimensional Monge-Ampère equation, ESAIM. Math. Modell. Numer. Anal. **46** (2012), 979–1001.
[27] S. C. Brenner and L. R. Scott, *The mathematical theory of finite element methods*, in *Texts in Applied Mathematics*, 3rd ed., xviii+397, Vol **15**, Springer, New York, 2008. https://doi.org/10.1007/978-0-387-75934-0.
[28] D. Brigo, M. Morini, and A. Pallavicini, *Counterparty credit risk, collateral and funding: With pricing cases for all asset classes*, Vol **478**, John Wiley & Sons, 2013.
[29] W. Cai, X. Li, and L. Liu, A phase shift deep neural network for high frequency approximation and wave problems, SIAM J. Sci. Comput. **42** (2020), A3285–A3312.
[30] W. Cai and Z.-Q. J. Xu, *Multi-scale deep neural networks for solving high dimensional PDEs*, arXiv:1910.11710v1, 2019.
[31] Q. Chan-Wai-Nam, J. Mikael, and X. Warin, *Machine learning for semi linear PDEs*, 2018.
[32] J.-F. Chassagneux, Linear multistep schemes for BSDEs, SIAM J. Numer. Anal. **52** (2014), 2815–2836.
[33] J.-F. Chassagneux, A. Richou, et al., Numerical simulation of quadratic BSDEs, Ann. Appl. Probab. **26** (2016), 262–304.
[34] X. Chen, J. Duan, and G. E. Karniadakis, *Learning and meta-learning of stochastic advection–diffusion–reaction systems from sparse measurements*, arXiv:1910.09098, 2019.
[35] P. Cheridito et al., Second-order backward stochastic differential equations and fully nonlinear parabolic PDEs, Comm. Pure Appl. Math. **60** (2007), 1081–1110.
[36] S. Crépey, Bilateral counterparty risk under funding constraints—Part I: Pricing, Math. Finance **25** (2015), 1–22.
[37] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Systems **2** (1989), 303–314.
[38] J. Darbon, G. P. Langlois, and T. Meng, Overcoming the curse of dimensionality for some Hamilton–Jacobi partial differential equations via neural network architectures, Res. Math. Sci. **7** (2020), 20.
[39] J. Darbon and T. Meng, On some neural network architectures that can represent viscosity solutions of certain high dimensional Hamilton–Jacobi partial differential equations, J. Comput. Phys. **425** (2020), 109907.
[40] J. Darbon and S. Osher, Algorithms for overcoming the curse of dimensionality for certain Hamilton–Jacobi equations arising in control theory and elsewhere, Res. Math. Sci. **3**, 19–2016.
[41] R. W. Darling, E. Pardoux, et al., Backwards SDE with random terminal time and applications to semilinear elliptic PDE, Ann. Probab. **25** (1997), 1135–1159.
[42] J. Duchi, E. Hazan, and Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. **12** (2011).
[43] W. E. J. Han and A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Commun. Math. Stat. **5** (2017), 349–380.
[44] W. E. J. Han and B. Yu, The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. **6** (2018), 1–12.
[45] W. E. J. Han et al., *Linear scaling algorithms for solving high-dimensional nonlinear parabolic differential equations*, SAM Research Report 2017-43, ETH, Zurich, 2017.
[46] W. E. J. Han, et al., *Multilevel Picard iterations for solving smooth semilinear parabolic heat equations*, arXiv:1607.03295v4, 2019.
[47] W. E. J. Han et al., On multilevel Picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations, J. Sci. Comput. **79** (2019), 1534–1571.
[48] N. El Karoui, S. Peng, and M. C. Quenez, Backward stochastic differential equations in finance, Math. Finance **7** (1997), 1–71.
[49] D. Elbrächter et al., *DNN expression rate analysis of high-dimensional PDEs: Application to option pricing*, arXiv:1809.07669 (2018).
[50] H. Emmerich, *The diffuse interface approach in materials science: Thermodynamic concepts and applications of phase-field models*, Vol **73**, Springer Science & Business Media, 2003.
[51] X. Feng and M. Neilan, Mixed finite element methods for the fully nonlinear Monge-Ampère equation based on the vanishing moment method, SIAM J. Numer. Anal. **47** (2009), 1226–1250.
[52] W. H. Fleming and R. W. Rishel, *Deterministic and stochastic optimal control*, vii+222, in *Applications of Mathematics*, Springer-Verlag, Berlin, New York, 1975 No. 1.
[53] M. Fujii, A. Takahashi, and M. Takahashi, Asymptotic expansion as prior knowledge in deep learning method for high dimensional BSDEs, Asia-Pacific Financ. Mark. **26** (2019), 391–408.
[54] A. Genthon, The concept of velocity in the history of Brownian motion, Eur. Phys. J. H **45** (2020), 49–105.
[55] D. Gilbarg and N. S. Trudinger, *Elliptic partial differential equations of second order*, in *Classics in Mathematics*, xiv+517, Springer-Verlag, Berlin, 2001. Reprint of the 1998 edition.
[56] M. B. Giles, Multilevel Monte Carlo path simulation, Oper. Res. **56** (2008), 607–617.
[57] M. B. Giles, A. Jentzen, and T. Welti, *Generalised multilevel Picard approximations*, arXiv:1911.03188v1, 2019.
[58] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in *Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings*, 249–256.
[59] E. Gobet, *Monte-Carlo methods and stochastic processes: From linear to non-linear*, CRC Press, 2016.

[60] E. Gobet and P. Turkedjiev, Adaptive importance sampling in least-squares Monte Carlo algorithms for backward stochastic differential equations, Stochastic Process. Appl. **127** (2017), 1171–1203.

[61] L. Gonon and C. Schwab, *Deep ReLU network expression rates for option prices in high-dimensional, exponential Lévy models*, 2020-52, Seminar for Applied Mathematics, ETH, Zürich, Switzerland, 2020 https://www.sam.math.ethz.ch/sam_reports/reports_final/reports2020/2020-52.pdf.

[62] L. Gonon and C. Schwab, *Deep ReLU neural network approximation for stochastic differential equations with jumps*, 2021-08, Seminar for Applied Mathematics, ETH, Zürich, 2021.

[63] I. Goodfellow et al., *Deep Learning*, MIT Press, Cambridge, MA, 2016.

[64] C. Graham and D. Talay, *Stochastic simulation and Monte Carlo methods*, in *Stochastic Modelling and Applied Probability. Mathematical foundations of stochastic simulation*, Vol **68**, Springer, Heidelberg, 2013, xvi+260. https://doi.org/10.1007/978-3-642-39363-1.

[65] P. Grohs et al., *A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations*, arXiv:1809.02362, 2018.

[66] E. Haghighat et al., *A deep learning framework for solution and discovery in solid mechanics*, arXiv:2003.02751, 2020.

[67] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving ordinary differential equations. I: nonstiff problems*, in *Springer Series in Computational Mathematics*, Vol **8**, Springer-Verlag, Berlin, Heidelberg, 1993. https://doi.org/10.1007/978-3-540-78862-1.

[68] J. Han and W. E. J. Han, *Deep learning approximation for stochastic control problems*, arXiv:1611.07422, 2016.

[69] J. Han, A. Jentzen, and W. E. J. Han, Solving high-dimensional partial differential equations using deep learning, Proc. Nat. Acad. Sci. U.S.A. **115** (2018), 8505–8510.

[70] A. Heinlein, A. Klawonn, M. Lanser, and J. Weber, Combining machine learning and domain decomposition methods for the solution of partial differential equations—A review. GAMM-Mitteilungen. 2021;**44**:e202100001. https://doi.org/10.1002/gamm.202100001.

[71] P. Henry-Labordere, *Deep primal-dual algorithm for BSDEs: Applications of machine learning to CVA and IM*, Available at SSRN 3071506, 2017.

[72] P. Henry-Labordere, C. Litterer, and Z. Ren, A dual algorithm for stochastic control problems: Applications to uncertain volatility models and CVA, SIAM J. Financ. Math. **7** (2016), 159–182.

[73] P. Henry-Labordere, X. Tan, and N. Touzi, A numerical algorithm for a class of BSDEs via the branching process, Stochastic Process. Appl. **124** (2014), 1112–1140.

[74] P. Henry-Labordere et al., Branching diffusion representation of semilinear PDEs and Monte Carlo approximation, Ann. Inst. Henri Poincaré Probab. Stat. **55** (2019), 184–210.

[75] L. Herrmann, J. A. A. Opschoor, and C. Schwab, *Constructive deep ReLU neural network approximation*, in *2021-04, Seminar for Applied Mathematics*, ETH, Zürich, 2021.

[76] D. J. Higham, Stochastic ordinary differential equations in applied and computational mathematics, IMA J. Appl. Math. **76** (2011), 449–474.

[77] E. Hopf, Generalized solutions of non-linear equations of first order, J. Math. Mech. **14** (1965), 951–973.

[78] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. **4** (1991), 251–257.

[79] K. Hornik, M. Stinchcombe, and H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Netw. **3** (1990), 551–560.

[80] C. Huré et al., *Deep neural networks algorithms for stochastic control problems on finite horizon: Convergence analysis*, 2021.

[81] M. Hutzenthaler, A. Jentzen, and T. Kruse, *Overcoming the curse of dimensionality in the numerical approximation of parabolic partial differential equations with gradient-dependent nonlinearities*, arXiv:1912.02571, 2019.

[82] M. Hutzenthaler, A. Jentzen, and V. W. Wurstemberger, Overcoming the curse of dimensionality in the approximative pricing of financial derivatives with default risks, Electron. J. Probab. **25** (2020), 73. https://doi.org/10.1214/20-EJP423.

[83] M. Hutzenthaler and T. Kruse, Multilevel Picard approximations of high-dimensional semilinear parabolic differential equations with gradient-dependent nonlinearities, SIAM J. Numer. Anal. **58** (2020), 929–961.

[84] M. Hutzenthaler et al., Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations, Proc. R. Soc. A: Math. Phys. Eng. Sci. **476** (2020), 20190630.

[85] M. Hutzenthaler et al., A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations, SN Part. Differ. Equations Appl. **1** (2020), 1–34.

[86] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in *International conference on machine learning, PMLR*, 448–456.

[87] K. Itô, *Stochastic integral*, Vol **20**, Proceedings of the Imperial Academy, Tokyo, 1944, 519–524 http://projecteuclid.org/euclid.pja/1195572786.

[88] A. Jagtap and G. Karniadakis, Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. **28** (2020), 2002–2041.

[89] A. D. Jagtapa, K. Kawaguchib, and G. EmKarniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. **404** (2020), 109–136.

[90] A. D. Jagtap, K. Kawaguchi, and G. Em Karniadakis, Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks, Proc. R. Soc. A **476** (2020), 20200334.

[91] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Comput. Methods Appl. Mech. Engrg. **365** (2020), 113028.

[92] A. Jentzen, D. Salimova, and T. Welti, *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:1809.07321, 2018.

[93] M. Kac, On distributions of certain Wiener functionals, Trans. Amer. Math. Soc. **65** (1949), 1–13.

[94] I. Karatzas and S. Shreve, *Brownian motion and stochastic calculus*, Vol **113**, Springer, Berlin, 2014.

[95] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv:1412.6980 2014.

[96] P. E. Kloeden and E. Platen, *Numerical solution of stochastic differential equations*, in *Applications of Mathematics (New York)*, Vol **23**, Springer-Verlag, Berlin, 1992, xxxvi+632. https://doi.org/10.1007/978-3-662-12616-5.

[97] H. J. Kushner, Finite difference methods for the weak solutions of the Kolmogorov equations for the density of both diffusion and conditional diffusion processes, J. Math. Anal. Appl. **53** (1976), 251–265.

[98] H. J. Kushner, A survey of some applications of probability and stochastic control theory to finite difference methods for degenerate elliptic and parabolic equations, SIAM Rev. Publ. Soc. Indust. Appl. Math. **18** (1976), 545–577.

[99] F. Laakmann and P. Petersen, Efficient approximation of solutions of parametric linear transport equations by ReLU DNNs, Adv. Comput. Math. **47** (2021), 1–32.

[100] I. E. Lagaris, A. Likas, and D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. **9** (1998), 987–1000.

[101] I. E. Lagaris, A. Likas, and D. G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. **11** (2000), 1041–1049.

[102] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, Nature **521** (2015), 436–444.

[103] H. Lee and I. S. Kang, Neural algorithm for solving differential equations, J. Comput. Phys. **91** (1990), 110–131.

[104] X.-A. Li, A multi-scale DNN algorithm for nonlinear elliptic equations with multiple scales, Commun. Comput. Phys. **28** (2020), 1886–1906.

[105] Z. Li et al., *Fourier neural operator for parametric partial differential equations*, arXiv:2010.08895 2020.

[106] Z. Li et al., *Multipole graph neural operator for parametric partial differential equations*, arXiv:2006.09535, 2020.

[107] Z. Li et al., *Neural operator: Graph kernel network for partial differential equations*, arXiv:2003.03485, 2020.

[108] Y. Liao and P. Ming, *Deep Nitsche method: Deep Ritz method with essential boundary conditions*, arXiv:1912.01309, 2019.

[109] D. C. Liu and J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Programm. **45** (1989), 503–528.

[110] Z. Liu, W. Cai, and Z.-Q. J. Xu, Multi-scale deep neural network (MscaleDNN) for solving Poisson-Boltzmann equation in complex domains, Commun. Comput. Phys. **28** (2020), 1970–2001.

[111] D. Lu et al., Multimodal and multiscale deep neural networks for the early diagnosis of Alzheimer's disease using structural MR and FDG-PET images, Sci. Rep. **8** (2018), 1–13.

[112] L. Lu, P. Jin, and G. E. Karniadakis, *Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators*, arXiv:1910.03193, 2019.

[113] K. O. Lye, S. Mishra, and D. Ray, Deep learning observables in computational fluid dynamics, J. Comput. Phys. **410** (2020), 109339.

[114] A. L. Maas, A. Y. Hannun, and A. Y. Ng, *Rectifier nonlinearities improve neural network acoustic models*, in *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

[115] J. Magiera et al., Constraint-aware neural networks for Riemann problems, J. Comput. Phys. **409** (2020), 109345.

[116] A. Malek and R. S. Beidokhti, Numerical solution for high order differential equations using a hybrid neural network—Optimization method, Appl. Math. Comput. **183** (2006), 260–271.

[117] Z. Mao, A. D. Jagtap, and G. E. Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Engrg. **360** (2020), 112789 http://www.sciencedirect.com/science/article/pii/S0045782519306814.

[118] C. Marcati et al., *Exponential ReLU neural network approximation rates for point and edge singularities*, in *2020-65 (revised), Seminar for Applied Mathematics*, ETH, Zürich, 2020.

[119] G. Maruyama, Continuous Markov processes and stochastic equations, Rend. Circ. Mat. Palermo (2) **4** (1955), 48.

[120] A. J. Meade Jr. and A. A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks, Math. Comput. Modelling **19** (1994), 1–25.

[121] X. Meng and G. E. Karniadakis, A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems, J. Comput. Phys. **401** (2020), 109020.

[122] H. N. Mhaskar, Neural networks for optimal approximation of smooth and analytic functions, Neural Comput. **8** (1996), 164–177.

[123] H. N. Mhaskar and T. Poggio, Deep vs. shallow networks: An approximation theory perspective, Anal. Appl. **14** (2016), 829–848.

[124] G. Milstein, Weak approximation of a diffusion process in a bounded domain, Stochastics: Int. J. Probab. Stochastic Process. **62** (1997), 147–200.

[125] S. Mishra, *A machine learning framework for data driven acceleration of computations of differential equations*, arXiv:1807.09519, 2018.

[126] S. Mishra and R. Molinaro, *Estimates on the generalization error of physics informed neural networks (PINNs) for approximating PDEs II: A class of inverse problems*, arXiv:2007.01138, 2020.

[127] G. S. Misyris, A. Venzke, and S. Chatzivasileiadis, *Physics-informed neural networks for power systems*, in *2020 IEEE Power & Energy Society General Meeting (PESGM)*, IEEE, 1–5.

[128] H. Montanelli and Q. Du, New error bounds for deep ReLU networks using sparse grids, SIAM J. Math. Data Sci. **1** (2019), 78–92.

[129] M. A. Nabian and H. Meidani, A deep learning solution approach for high-dimensional random partial differential equations, Probab. Eng. Mech. **57** (2019), 14–25.

[130] M. Neilan, Convergence analysis of a finite element method for second order non-variational elliptic problems, J. Numer. Math. **25** (2017), 169–184.

[131] N. H. Nelsen and A. M. Stuart, *The random feature model for input-output maps between Banach spaces*, arXiv:2005.10224, 2020.

[132] V. M. Nguyen-Thanh, X. Zhuang, and T. Rabczuk, A deep energy method for finite deformation hyperelasticity, Eur. J.Mechanics-A/Solids **80** (2020), 103874.

[133] B. Øksendal, *Stochastic differential equations*, Springer, Berlin, Heidelberg, 2003. https://doi.org/10.1007/978-3-642-14394-6.

[134] J. A. A. Opschoor, P. C. Petersen, and C. Schwab, Deep ReLU networks and high-order finite element methods, Anal. Appl. **18** (2020), 715–770.

[135] G. Pang, L. Lu, and G. E. Karniadakis, fPINNs: Fractional physics-informed neural networks, SIAM J. Sci. Comput. **41** (2019), A2603–A2626.

[136] G. Pang, L. Yang, and G. E. Karniadakis, Neural-net-induced Gaussian process regression for function approximation and PDE solution, J. Comput. Phys. **384** (2019), 270–288.

[137] E. Pardoux and S. Peng, Adapted solution of a backward stochastic differential equation, Systems Control Lett. **14** (1990), 55–61.

[138] E. Pardoux and S. Peng, *Backward stochastic differential equations and quasilinear parabolic partial differential equations*, in *Stochastic partial differential equations and their applications*, Springer, 1992, 200–217.

[139] E. Pardoux and D. Talay, Discretization and simulation of stochastic differential equations, Acta Appl. Math. **3** (1985), 23–47.

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

[141] A. Paszke et al., *Automatic differentiation in PyTorch*, NIPS 2017 Workshop, 2017.

[142] D. Perekrestenko et al., *The universal approximation power of finite-width deep ReLU networks*, arXiv:1806.01528, 2018.

[143] P. Petersen, M. Raslan, and F. Voigtlaender, Topological properties of the set of functions generated by neural networks of fixed size, Found. Comput. Math. (2020), 1–70.

[144] P. Petersen and F. Voigtlaender, Optimal approximation of piecewise smooth functions using deep ReLU neural networks, Neural Netw. **108** (2018), 296–330.

[145] H. Pham, *Continuous-time stochastic control and optimization with financial applications Stochastic Modelling and Applied Probability*, vol. **61**, xviii+232, Springer-Verlag, Berlin, 2009, doi: 10.1007/978-3-540-89500-8.

[146] A. Pinkus, Approximation theory of the MLP model, Acta Numer. **8** (1999), 143–195.

[147] W. B. Powell, What you should know about approximate dynamic programming, Naval Res. Logist. **56** (2009), 239–249.

[148] P. E. Protter, *Stochastic integration and differential equations. Stochastic modelling and applied probability*, 2nd ed., Vol **21**, Springer-Verlag, Berlin, 2005. https://doi.org/10.1007/978-3-662-10061-5 xiv+419, Version 2.1, Corrected third printing.

[149] D. C. Psichogios and L. H. Ungar, A hybrid neural network-first principles approach to process modeling, AIChE J. **38** (1992), 1499–1511.

[150] C. Rackauckas and Q. Nie, DifferentialEquations.jl—A performant and feature-rich ecosystem for solving differential equations in Julia, J. Open Res. Softw. **5** (2017).

[151] M. Raissi, *Deep hidden physics models: Deep learning of nonlinear partial differential equations*, arXiv:1801.06637, 2018.

[152] M. Raissi, *Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations*, arXiv:1804.07010, 2018.

[153] M. Raissi and G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, J. Comput. Phys. **357** (2018), 125–141.

[154] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, J. Comput. Phys. **348** (2017), 683–693.

[155] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Numerical Gaussian processes for time-dependent and non-linear partial differential equations*, arXiv:1703.10230, 2017.

[156] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations*, arXiv:1711.10561, 2017.

[157] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics informed deep learning (Part II): Data-driven discovery of nonlinear partial differential equations*, arXiv:1711.10566, 2017.

[158] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Numerical Gaussian processes for time-dependent and nonlinear partial differential equations, SIAM J. Sci. Comput. **40** (2018), A172–A198.

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

[160] M. Raissi, A. Yazdani, and G. E. Karniadakis, *Hidden fluid mechanics: A Navier–Stokes informed deep learning framework for assimilating flow visualization data*, arXiv:1808.04327, 2018.

[161] P. Ramuhalli, L. Udpa, and S. S. Udpa, Finite-element neural networks for solving differential equations, IEEE Trans. Neural Netw. **16** (2005), 1381–1392.

[162] C. Rao, H. Sun, and Y. Liu, *Physics informed deep learning for computational elastodynamics without labeled data*, arXiv:2006.08472, 2020.

[163] C. Rao, H. Sun, and Y. Liu, Physics-informed deep learning for incompressible laminar flows, Theor. Appl. Mech. Lett. **10** (2020), 207–212.

[164] C. E. Rasmussen, *Gaussian processes in machine learning. Summer school on machine learning*, Springer, 63–71.

[165] D. Revuz and M. Yor, *Continuous martingales and Brownian motion, Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*, vol. **293**, 3rd ed.. Springer-Verlag, Berlin, 1999, https://doi.org/10.1007/978-3-662-06400-9, xiv+602.
[166] K. Rudd, *Solving partial differential equations using artificial neural networks*, Ph.D. thesis, Duke University, 2013.
[167] S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv:1609.04747, 2016.
[168] S. H. Rudy et al., Data-driven discovery of partial differential equations, Sci. Adv. **3** (2017).
[169] W. Schachermayer, *Introduction: Bachelier's thesis from 1900*, in *Lectures on Probability Theory and Statistics, Lecture Notes in Mathematics*, Vol **1816**, S. Albeverio, W. Schachermayer, and M. Talagrand, Eds., Springer-Verlag, Berlin, Heidelberg, 2003, 111–126. https://doi.org/10.1007/3-540-44922-1_7.
[170] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*, Cambridge University Press, Cambridge, MA, 2014.
[171] J. Sirignano and K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, J. Comput. Phys. **375** (2018), 1339–1364.
[172] I. Smears and E. Süli, Discontinuous Galerkin finite element approximation of Hamilton-Jacobi-Bellman equations with Cordes coefficients, SIAM J. Numer. Anal. **52** (2014), 993–1016.
[173] J. M. Steele, *Stochastic calculus and financial applications*, Vol **45**, Springer Science & Business Media, 2012.
[174] M. Stein, Large sample properties of simulations using Latin hypercube sampling, Technometrics. J. Stat. Phys. Chem. Eng. Sci. **29** (1987), 143–151.
[175] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT press, 2018.
[176] B. Wang, Multi-scale deep neural network (MscaleDNN) methods for oscillatory Stokes flows in complex domains, Commun. Comput. Phys. **28** (2020), 2139–2157.
[177] H. Wessels, C. Weißenfels, and P. Wriggers, The neural particle method—An updated Lagrangian physics informed neural network for computational fluid dynamics, Comput. Methods Appl. Mech. Engrg. **368** (2020), 113127.
[178] J. Yang, G. Zhang, and W. Zhao, A first-order numerical scheme for forward-backward stochastic differential equations in bounded domains, J. Comput. Math. **36** (2018), 237–258.
[179] L. Yang, D. Zhang, and G. E. Karniadakis, Physics-informed generative adversarial networks for stochastic differential equations, SIAM J. Sci. Comput. **42** (2020), A292–A317.
[180] D. Yarotsky, Error bounds for approximations with deep ReLU networks, Neural Netw. **94** (2017), 103–114.
[181] D. Yarotsky, *Universal approximations of invariant maps by neural networks*, arXiv:1804.10306, 2018.
[182] J. Yong and X. Y. Zhou, *Stochastic controls: Hamiltonian systems and HJB equations*, Vol **43**, Springer Science & Business Media, 1999.
[183] Y. Zhu et al., Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. **394** (2019), 56–81.

How to cite this article: Blechschmidt J, Ernst OG. Three ways to solve partial differential equations with neural networks — A review. *GAMM-Mitteilungen*. 2021;44:e202100006. https://doi.org/10.1002/gamm.202100006