Journal of Computational Physics 375 (2018) 1339–1364

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

DGM: A deep learning algorithm for solving partial differential equations ☆,☆☆

Justin Sirignano [a,*], Konstantinos Spiliopoulos [b]

[a] University of Illinois at Urbana Champaign, Urbana, United States of America
[b] Department of Mathematics and Statistics, Boston University, Boston, United States of America

ARTICLE INFO

Article history:
Received 2 February 2018
Received in revised form 10 August 2018
Accepted 20 August 2018
Available online 24 August 2018

Keywords:
Partial differential equations
Machine learning
Deep learning
High-dimensional partial differential equations

ABSTRACT

High-dimensional PDEs have been a longstanding computational challenge. We propose to solve high-dimensional PDEs by approximating the solution with a deep neural network which is trained to satisfy the differential operator, initial condition, and boundary conditions. Our algorithm is meshfree, which is key since meshes become infeasible in higher dimensions. Instead of forming a mesh, the neural network is trained on batches of randomly sampled time and space points. The algorithm is tested on a class of high-dimensional free boundary PDEs, which we are able to accurately solve in up to 200 dimensions. The algorithm is also tested on a high-dimensional Hamilton–Jacobi–Bellman PDE and Burgers' equation. The deep learning algorithm approximates the general solution to the Burgers' equation for a continuum of different boundary conditions and physical conditions (which can be viewed as a high-dimensional space). We call the algorithm a "Deep Galerkin Method (DGM)" since it is similar in spirit to Galerkin methods, with the solution approximated by a neural network instead of a linear combination of basis functions. In addition, we prove a theorem regarding the approximation power of neural networks for a class of quasilinear parabolic PDEs.

1. Deep learning and high-dimensional PDEs

High-dimensional partial differential equations (PDEs) are used in physics, engineering, and finance. Their numerical solution has been a longstanding challenge. Finite difference methods become infeasible in higher dimensions due to the explosion in the number of grid points and the demand for reduced time step size. If there are d space dimensions and 1 time dimension, the mesh is of size $\mathcal{O}^{d+1}$. This quickly becomes computationally intractable when the dimension d becomes even moderately large. We propose to solve high-dimensional PDEs using a meshfree deep learning algorithm. The method is similar in spirit to the Galerkin method, but with several key changes using ideas from machine learning. The Galerkin method is a widely-used computational method which seeks a reduced-form solution to a PDE as a linear combination

☆ The authors thank seminar participants at the JP Morgan Machine Learning and AI Forum seminar, the Imperial College London Applied Mathematics and Mathematical Physics seminar, the Department of Applied Mathematics at the University of Colorado Boulder, Princeton University, and Northwestern University for their comments. The authors would also like to thank participants at the 2017 INFORMS Applied Probability Conference, the 2017 Greek Stochastics Conference, and the 2018 SIAM Annual Meeting for their comments.

☆☆ Research of K.S. supported in part by the National Science Foundation (DMS 1550918). Computations for this paper were performed using the Blue Waters supercomputer grant "Distributed Learning with Neural Networks".

* Corresponding author.
E-mail addresses: jasirign@illinois.edu (J. Sirignano), kspiliop@math.bu.edu (K. Spiliopoulos).

https://doi.org/10.1016/j.jcp.2018.08.029
0021-9991/

of basis functions. The deep learning algorithm, or "Deep Galerkin Method" (DGM), uses a deep neural network instead of a linear combination of basis functions. The deep neural network is trained to satisfy the differential operator, initial condition, and boundary conditions using stochastic gradient descent at randomly sampled spatial points. By randomly sampling spatial points, we avoid the need to form a mesh (which is infeasible in higher dimensions) and instead convert the PDE problem into a machine learning problem.

DGM is a natural merger of Galerkin methods and machine learning. The algorithm in principle is straightforward; see Section 2. Promising numerical results are presented later in Section 4 for a class of high-dimensional free boundary PDEs. We also accurately solve a high-dimensional Hamilton–Jacobi–Bellman PDE in Section 5 and Burger's equation in Section 6. DGM converts the computational cost of finite difference to a more convenient form: instead of a huge mesh of $\mathcal{O}^{d+1}$ (which is infeasible to handle), many batches of random spatial points are generated. Although the total number of spatial points could be vast, the algorithm can process the spatial points sequentially without harming the convergence rate.

Deep learning has revolutionized fields such as image, text, and speech recognition. These fields require statistical approaches which can model nonlinear functions of high-dimensional inputs. Deep learning, which uses multi-layer neural networks (i.e., "deep neural networks"), has proven very effective in practice for such tasks. A multi-layer neural network is essentially a "stack" of nonlinear operations where each operation is prescribed by certain parameters that must be estimated from data. Performance in practice can strongly depend upon the specific form of the neural network architecture and the training algorithms which are used. The design of neural network architectures and training methods has been the focus of intense research over the past decade. Given the success of deep learning, there is also growing interest in applying it to a range of other areas in science and engineering (see Section 1.2 for some examples).

Evaluating the accuracy of the deep learning algorithm is not straightforward. PDEs with semi-analytic solutions may not be sufficiently challenging. (After all, the semi-analytic solution exists since the PDE can be transformed into a lower-dimensional equation.) It cannot be benchmarked against traditional finite difference (which fails in high dimensions). We test the deep learning algorithm on a class of high-dimensional free boundary PDEs which have the special property that error bounds can be calculated for any approximate solution. This provides a unique opportunity to evaluate the accuracy of the deep learning algorithm on a class of high-dimensional PDEs *with no semi-analytic solutions*.

This class of high-dimensional free boundary PDEs also has important applications in finance, where it used to price American options. An American option is a financial derivative on a portfolio of stocks. The number of space dimensions in the PDE equals the number of stocks in the portfolio. Financial institutions are interested in pricing options on portfolios ranging from dozens to even hundreds of stocks [43]. Therefore, there is a significant need for numerical methods to accurately solve high-dimensional free boundary PDEs.

We also test the deep learning algorithm on a high-dimensional Hamilton–Jacobi–Bellman PDE with accurate results. We consider a high-dimensional Hamilton–Jacobi–Bellman PDE motivated by the problem of optimally controlling a stochastic heat equation.

Finally, it is often of interest to find the solution of a PDE over a range of problem setups (e.g., different physical conditions and boundary conditions). For example, this may be useful for the design of engineering systems or uncertainty quantification. The problem setup space may be high-dimensional and therefore may require solving many PDEs for many different problem setups, which can be computationally expensive. We use our deep learning algorithm to approximate the general solution to the Burgers' equation for different boundary conditions, initial conditions, and physical conditions.

In the remainder of the Introduction, we provide an overview of our results regarding the approximation power of neural networks for quasilinear parabolic PDEs (Section 1.1), and relevant literature (Section 1.2). The deep learning algorithm for solving PDEs is presented in Section 2. An efficient scheme for evaluating the diffusion operator is developed in Section 3. Numerical analysis of the algorithm is presented in Sections 4, 5, and 6. We implement and test the algorithm on a class of high-dimensional free boundary PDEs in up to 200 dimensions. The theorem and proof for the approximation of PDE solutions with neural networks is presented in Section 7. Conclusions are in Section 8. For readability purposes, proofs from Section 7 have been collected in Appendix A.

1.1. Approximation power of neural networks for PDEs

We also prove a theorem regarding the approximation power of neural networks for a class of quasilinear parabolic PDEs. Consider the potentially nonlinear PDE

$$\begin{aligned} &\partial_t u(t,x) + \mathcal{L}u(t,x) = 0, \qquad (t,x) \in [0,T] \times \Omega \\ &u(0,x) = u_0(x), \qquad x \in \Omega \\ &u(t,x) = g(t,x), \qquad x \in [0,T] \times \partial\Omega, \end{aligned} \tag{1.1}$$

where $\partial\Omega$ is the boundary of the domain Ω. The solution $u(t,x)$ is of course unknown, but an approximate solution $f(t,x)$ can be found by minimizing the L^2 error

$$J(f) = \|\partial_t f + \mathcal{L}f\|^2_{2,[0,T]\times\Omega} + \|f - g\|^2_{2,[0,T]\times\partial\Omega} + \|f(0,\cdot) - u_0\|^2_{2,\Omega}.$$

The error function $J(f)$ measures how well the approximate solution f satisfies the differential operator, boundary condition, and initial condition. Note that no knowledge of the actual solution u is assumed; $J(f)$ can be directly calculated

from the PDE (1.1) for any approximation f. The goal is to construct functions f for which $J(f)$ is as close to 0 as possible. Define $\mathfrak{C}^n$ as the class of neural networks with a single hidden layer and n hidden units.[1] Let f^n be a neural network with n hidden units which minimizes $J(f)$. We prove that, under certain conditions,

$$\text{there exists } f^n \in \mathfrak{C}^n \text{ such that } J(f^n) \to 0, \text{ as } n \to \infty, \text{ and}$$
$$f^n \to u \quad \text{as} \quad n \to \infty,$$

strongly in, $L^\rho([0,T]\times\Omega)$, with $\rho < 2$, for a class of quasilinear parabolic PDEs; see subsection 7.2 and Theorem 7.3 therein for the precise statement. That is, the neural network will converge in L^ρ, $\rho < 2$ to the solution of the PDE as the number of hidden units tends to infinity. The precise statement of the theorem and its proof are presented in Section 7. The proof requires the joint analysis of the approximation power of neural networks as well as the continuity properties of partial differential equations. Note that $J(f^n) \to 0$ does not necessarily imply that $f^n \to u$, given that we only have L^2 control on the approximation error. First, we prove that $J(f^n) \to 0$ as $n \to \infty$. We then establish that each neural network $\{f^n\}_{n=1}^\infty$ satisfies a PDE with a source term $h^n(t,x)$. We are then able to prove, under certain conditions, the convergence of $f^n \to u$ as $n \to \infty$ in $L^\rho([0,T]\times\Omega)$, for $\rho < 2$, using the smoothness of the neural network approximations and compactness arguments.

Theorem 7.3 establishes the approximation power of neural networks for solving PDEs (at least within a class of quasilinear parabolic PDEs); however, directly minimizing $J(f)$ is not computationally tractable since it involves high-dimensional integrals. The DGM algorithm minimizes $J(f)$ using a meshfree approach; see Section 2.

1.2. Relevant literature

Solving PDEs with a neural network as an approximation is a natural idea, and has been considered in various forms previously. [29], [30], [46], [31], and [35] propose to use neural networks to solve PDEs and ODEs. These papers estimate neural network solutions on an a priori fixed mesh. This paper proposes using *deep* neural networks and is *meshfree*, which is key to solving high-dimensional PDEs.

In particular, this paper explores several new innovations. First, we focus on high-dimensional PDEs and apply deep learning advances of the past decade to this problem (deep neural networks instead of shallow neural networks, improved optimization methods for neural networks, etc.). Algorithms for high-dimensional free boundary PDEs are developed, efficiently implemented, and tested. In particular, we develop an iterative method to address the free boundary. Secondly, to avoid ever forming a mesh, we sample a sequence of random spatial points. This produces a meshfree method, which is essential for high-dimensional PDEs. Thirdly, the algorithm incorporates a new computational scheme for the efficient computation of neural network gradients arising from the second derivatives of high-dimensional PDEs.

Recently, [41,42] develop physics informed deep learning models. They estimate deep neural network models which merge data observations with PDE models. This allows for the estimation of physical models from limited data by leveraging a priori knowledge that the physical dynamics should obey a class of PDEs. Their approach solves PDEs in one and two spatial dimensions using deep neural networks. [32] uses a deep neural network to model the Reynolds stresses in a Reynolds-averaged Navier–Stokes (RANS) model. RANS is a reduced-order model for turbulence in fluid dynamics. [15,2] have also recently developed a scheme for solving a class of quasilinear PDEs which can be represented as forward–backward stochastic differential equations (FBSDEs) and [16] further develops the algorithm. The algorithm developed in [15,2,16] focuses on computing the value of the PDE solution at a single point. The algorithm that we present here is different; in particular, it does not rely on the availability of FBSDE representations and yields the entire solution of the PDE across all time and space. In addition, the deep neural network architecture that we use, which is different from the ones used in [15,2], seems to be able to recover accurately the entire solution (at least for the equations that we studied). [49] use a convolutional neural network to solve a large sparse linear system which is required in the numerical solution of the Navier–Stokes PDE. In addition, [9] has recently developed a novel partial differential equation approach to optimize deep neural networks.

[33] developed an algorithm for the solution of a discrete-time version of a class of free boundary PDEs. Their algorithm, commonly called the "Longstaff–Schwartz method", uses dynamic programming and approximates the solution using a separate function approximator at each discrete time (typically a linear combination of basis functions). Our algorithm directly solves the PDE, and uses a single function approximator for all space and all time. The Longstaff–Schwartz algorithm has been further analyzed by [45], [23], and others. Sparse grid methods have also been used to solve high-dimensional PDEs; see [43], [44], [22], [6], and [7].

In regard to general results on the approximation power of neural networks we refer the interested reader to classical works [11,25,26,39] and we also mention the recent work by [38], where the authors study the necessary and sufficient complexity of ReLU neural networks that is required for approximating classifier functions in the mean square sense.

[1] A neural network with a single hidden layer and n hidden units is a function of the form $\mathfrak{C}^n = \{h(t,x) : \mathbb{R}^{1+d} \mapsto \mathbb{R} : h(t,x) = \sum_{i=1}^n \beta_i \psi(\alpha_{1,i} t + \sum_{j=1}^d \alpha_{j,i} x_j + c_j)\}$ where $\Psi : \mathbb{R} \to \mathbb{R}$ is a nonlinear "activation" function such as a sigmoid or tanh function.

2. Algorithm

Consider a parabolic PDE with d spatial dimensions:

$$\begin{aligned}&\frac{\partial u}{\partial t}(t,x)+\mathcal{L}u(t,x)=0, \quad (t,x)\in[0,T]\times\Omega,\\&u(t=0,x)=u_0(x),\\&u(t,x)=g(t,x), \quad x\in\partial\Omega,\end{aligned} \tag{2.1}$$

where $x\in\Omega\subset\mathbb{R}^d$. The DGM algorithm approximates $u(t,x)$ with a deep neural network $f(t,x;\theta)$ where $\theta\in\mathbb{R}^K$ are the neural network's parameters. Note that the differential operators $\frac{\partial f}{\partial t}(t,x;\theta)$ and $\mathcal{L}f(t,x;\theta)$ can be calculated analytically. Construct the objective function:

$$J(f)=\left\|\frac{\partial f}{\partial t}(t,x;\theta)+\mathcal{L}f(t,x;\theta)\right\|^2_{[0,T]\times\Omega,\nu_1}+\|f(t,x;\theta)-g(t,x)\|^2_{[0,T]\times\partial\Omega,\nu_2}+\|f(0,x;\theta)-u_0(x)\|^2_{\Omega,\nu_3}.$$

Here, $\|f(y)\|^2_{\mathcal{Y},\nu}=\int_{\mathcal{Y}}|f(y)|^2\,\nu(y)dy$ where $\nu(y)$ is a positive probability density on $y\in\mathcal{Y}$. $J(f)$ measures how well the function $f(t,x;\theta)$ satisfies the PDE differential operator, boundary conditions, and initial condition. If $J(f)=0$, then $f(t,x;\theta)$ is a solution to the PDE (2.1).

The goal is to find a set of parameters θ such that the function $f(t,x;\theta)$ minimizes the error $J(f)$. If the error $J(f)$ is small, then $f(t,x;\theta)$ will closely satisfy the PDE differential operator, boundary conditions, and initial condition. Therefore, a θ which minimizes $J(f(\cdot;\theta))$ produces a reduced-form model $f(t,x;\theta)$ which approximates the PDE solution $u(t,x)$.

Estimating θ by directly minimizing $J(f)$ is infeasible when the dimension d is large since the integral over Ω is computationally intractable. However, borrowing a machine learning approach, one can instead minimize $J(f)$ using stochastic gradient descent on a sequence of time and space points *drawn at random* from Ω and $\partial\Omega$. This avoids ever forming a mesh.

The DGM algorithm is:

1. Generate random points (t_n,x_n) from $[0,T]\times\Omega$ and (τ_n,z_n) from $[0,T]\times\partial\Omega$ according to respective probability densities ν_1 and ν_2. Also, draw the random point w_n from Ω with probability density ν_3.
2. Calculate the squared error $G(\theta_n,s_n)$ at the randomly sampled points $s_n=\{(t_n,x_n),(\tau_n,z_n),w_n\}$ where:

$$G(\theta_n,s_n)=\left(\frac{\partial f}{\partial t}(t_n,x_n;\theta_n)+\mathcal{L}f(t_n,x_n;\theta_n)\right)^2+\left(f(\tau_n,z_n;\theta_n)-g(\tau_n,z_n)\right)^2+\left(f(0,w_n;\theta_n)-u_0(w_n)\right)^2.$$

3. Take a descent step at the random point s_n:

$$\theta_{n+1}=\theta_n-\alpha_n\nabla_\theta G(\theta_n,s_n)$$

4. Repeat until convergence criterion is satisfied.

The "learning rate" α_n decreases with n. The steps $\nabla_\theta G(\theta_n,s_n)$ are unbiased estimates of $\nabla_\theta J(f(\cdot;\theta_n))$:

$$\mathbb{E}\left[\nabla_\theta G(\theta_n,s_n)\middle|\theta_n\right]=\nabla_\theta J(f(\cdot;\theta_n)).$$

Therefore, the stochastic gradient descent algorithm will on average take steps in a *descent direction* for the objective function J. A descent direction means that the objective function decreases after an iteration (i.e., $J(f(\cdot;\theta_{n+1}))<J(f(\cdot;\theta_n))$), and θ_{n+1} is therefore a better parameter estimate than θ_n.

Under (relatively mild) technical conditions (see [3]), the algorithm θ_n will converge to a critical point of the objective function $J(f(\cdot;\theta))$ as $n\to\infty$:

$$\lim_{n\to\infty}\|\nabla_\theta J(f(\cdot;\theta_n))\|=0.$$

It's important to note that θ_n may only converge to a local minimum when $f(t,x;\theta)$ is non-convex. This is generally true for non-convex optimization and is not specific to this paper's algorithm. In particular, deep neural networks are non-convex. Therefore, it is well known that stochastic gradient descent may only converge to a local minimum (and not a global minimum) for a neural network. Nevertheless, stochastic gradient descent has proven very effective in practice and is the fundamental building block of nearly all approaches for training deep learning models.

3. A Monte Carlo method for fast computation of second derivatives

This section describes a modified algorithm which may be more computationally efficient in some cases. The term $\mathcal{L}f(t,x;\theta)$ contains second derivatives $\frac{\partial^2 f}{\partial x_i x_j}(t,x;\theta)$ which may be expensive to compute in higher dimensions. For instance, 20,000 second derivatives must be calculated in $d = 200$ dimensions.

The complicated architectures of neural networks can make it computationally costly to calculate the second derivatives (for example, see the neural network architecture (4.2)). The computational cost for calculating second derivatives (in both total arithmetic operations and memory) is $\mathcal{O}(d^2 \times N)$ where d is the spatial dimension of x and N is the batch size. In comparison, the computational cost for calculating first derivatives is $\mathcal{O}(d \times N)$. The cost associated with the second derivatives is further increased since we actually need the third-order derivatives $\nabla_\theta \frac{\partial^2 f}{\partial x^2}(t,x;\theta)$ for the stochastic gradient descent algorithm. Instead of directly calculating these second derivatives, we approximate the second derivatives using a Monte Carlo method.

Suppose the sum of the second derivatives in $\mathcal{L}f(t,x,;\theta)$ is of the form $\frac{1}{2}\sum_{i,j=1}^d \rho_{i,j}\sigma_i(x)\sigma_j(x)\frac{\partial^2 f}{\partial x_i x_j}(t,x;\theta)$, assume $[\rho_{i,j}]_{i,j=1}^d$ is a positive definite matrix, and define $\sigma(x) = \Big(\sigma_1(x),\ldots,\sigma_d(x)\Big)$. For example, such PDEs arise when considering expectations of functions of stochastic differential equations, where the $\sigma(x)$ represents the diffusion coefficient. See equation (4.1) and the corresponding discussion. A generalization of the algorithm in this section to second derivatives with nonlinear coefficient dependence on $u(t,x)$ is also possible. Then,

$$\sum_{i,j=1}^d \rho_{i,j}\sigma_i(x)\sigma_j(x)\frac{\partial^2 f}{\partial x_i x_j}(t,x;\theta) = \lim_{\Delta\to 0}\mathbb{E}\Bigg[\sum_{i=1}^d \frac{\frac{\partial f}{\partial x_i}(t,x+\sigma(x)W_\Delta;\theta) - \frac{\partial f}{\partial x_i}(t,x;\theta)}{\Delta}\sigma_i(x)W_\Delta^i\Bigg], \tag{3.1}$$

where $W_t \in \mathbb{R}^d$ is a Brownian motion and $\Delta \in \mathbb{R}_+$ is the step-size. The convergence rate for (3.1) is $\mathcal{O}(\sqrt{\Delta})$.[2] Define:

$$G_1(\theta_n,s_n) := \left(\frac{\partial f}{\partial t}(t_n,x_n;\theta_n) + \mathcal{L}f(t_n,x_n;\theta_n)\right)^2,$$

$$G_2(\theta_n,s_n) := \Big(f(\tau_n,z_n;\theta_n) - g(\tau_n,z_n)\Big)^2,$$

$$G_3(\theta_n,s_n) := \Big(f(0,w_n;\theta_n) - u_0(w_n)\Big)^2,$$

$$G(\theta_n,s_n) := G_1(\theta_n,s_n) + G_2(\theta_n,s_n) + G_3(\theta_n,s_n).$$

The DGM algorithm use the gradient $\nabla_\theta G_1(\theta_n,s_n)$, which requires the calculation of the second derivative terms in $\mathcal{L}f(t_n,x_n;\theta_n)$. Define the first derivative operators as

$$\mathcal{L}_1 f(t_n,x_n;\theta_n) := \mathcal{L}f(t_n,x_n;\theta_n) - \frac{1}{2}\sum_{i,j=1}^d \rho_{i,j}\sigma_i(x_n)\sigma_j(x_n)\frac{\partial^2 f}{\partial x_i x_j}(t_n,x_n;\theta).$$

Using (3.1), $\nabla_\theta G_1$ is approximated as $\tilde{G}_1$ with a fixed constant $\Delta > 0$:

$$\tilde{G}_1(\theta_n,s_n) := 2\Bigg(\frac{\partial f}{\partial t}(t_n,x_n;\theta_n) + \mathcal{L}_1 f(t_n,x_n;\theta_n) + \frac{1}{2}\sum_{i=1}^d \frac{\frac{\partial f}{\partial x_i}(t,x_n+\sigma(x_n)W_\Delta;\theta) - \frac{\partial f}{\partial x_i}(t,x_n;\theta)}{\Delta}\sigma_i(x_n)W_\Delta^i\Bigg)$$
$$\times\ \nabla_\theta\Bigg(\frac{\partial f}{\partial t}(t_n,x_n;\theta_n) + \mathcal{L}_1 f(t_n,x_n;\theta_n) + \frac{1}{2}\sum_{i=1}^d \frac{\frac{\partial f}{\partial x_i}(t,x_n+\sigma(x_n)\tilde{W}_\Delta;\theta) - \frac{\partial f}{\partial x_i}(t,x_n;\theta)}{\Delta}\sigma_i(x_n)\tilde{W}_\Delta^i\Bigg),$$

where W_Δ is a d-dimensional normal random variable with $\mathbb{E}[W_\Delta] = 0$ and $\mathrm{Cov}[(W_\Delta)_i,(W_\Delta)_j] = \rho_{i,j}\Delta$. $\tilde{W}_\Delta$ has the same distribution as W_Δ. W_Δ and $\tilde{W}_\Delta$ are independent. $\tilde{G}_1(\theta_n,s_n)$ is a Monte Carlo approximation of $\nabla_\theta G_1(\theta_n,s_n)$. $\tilde{G}_1(\theta_n,s_n)$ has $\mathcal{O}(\sqrt{\Delta})$ bias as an approximation for $\nabla_\theta G_1(\theta_n,s_n)$. This approximation error can be further improved via the following scheme using "antithetic variates":

[2] Let f be a three-times differentiable function in x with bounded third-order derivatives in x. Then, it directly follows from a Taylor expansion that $\left|\sum_{i,j=1}^d \rho_{i,j}\sigma_i(x)\sigma_j(x)\frac{\partial^2 f}{\partial x_i x_j}(t,x;\theta) - \mathbb{E}\left[\sum_{i=1}^d \frac{\frac{\partial f}{\partial x_i}(t,x+\sigma(x)W_\Delta;\theta) - \frac{\partial f}{\partial x_i}(t,x;\theta)}{\Delta}\sigma_i(x)W_\Delta^i\right]\right| \le C(x)\sqrt{\Delta}$. The constant $C(x)$ depends upon ρ, $f_{xxx}(t,x;\theta)$ and $\sigma(x)$.

$$\tilde{G}_1(\theta_n, s_n) := \tilde{G}_{1,a}(\theta_n, s_n) + \tilde{G}_{1,b}(\theta_n, s_n) \tag{3.2}$$

$$\tilde{G}_{1,a}(\theta_n, s_n) := \left(\frac{\partial f}{\partial t}(t_n, x_n; \theta_n) + \mathcal{L}_1 f(t_n, x_n; \theta_n) + \frac{1}{2} \sum_{i=1}^{d} \frac{\frac{\partial f}{\partial x_i}(t, x_n + \sigma(x_n) W_\Delta; \theta) - \frac{\partial f}{\partial x_i}(t, x_n; \theta)}{\Delta} \sigma_i(x_n) W_\Delta^i \right)$$
$$\times \nabla_\theta \left(\frac{\partial f}{\partial t}(t_n, x_n; \theta_n) + \mathcal{L}_1 f(t_n, x_n; \theta_n) + \frac{1}{2} \sum_{i=1}^{d} \frac{\frac{\partial f}{\partial x_i}(t, x_n + \sigma(x_n) \tilde{W}_\Delta; \theta) - \frac{\partial f}{\partial x_i}(t, x_n; \theta)}{\Delta} \sigma_i(x_n) \tilde{W}_\Delta^i \right),$$

$$\tilde{G}_{1,b}(\theta_n, s_n) := \left(\frac{\partial f}{\partial t}(t_n, x_n; \theta_n) + \mathcal{L}_1 f(t_n, x_n; \theta_n) - \frac{1}{2} \sum_{i=1}^{d} \frac{\frac{\partial f}{\partial x_i}(t, x_n - \sigma(x_n) W_\Delta; \theta) - \frac{\partial f}{\partial x_i}(t, x_n; \theta)}{\Delta} \sigma_i(x_n) W_\Delta^i \right)$$
$$\times \nabla_\theta \left(\frac{\partial f}{\partial t}(t_n, x_n; \theta_n) + \mathcal{L}_1 f(t_n, x_n; \theta_n) - \frac{1}{2} \sum_{i=1}^{d} \frac{\frac{\partial f}{\partial x_i}(t, x_n - \sigma(x_n) \tilde{W}_\Delta; \theta) - \frac{\partial f}{\partial x_i}(t, x_n; \theta)}{\Delta} \sigma_i(x_n) \tilde{W}_\Delta^i \right).$$

The approximation (3.2) has $\mathcal{O}(\Delta)$ bias as an approximation for $\nabla_\theta G_1(\theta_n, s_n)$. Eq. (3.2) uses antithetic variates in the sense that $\tilde{G}_{1,a}(\theta_n, s_n)$ uses the random variables $(W_\Delta, \tilde{W}_\Delta)$ while $\tilde{G}_{1,b}(\theta_n, s_n)$ uses $(-W_\Delta, -\tilde{W}_\Delta)$. See [1] for a background on antithetic variates in simulation algorithms. A Taylor expansion can be used to show the approximation error is $\mathcal{O}(\Delta)$. It is important to highlight that there is no computational cost associated with the magnitude of Δ; an arbitrarily small Δ can be chosen with no additional computational cost (although there may be numerical underflow or overflow problems). The modified algorithm using the Monte Carlo approximation for the second derivatives is:

1. Generate random points (t_n, x_n) from $[0, T] \times \Omega$ and (τ_n, z_n) from $[0, T] \times \partial\Omega$ according to respective densities ν_1 and ν_2. Also, draw the random point w_n from Ω with density ν_3.
2. Calculate the step $\tilde{G}(\theta_n, s_n) = \tilde{G}_1(\theta_n, s_n) + \nabla_\theta G_2(\theta_n, s_n) + \nabla_\theta G_3(\theta_n, s_n)$ at the randomly sampled points $s_n = \{(t_n, x_n), (\tau_n, z_n), w_n\}$. $\tilde{G}(\theta_n, s_n)$ is an approximation for $\nabla_\theta G(\theta_n, s_n)$.
3. Take a step at the random point s_n:
$$\theta_{n+1} = \theta_n - \alpha_n \tilde{G}(\theta_n, s_n)$$
4. Repeat until convergence criterion is satisfied.

In conclusion, the modified algorithm here is computationally less expensive than the original algorithm in Section 2 but introduces some bias and variance. The variance essentially increases the i.i.d. noise in the stochastic gradient descent step; this noise averages out over a large number of samples though. The original algorithm in Section 2 is unbiased and has lower variance, but is computationally more expensive. We numerically implement the algorithm for a class of free boundary PDEs in Section 4. Future research may investigate other methods to further improve the computational evaluation of the second derivative terms (for instance, multi-level Monte Carlo).

4. Numerical analysis for a high-dimensional free boundary PDE

We test our algorithm on a class of high-dimensional free boundary PDEs. These free boundary PDEs are used in finance to price American options and are often referred to as "American option PDEs". An American option is a financial derivative on a portfolio of stocks. The option owner may at any time $t \in [0, T]$ choose to exercise the American option and receive a payoff which is determined by the underlying prices of the stocks in the portfolio. T is called the maturity date of the option and the payoff function is $g(x) : \mathbb{R}^d \to \mathbb{R}$. Let $X_t \in \mathbb{R}^d$ be the prices of d stocks. If at time t the stock prices $X_t = x$, the price of the option is $u(t, x)$. The price function $u(t, x)$ satisfies a free boundary PDE on $[0, T] \times \mathbb{R}^d$. For American options, one is primarily interested in the solution $u(0, X_0)$ since this is the fair price to buy or sell the option.

Besides the high dimensions and the free boundary, the American option PDE is challenging to numerically solve since the payoff function $g(x)$ (which both appears in the initial condition and determines the free boundary) is not continuously differentiable.

Section 4.1 states the free boundary PDE and the deep learning algorithm to solve it. To address the free boundary, we supplement the algorithm presented in Section 2 with an iterative method; see Section 4.1. Section 4.2 describes the architecture and implementation details for the neural network. Section 4.3 reports numerical accuracy for a case where a semi-analytic solution exists. Section 4.4 reports numerical accuracy for a case where no semi-analytic solution exists.

4.1. The free boundary PDE

We now specify the free boundary PDE for $u(t, x)$. The stock price dynamics and option price are:

$$dX_t^i = \mu(X_t^i)dt + \sigma(X_t^i)dW_t^i,$$
$$u(t, x) = \sup_{\tau \geq t} \mathbb{E}[e^{-r(\tau \wedge T)} g(X_{\tau \wedge T}) | X_t = x],$$

where $W_t \in \mathbb{R}^d$ is a standard Brownian motion and $\text{Cov}[dW_t^i, dW_t^j] = \rho_{i,j}dt$. The price of the American option is $u(0, X_0)$.

The model (4.1) for the stock price dynamics is widely used in practice and captures several desirable characteristics. First, the drift $\mu(x)$ measures the "average" growth in the stock prices. The Brownian motion W_t represents the randomness in the stock price, and the magnitude of the randomness is given by the coefficient function $\sigma(X_t^i)$. The movement of stock prices are correlated (e.g., if Microsoft's price increases, it is likely that Apple's price will also increase). The magnitude of the correlation between two stocks i and j is specified by the parameter $\rho_{i,j}$. An example is the well-known Black–Scholes model $\mu(x) = \mu x$ and $\sigma(x) = \sigma x$. In the Black–Scholes model, the average rate of return for each stock is μ.

An American option is a financial derivative which the owner can choose to "exercise" at any time $t \in [0, T]$. If the owner exercises the option, they receive the financial payoff $g(X_t)$ where X_t is the prices of the underlying stocks. If the owner does not choose to exercise the option, they receive the payoff $g(X_T)$ at the final time T. The value (or price) of the American option at time t is $u(t, X_t)$. Some typical examples of the payoff function $g(x) : \mathbb{R}^d \to \mathbb{R}$ are $g(x) = \max\left((\prod_{i=1}^d x_i)^{1/d} - K, 0\right)$ and $g(x) = \max\left(\frac{1}{d}\sum_{i=1}^d x_i - K, 0\right)$. The former is referred to as a "geometric payoff function" while the latter is called an "arithmetic payoff function." K is the "strike price" and is a positive number.

The price function $u(t, x)$ in (4.1) is the solution to a free boundary PDE and will satisfy:

$$
\begin{aligned}
&0 = \frac{\partial u}{\partial t}(t,x) + \mu(x)\cdot\frac{\partial u}{\partial x}(t,x) + \frac{1}{2}\sum_{i,j=1}^d \rho_{i,j}\sigma(x_i)\sigma(x_j)\frac{\partial^2 u}{\partial x_i \partial x_j}(t,x) - ru(t,x), \quad \forall\big\{(t,x) : u(t,x) > g(x)\big\}.\\
&u(t,x) \geq g(x), \quad \forall (t,x).\\
&u(t,x) \in C^1(\mathbb{R}_+ \times \mathbb{R}^d), \quad \forall\big\{(t,x) : u(t,x) = g(x)\big\}.\\
&u(T,x) = g(x), \quad \forall x.
\end{aligned} \tag{4.1}
$$

The free boundary set is $F = \big\{(t,x) : u(t,x) = g(x)\big\}$. $u(t,x)$ satisfies a partial differential equation "above" the free boundary set F, and $u(t,x)$ equals the function $g(x)$ "below" the free boundary set F.

The deep learning algorithm for solving the PDE (4.1) requires simulating points above and below the free boundary set F. We use an iterative method to address the free boundary. The free boundary set F is approximated using the current parameter estimate θ_n. This approximate free boundary is used in the probability measure that we simulate points with. The gradient is not taken with respect to the θ_n input of the probability density used to simulate random points. For this purpose, define the objective function:

$$
\begin{aligned}
J(f;\theta,\tilde{\theta}) = &\left\| \frac{\partial f}{\partial t}(t,x;\theta) + \mu(x)\cdot\frac{\partial f}{\partial x}(t,x;\theta) + \frac{1}{2}\sum_{i,j=1}^d \rho_{i,j}\sigma(x_i)\sigma(x_j)\frac{\partial^2 f}{\partial x_i \partial x_j}(t,x;\theta) - rf(t,x;\theta)\right\|^2_{[0,T]\times\Omega,\nu_1(\tilde{\theta})}\\
&+ \|\max(g(x) - f(t,x;\theta), 0)\|^2_{[0,T]\times\Omega,\nu_2(\tilde{\theta})}\\
&+ \|f(T,x;\theta) - g(x)\|^2_{\Omega,\nu_3}.
\end{aligned}
$$

Descent steps are taken in the direction $-\nabla_\theta J(f;\theta,\tilde{\theta})$. $\nu_1(\tilde{\theta})$ and $\nu_2(\tilde{\theta})$ are the densities of the points in $\tilde{B}^1$ and $\tilde{B}^2$, which are defined below. The deep learning algorithm is:

1. Generate the random batch of points $B^1 = \{t_m, x_m\}_{m=1}^M$ from $[0,T]\times\Omega$ according to the probability density ν_1^0. Select the points $\tilde{B}^1 = \{(t,x) \in B^1 : f(t,x;\theta_n) > g(x)\}$.
2. Generate the random batch of points $B^2 = \{\tau_m, z_m\}_{m=1}^M$ from $[0,T]\times\partial\Omega$ according to the probability density ν_2^0. Select the points $\tilde{B}^2 = \{(\tau,z) \in B^2 : f(\tau,z;\theta_n) \leq g(z)\}$.
3. Generate the random batch of points $B^3 = \{w_m\}_{m=1}^M$ from Ω with probability density ν_3.
4. Approximate $J(f;\theta_n,\tilde{\theta}_n)$ as $J(f;\theta_n,S_n)$ at the randomly sampled points $S_n = \{\tilde{B}^1, \tilde{B}^2, B^3\}$:

$$
\begin{aligned}
J(f;\theta_n,S_n) = &\frac{1}{|\tilde{B}^1|}\sum_{(t_m,x_m)\in\tilde{B}^1}\left(\frac{\partial f}{\partial t}(t_m,x_m;\theta_n) + \mu(x_m)\cdot\frac{\partial f}{\partial x}(t_m,x_m;\theta_n)\right.\\
&\left.+ \frac{1}{2}\sum_{i,j=1}^d \rho_{i,j}\sigma(x_i)\sigma(x_j)\frac{\partial^2 f}{\partial x_i\partial x_j}(t_m,x_m;\theta_n) - rf(t_m,x_m;\theta_n)\right)^2\\
&+ \frac{1}{|\tilde{B}^2|}\sum_{(\tau_m,z_m)\in\tilde{B}^2}\max\big(g(z_m) - f(\tau_m,z_m;\theta_n), 0\big)^2\\
&+ \frac{1}{|B^3|}\sum_{w_m\in B^3}\Big(f(T,w_m;\theta) - g(w_m)\Big)^2.
\end{aligned}
$$

5. Take a descent step for the random batch S_n:

$$\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta J(f; \theta_n, S_n).$$

6. Repeat until convergence criterion is satisfied.

The second derivatives in the above algorithm can be approximated using the method from Section 3.

4.2. Implementation details for the algorithm

This section provides details for the implementation of the algorithm, including the DGM network architecture, hyperparameters, and computational approach.

The architecture of a neural network can be crucial to its success. Frequently, different applications require different architectures. For example, convolution networks are essential for image recognition while long short-term networks (LSTMs) are useful for modeling sequential data. Clever choices of architectures, which exploit a priori knowledge about an application, can significantly improve performance. In the PDE applications in this paper, we found that a neural network architecture similar in spirit to that of LSTM networks improved performance.

The PDE solution requires a model $f(t,x;\theta)$ which can make "sharp turns" due to the final condition, which is of the form $u(T,x) = \max(p(x),0)$ (the first derivative is discontinuous when $p(x)=0$). The shape of the solution $u(t,x)$ for $t<T$, although "smoothed" by the diffusion term in the PDE, will still have a nonlinear profile which is rapidly changing in certain spatial regions. In particular, we found the following network architecture to be effective:

$$\begin{aligned}
S^1 &= \sigma(W^1 \vec{x} + b^1),\\
Z^\ell &= \sigma(U^{z,\ell}\vec{x} + W^{z,\ell} S^\ell + b^{z,\ell}), \quad \ell = 1,\dots,L,\\
G^\ell &= \sigma(U^{g,\ell}\vec{x} + W^{g,\ell} S^1 + b^{g,\ell}), \quad \ell = 1,\dots,L,\\
R^\ell &= \sigma(U^{r,\ell}\vec{x} + W^{r,\ell} S^\ell + b^{r,\ell}), \quad \ell = 1,\dots,L,\\
H^\ell &= \sigma(U^{h,\ell}\vec{x} + W^{h,\ell}(S^\ell \odot R^\ell) + b^{h,\ell}), \quad \ell = 1,\dots,L,\\
S^{\ell+1} &= (1-G^\ell)\odot H^\ell + Z^\ell \odot S^\ell, \quad \ell = 1,\dots,L,\\
f(t,x;\theta) &= W S^{L+1} + b,
\end{aligned} \tag{4.2}$$

where $\vec{x} = (t,x)$, the number of hidden layers is $L+1$, and $\odot$ denotes element-wise multiplication (i.e., $z \odot v = (z_0 v_0, \dots, z_N v_N)$). The parameters are

$$\theta = \left\{ W^1, b^1, \left(U^{z,\ell}, W^{z,\ell}, b^{z,\ell}\right)_{\ell=1}^{L}, \left(U^{g,\ell}, W^{g,\ell}, b^{g,\ell}\right)_{\ell=1}^{L}, \left(U^{r,\ell}, W^{r,\ell}, b^{r,\ell}\right)_{\ell=1}^{L}, \left(U^{h,\ell}, W^{h,\ell}, b^{h,\ell}\right)_{\ell=1}^{L}, W, b \right\}.$$

The number of units in each layer is M and $\sigma : \mathbb{R}^M \to \mathbb{R}^M$ is an element-wise nonlinearity:

$$\sigma(z) = \Big(\phi(z_1), \phi(z_2), \dots, \phi(z_M)\Big), \tag{4.3}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function such as the tanh function, sigmoidal function $\frac{e^y}{1+e^y}$, or rectified linear unit (ReLU) $\max(y,0)$. The parameters in θ have dimensions $W^1 \in \mathbb{R}^{M\times(d+1)}$, $b^1 \in \mathbb{R}^M$, $U^{z,\ell} \in \mathbb{R}^{M\times(d+1)}$, $W^{z,\ell} \in \mathbb{R}^{M\times M}$, $b^{z,\ell} \in \mathbb{R}^M$, $U^{g,\ell} \in \mathbb{R}^{M\times(d+1)}$, $W^{g,\ell} \in \mathbb{R}^{M\times M}$, $b^{g,\ell} \in \mathbb{R}^M$, $U^{r,\ell} \in \mathbb{R}^{M\times(d+1)}$, $W^{r,\ell} \in \mathbb{R}^{M\times M}$, $b^{r,\ell} \in \mathbb{R}^M$, $U^{h,\ell} \in \mathbb{R}^{M\times(d+1)}$, $W^{h,\ell} \in \mathbb{R}^{M\times M}$, $b^{h,\ell} \in \mathbb{R}^M$, $W \in \mathbb{R}^{1\times M}$, and $b \in \mathbb{R}$.

The architecture (4.2) is relatively complicated. Within each layer, there are actually many "sub-layers" of computations. The important feature is the repeated element-wise multiplication of nonlinear functions of the input. This helps to model more complicated functions which are rapidly changing in certain time and space regions. The neural network architecture (4.2) is similar to the architecture for LSTM networks (see [24]) and highway networks (see [47]).

The key hyperparameters in the neural network (4.2) are the number of layers L, the number of units M in each sub-layer, and the choice of activation unit $\phi(y)$. We found for the applications in this paper that the hyperparameters $L=3$ (i.e., four hidden layers), $M=50$, and $\phi(y) = \tanh(y)$ were effective. It is worthwhile to note that the choice of $\phi(y) = \tanh(y)$ means that $f(t,x;\theta)$ is smooth and therefore can solve for a "classical solution" of the PDE. The neural network parameters are initialized using the Xavier initialization (see [18]). The architecture (4.2) is bounded in the input x (for a fixed choice of parameters θ) if $\sigma(\cdot)$ is a tanh or sigmoidal function; it may be helpful to allow the network to be unbounded for approximating unbounded/growing functions. We found that replacing the $\sigma(\cdot)$ in the H^ℓ sub-layer with the identity function can be an effective way to develop an unbounded network.

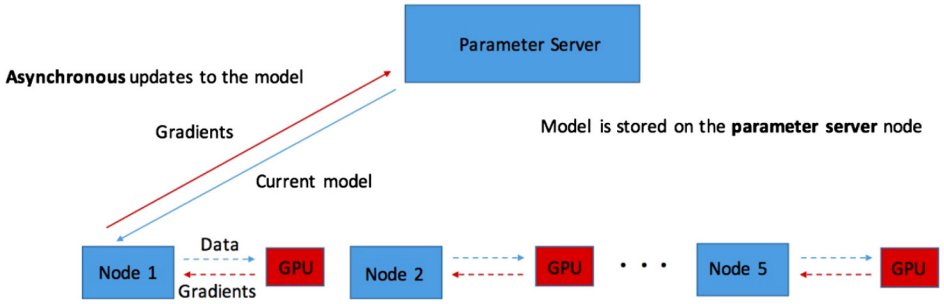

Fig. 1. Asynchronous stochastic gradient descent on a cluster of GPU nodes. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

We emphasize that the only input to the network is (t, x). We do not use any custom-designed nonlinear transformations of (t, x). If properly chosen, such additional inputs might help performance. For example, the European option PDE solution (which has an analytic formula) could be included as an input.

A regularization term (such as an ℓ^2 penalty) could also be included in the objective function for the algorithm. Such regularization terms are used for reducing overfitting in machine learning models estimated using datasets which have a limited number of data samples. (For example, a model estimated on a dataset of 60,000 images.) However, it's unclear if this will be helpful in the context of this paper's application, since there is no strict upper bound on the size of the dataset (i.e., one can always simulate more time/space points).

Our computational approach to training the neural network involved several components. The second derivatives are approximated using the method from Section 3. Training is distributed across 6 GPU nodes using asynchronous stochastic gradient descent (we provide more details on this below). Parameters are updated using the well-known ADAM algorithm (see [27]) with a decaying learning rate schedule (more details on the learning rate are provided below). Accuracy can be improved by calculating a running average of the neural network solutions over a sequence of training iterations (essentially a computationally cheap approach for building a model ensemble). We also found that model ensembles (of even small sizes of 5) can slightly increase accuracy.

Training of the neural network is distributed across several GPU nodes in order to accelerate training. We use asynchronous stochastic gradient descent, which is a widely-used method for parallelizing training of machine learning models. On each node, i.i.d. space and time samples are generated. Each node calculates the gradient of the objective function with respect to the parameters on its respective batch of simulated data. These gradients are then used to update the model, which is stored on a central node called a "parameter server". Fig. 1 displays the computational setup. Updates occur asynchronously; that is, node i updates the model immediately upon completion of its work, and does not wait for node j to finish its work. The "work" here is calculating the gradients for a batch of simulated data. Before a node calculates the gradient for a new batch of simulated data, it receives an updated model from the parameter server. For more details on asynchronous stochastic gradient descent, see [13].

During training, we decrease the learning as the number of iterations increases. We use a learning rate schedule where the learning rate is a piecewise constant function of the number of iterations. This is a typical choice. We found the following learning rate schedule to be effective:

$$\alpha_n = \begin{cases} 10^{-4} & n \leq 5{,}000 \\ 5 \times 10^{-4} & 5{,}000 < n \leq 10{,}000 \\ 10^{-5} & < 10{,}000 < n \leq 20{,}000 \\ 5 \times 10^{-6} & 20{,}000 < n \leq 30{,}000 \\ 10^{-6} & 30{,}000 < n \leq 40{,}000 \\ 5 \times 10^{-7} & 40{,}000 < n \leq 45{,}000 \\ 10^{-7} & 45{,}000 < n \end{cases}$$

We use approximately 100,000 iterations. An "iteration" involves batches of size 1,000 on each of the GPU nodes. Therefore, there are 5,000 simulated time/space points for each iteration. In total, we used approximately 500 million simulated time/space points to train the neural network.

We implement the algorithm using TensorFlow and PyTorch, which are software libraries for deep learning. TensorFlow has reverse mode automatic differentiation which allows the calculation of derivatives for a broad range of functions. For example, TensorFlow can be used to calculate the gradient of the neural network (4.2) with respect to x or θ. TensorFlow also allows for the training of models on graphics processing units (GPUs). A GPU, which has thousands of cores, can be used to highly parallelize the training of deep learning models. We furthermore distribute our computations across multiple GPU nodes, as described above. The computations in this paper were performed on the Blue Waters supercomputer which has a large number of GPU nodes.

Table 1
The deep learning algorithm solution is compared with a semi-analytic solution for the Black–Scholes model. The parameters $\mu(x) = (r-c)x$ and $\sigma(x) = \sigma x$. All stocks are identical with correlation $\rho_{i,j} = .75$, volatility $\sigma = .25$, initial stock price $X_0 = 1$, dividend rate $c = 0.02$, and interest rate $r = 0$. The maturity of the option is $T = 2$ and the strike price is $K = 1$. The payoff function is $g(x) = \max((\prod_{i=1}^{d} x_i)^{1/d} - K, 0)$. The error is reported for the price $u(0, X_0)$ of the at-the-money American call option. The error is $\frac{|f(0,X_0;\theta)-u(0,X_0)|}{|u(0,X_0)|} \times 100\%$.

Number of dimensions	Error
3	0.05%
20	0.03%
100	0.11%
200	0.22%

4.3. A high-dimensional free boundary PDE with a semi-analytic solution

We implement our deep learning algorithm to solve the PDE (4.1). The accuracy of our deep learning algorithm is evaluated in up to 200 dimensions. The results are reported below in Table 1.

The semi-analytic solution used in Table 1 is provided below. Let $\mu(x) = (r-c)x$, $\sigma(x) = \sigma x$, and $\rho_{i,j} = \rho$ for $i \neq j$ (i.e., the Black–Scholes model). If the payoff function in (4.1) is $g(x) = \max\left((\prod_{i=1}^{d} x_i)^{1/d} - K, 0\right)$, then there is a semi-analytic solution to (4.1):

$$u(t,x) = v(t, (\prod_{i=1}^{d} x_i)^{1/d} - K), \tag{4.4}$$

where $v(t,x)$ satisfies the one-dimensional free boundary PDE

$$\begin{aligned}
0 &= \frac{\partial v}{\partial t}(t,x) + \hat{\mu} x \frac{\partial v}{\partial x}(t,x) + \frac{1}{2}\hat{\sigma}^2 x \frac{\partial^2 v}{\partial x^2}(t,x) - r v(t,x), \quad \forall \ \left\{(t,x) : v(t,x) > \hat{g}(x)\right\}. \\
v(t,x) &\geq \hat{g}(x), \quad \forall \ (t,x). \\
\hat{v}(t,x) &\in C^1(\mathbb{R}_+ \times \mathbb{R}^d), \quad \forall \ \left\{(t,x) : v(t,x) = \hat{g}(x)\right\}. \\
v(T,x) &= \hat{g}(x), \quad \forall \ x,
\end{aligned} \tag{4.5}$$

where $\hat{\sigma}^2 = \frac{d\sigma^2 + d(d-1)\rho\sigma^2}{d^2}$, $\hat{\mu} = (r-c) + \frac{1}{2}\hat{\sigma}^2 - \frac{1}{2}\sigma^2$, and $\hat{g}(x) = \max(x, 0)$. The one-dimensional PDE (4.5) can be solved using finite difference methods. If $f(t,x;\theta)$ is the deep learning algorithm's estimate for the PDE solution at (t,x), the relative error at the point (t,x) is $\frac{|f(t,x;\theta)-u(t,x)|}{|u(t,x)|} \times 100\%$ and the absolute error at the point (t,x) is $|f(t,x;\theta) - u(t,x)|$. The relative error and absolute error at the point (t,x) can be evaluated using the semi-analytic solution (4.4).

Although the solution at $(t,x) = (0, X_0)$ is of primary interest for American options, most other PDE applications are interested in the entire solution $u(t,x)$. The deep learning algorithm provides an approximate solution across all time and space $(t,x) \in [0,T] \times \Omega$. As an example, we present in Fig. 2 contour plots of the absolute error and percent error across time and space for the American option PDE in 20 dimensions. The contour plot is produced in the following way:

1. Sample time points t^ℓ uniformly on $[0,T]$ and sample spatial points $x^\ell = (x_1^\ell, \ldots, x_{20}^\ell)$ from the joint distribution of $X_t^1, \ldots, X_t^{20}$ in equation (4.1). This produces an "envelope" of sampled points since X_t spreads out as a diffusive process from $X_0 = 1$.
2. Calculate the error E^ℓ at each sampled point (t^ℓ, x^ℓ) for $\ell = 1, \ldots, L$.
3. Aggregate the error over a two-dimensional subspace $\left(t^\ell, (\prod_{i=1}^{20} x_i^\ell)^{1/20}, E^\ell\right)$ for $\ell = 1, \ldots, L$.
4. Produce a contour plot from the data $\left(t^\ell, (\prod_{i=1}^{20} x_i^\ell)^{1/20}, E^\ell\right)_{\ell=1}^{L}$. The x-axis is t and the y-axis is the geometric average $(\prod_{i=1}^{20} x_i)^{1/20}$, which corresponds to the final condition $g(x)$.

Fig. 2 reports both the absolute error and the percent error. The percent error $\frac{|f(t,x;\theta)-u(t,x)|}{|u(t,x)|} \times 100\%$ is reported for points where $|u(t,x)| > 0.05$. The absolute error becomes relatively large in a few areas; however, the solution $u(t,x)$ also grows large in these areas and therefore the percent error remains small.

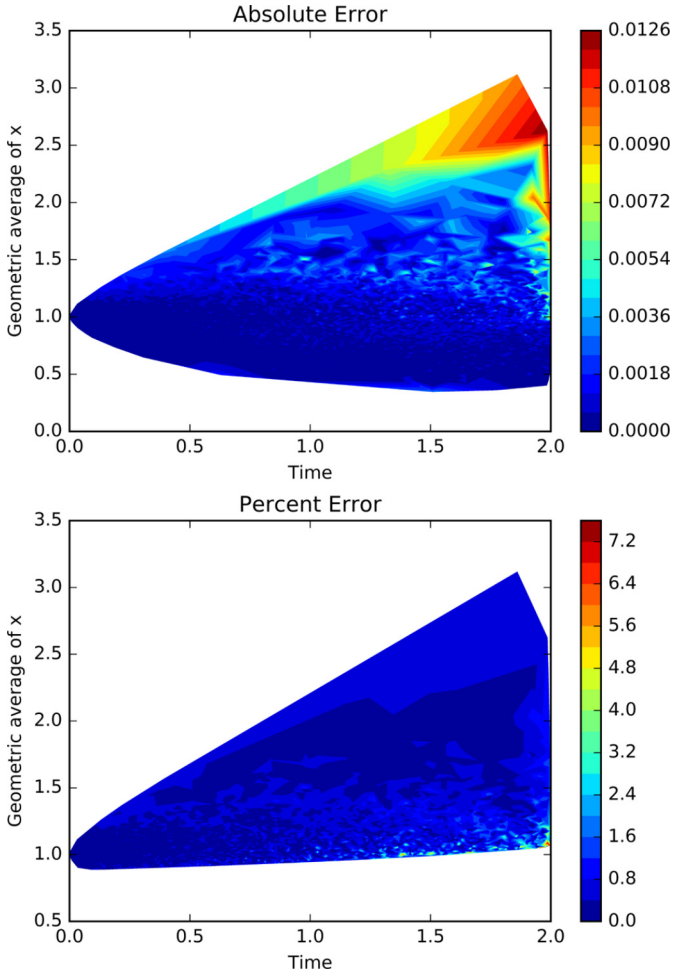

Fig. 2. Top: Absolute error. Bottom: Percent error. For reference, the price at time 0 is 0.1003 and the solution at time T is max(geometric average of $x-1, 0$).

4.4. *A high-dimensional free boundary PDE* without *a semi-analytic solution*

We now consider a case of the American option PDE which does not have a semi-analytic solution. The American option PDE has the special property that it is possible to calculate error bounds on an approximate solution. Therefore, we can evaluate the accuracy of the deep learning algorithm even on cases where no semi-analytic solution is available.

We previously only considered a symmetrical case where $\rho_{i,j} = 0.75$ and $\sigma = 0.25$ for all stocks. This section solves a more challenging heterogeneous case where $\rho_{i,j}$ and σ_i vary across all dimensions $i = 1, 2, \ldots, d$. The coefficients are fitted to actual data for the stocks IBM, Amazon, Tiffany, Amgen, Bank of America, General Mills, Cisco, Coca-Cola, Comcast, Deere, General Electric, Home Depot, Johnson & Johnson, Morgan Stanley, Microsoft, Nordstrom, Pfizer, Qualcomm, Starbucks, and Tyson Foods from 2000–2017. This produces a PDE with widely-varying coefficients for each of the $\frac{d^2+d}{2}$ second derivative terms. The correlation coefficients $\rho_{i,j}$ range from -0.53 to 0.80 for $i \neq j$ and σ_i ranges from 0.09 to 0.69.

Let $f(t, x; \theta)$ be the neural network approximation. [45] derived that the PDE solution $u(t, x)$ lies in the interval:

$$
\begin{aligned}
&u(t,x) \in \left[\underline{u}(t,x), \overline{u}(t,x)\right], \\
&\underline{u}(t,x) = \mathbb{E}\Big[\, g(X_\tau) | X_t = x, \tau > t \,\Big], \\
&\overline{u}(t,x) = \mathbb{E}\Big[\sup_{s\in[t,T]} \left[e^{-r(s-t)} g(X_s) - M_s\right]\Big],
\end{aligned} \tag{4.6}
$$

where $\tau = \inf\{t \in [0, T] : f(t, X_t; \theta) < g(X_t)\}$ and M_s is a martingale constructed from the approximate solution $f(t, x; \theta)$

Table 2
The accuracy of the deep learning algorithm is evaluated on a case where there is no semi-analytic solution. The parameters $\mu(x) = (r-c)x$ and $\sigma(x) = \sigma x$. The correlations $\rho_{i,j}$ and volatilities σ_i are estimated from data to generate a heterogeneous diffusion matrix. The initial stock price is $X_0 = 1$, dividend rate $c = 0.02$, and interest rate $r = 0$ for all stocks. The maturity of the option is $T = 2$. The payoff function is $g(x) = \max(\frac{1}{d}\sum_{i=1}^{d} x_i - K, 0)$. The neural network solution and its error bounds are reported for the price $u(0, X_0)$ of the American call option. The best estimate for the price of the American option is the midpoint of the interval $[\underline{u}(0, X_0), \overline{u}(0, X_0)]$, which has an error bound of $\frac{\overline{u}(0,X_0)-\underline{u}(0,X_0)}{2\underline{u}(0,X_0)} \times 100\%$. In order to calculate the upper bound, the integral (4.7) is discretized with time step size $\Delta = 5 \times 10^{-4}$.

Strike price	Neural network solution	Lower bound	Upper bound	Error bound
0.90	0.14833	0.14838	0.14905	0.23%
0.95	0.12286	0.12270	0.12351	0.33%
1.00	0.10136	0.10119	0.10193	0.37%
1.05	0.08334	0.08315	0.08389	0.44%
1.10	0.06841	0.06809	0.06893	0.62%

$$M_s = f(s, X_s; \theta) - f(t, X_t; \theta) - \int_t^s \left[\frac{\partial f}{\partial t}(s', X_{s'}; \theta) + \mu(X_{s'}) \frac{\partial f}{\partial x}(s', X_{s'}; \theta) + \frac{1}{2} \sum_{i,j=1}^{d} \sigma(X_{s',i}) \sigma(X_{s',j}) \frac{\partial^2 f}{\partial x_i \partial x_j}(s', X_{s'}; \theta) - r f(s', X_{s'}; \theta) \right] ds'. \tag{4.7}$$

The bounds (4.6) depend only on the approximation $f(t, x; \theta)$, which is known, and can be evaluated via Monte Carlo simulation. The integral for M_s must also be discretized. The best estimate for the price of the American option is the midpoint of the interval $[\underline{u}(0, X_0), \overline{u}(0, X_0)]$, which has an error bound of $\frac{\overline{u}(0,X_0)-\underline{u}(0,X_0)}{2\underline{u}(0,X_0)} \times 100\%$. Numerical results are in Table 2.

We present in Fig. 3 contour plots of the absolute error bound and percent error bound across time and space for the American option PDE in 20 dimensions for strike price $K = 1$. The contour plot is produced in the following way:

1. Sample time points t^ℓ uniformly on $[0, T]$ and sample spatial points $x^\ell = (x_1^\ell, \ldots, x_{20}^\ell)$ from the joint distribution of $X_t^1, \ldots, X_t^{20}$ in equation (4.1).
2. Calculate the error E^ℓ at each sampled point (t^ℓ, x^ℓ) for $\ell = 1, \ldots, L$.
3. Aggregate the error over a two-dimensional subspace $\left(t^\ell, \frac{1}{20}\sum_{i=1}^{20} x_i^\ell, E^\ell\right)$ for $\ell = 1, \ldots, L$.
4. Produce a contour plot from the data $\left(t^\ell, \frac{1}{20}\sum_{i=1}^{20} x_i^\ell, E^\ell\right)_{\ell=1}^{L}$. The x-axis is t and the y-axis is the geometric average $\frac{1}{20}\sum_{i=1}^{20} x_i^\ell$, which corresponds to the final condition $g(x)$.

Fig. 3 reports both the absolute error and the percent error. The percent error $\frac{|f(t,x;\theta)-u(t,x)|}{|u(t,x)|} \times 100\%$ is reported for points where $|u(t, x)| > 0.05$. It should be emphasized that these are *error bounds*; therefore, the actual error could be lower. The contour plot (Fig. 3) requires significant computations. For each point at which calculate an error bound, a new simulation of (4.6) is required. In total, a large number of simulations are required, which we distribute across hundreds of GPUs on the Blue Waters supercomputer.

5. High-dimensional Hamilton–Jacobi–Bellman PDE

We also test the deep learning algorithm on a high-dimensional Hamilton–Jacobi–Bellman (HJB) equation corresponding to the optimal control of a stochastic heat equation. Specifically, we demonstrate that the deep learning algorithm accurately solves the high-dimensional PDE (5.5). The PDE (5.5) is motivated by the problem of optimally controlling the stochastic partial differential equation (SPDE):

$$\begin{aligned} &\frac{\partial v}{\partial t}(t, x) = \alpha \frac{\partial^2 v}{\partial x^2}(t, x) + u(x) + \sigma \frac{\partial^2 W}{\partial t \partial x}(t, x), \quad x \in [0, L], \\ &v(t, x = 0) = \bar{v}(0), \\ &v(t, x = L) = \bar{v}(L), \\ &v(t = 0, x) = v_0(x), \end{aligned} \tag{5.1}$$

where $u(x)$ is the control and $W(t, x)$ is a Brownian sheet (i.e., $\frac{\partial^2 W}{\partial t \partial x}(t, x)$ is space–time white noise) defined on a stochastic basis $(\Omega, \mathcal{F}, \mathcal{F}_t, \mathbb{P})$. The square integrable, adapted to the filtration $\mathcal{F}_t$, control u is a source/sink term which can be used to guide the temperature $v(t, x)$ towards a target profile $\bar{v}(x)$ on $[0, L]$. As it is discussed in [10] such problems admit unique

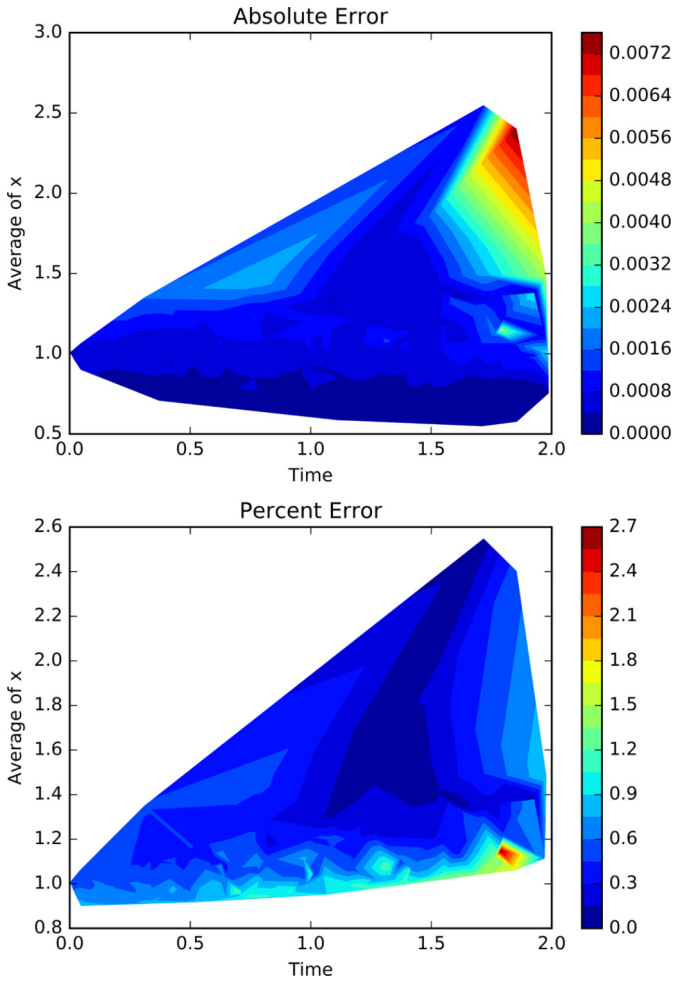

Fig. 3. Top: Absolute error. Bottom: Percent error. For reference, $u(0, X_0) \in [0.10119, 0.10193]$ and the solution at time T is max(average of $x - 1, 0$).

solutions in the appropriate generalized sense, see Theorem 3.1 in [10]. The endpoints at $x = 0, L$ are held at the target temperatures. Specifically, the optimal control minimizes

$$\mathbb{E}\Big[\int_0^\infty e^{-\gamma s}\int_0^L \big[(v(s,x) - \bar{v}(x))^2 + \lambda u(x)^2\big]dxds\Big]. \tag{5.2}$$

The constant $\gamma > 0$ is a discount factor. The constant $\lambda > 0$ penalizes large values for the control $u(x)$. The goal is to reach the target $\bar{v}(x)$ while expending the minimum amount of energy. The optimal control $u(x)$ satisfies an *infinite-dimensional* HJB equation. We refer the reader to Theorems 5.3 and 5.4 of [10] as well as [14] and [36] for an analysis of infinite-dimensional HJB equations for the stochastic heat equation.

An example of a problem represented by the SPDE (5.1) is the heating of a rod to a target temperature profile. One can control the heat applied to each portion of the rod along its length. There are also random fluctuations in the temperature of the rod due to other environmental factors, which is represented by the Brownian sheet $W(t, x)$. The goal is to guide the temperature profile of the rod to the target profile while expending the least amount of energy; see the objective function (5.2).

(5.1) can be discretized in space, which yields a system of stochastic differential equations (SDEs). (For example, see Section 3.2 of [19].) This system of SDEs can be used to derive a finite, high-dimensional PDE for the value function and optimal control. That is, we first approximate the SPDE with a finite-dimensional system of SDEs, and then we solve the high-dimensional PDE corresponding to the finite-dimensional system of SDEs.

$$dX_t^j = \frac{\alpha}{\Delta^2}(X_t^{j+1} - 2X_t^j + X_t^{j-1})dt + U_t^j dt + \frac{\sigma}{\sqrt{\Delta}}dW_t^j, \; X_0^j = v_0(j\Delta), \tag{5.3}$$

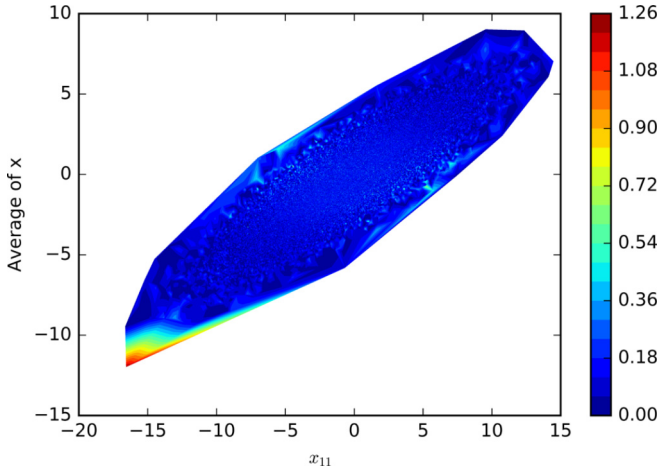

Fig. 4. Contour plot of the percent error for the deep learning algorithm for a 21-dimensional Hamilton–Jacobi–Bellman PDE. The horizontal axis is the 11-th dimension. The vertical axis is the average of all dimensions.

where Δ is the mesh size, $v(t, j\Delta) = X_t^j$, $u(j\Delta) = U_t^j$, and W_t^j are independent standard Brownian motions (see [12], [21], and [19] regarding numerical schemes for stochastic parabolic PDEs of the form considered in this section). The dimension of the SDE system (5.3) is $d = \frac{L}{\Delta} - 1$. Note that (5.3) uses a central difference scheme for the diffusion term in (5.1).

The objective function (5.2) becomes:

$$V(x) = \inf_{U_t \in \mathcal{U}} \mathbb{E}\left[\int_0^\infty e^{-\gamma s} \sum_{j=1}^{d} \left[(X_s^j - \bar{v}(j\Delta))^2 + \lambda (U_s^j)^2 \right] \Delta ds \,\middle|\, X_0 = x \right]. \tag{5.4}$$

The value function $V(x)$ satisfies a nonlinear PDE with d spatial dimensions $x_1, x_2, \ldots, x_d$.

$$\begin{aligned} 0 = {} & \Delta (x - \bar{v})^\top (x - \bar{v}) - \frac{1}{4\lambda\Delta} \sum_{j=1}^{d} \left(\frac{\partial V}{\partial x_j}(x) \right)^2 \\ & + \frac{\sigma^2}{2\Delta} \sum_{j=1}^{d} \frac{\partial^2 V}{\partial x_j^2}(x) + \frac{\alpha}{\Delta^2} \sum_{j=1}^{d} (x_{j+1} - 2x_j + x_{j-1}) \frac{\partial V}{\partial x_j}(x) - \gamma V(x). \end{aligned} \tag{5.5}$$

The vector $\bar{v} = (\bar{v}(\Delta), \bar{v}(2\Delta), \ldots, \bar{v}(d\Delta))$. Note that the values $x_{d+1} = \bar{v}(L)$ and $x_0 = \bar{v}(0)$ are constants which correspond to the boundary conditions in (5.1). The PDE (5.5) is high dimensional since the number of dimensions $d = \frac{L}{\Delta} - 1$. The optimal control is

$$U_t^j = -\frac{1}{2\lambda\Delta} \frac{\partial V}{\partial x_j}(X_t). \tag{5.6}$$

We solve the PDE (5.5) using the deep learning algorithm for $d = 21$ dimensions. The size of the domain is $L = 10^{-1}$. The coefficients are $\alpha = 10^{-4}$, $\sigma = 10^{-\frac{1}{2}}$, $\lambda = 1$, and $\gamma = 1$. The target profile is $\bar{v}(x) = 0$.

The deep learning algorithm's accuracy can be evaluated since a semi-analytic solution is available for (5.5).[3] Fig. 4 shows a contour plot of the percent error over space. The contour plot is produced in the following way:

1. Sample spatial points $x^\ell = (x_1^\ell, \ldots, x_{21}^\ell)$ from the distribution of (5.3) for $\ell = 1, \ldots, L$.
2. Calculate the percent error at each sampled point. The percent error is $A^\ell = \frac{|f(x^\ell;\theta) - V(x^\ell)|}{|V(x^\ell)|} \times 100\%$.
3. Aggregate the accuracy over a two-dimensional subspace $\left(x_{11}^\ell, \frac{1}{21} \sum_{i=1}^{21} x_i^\ell, A^\ell \right)$ for $\ell = 1, \ldots, L$.
4. Produce a contour plot from the data $\left(x_{11}^\ell, \frac{1}{21} \sum_{i=1}^{21} x_i^\ell, A^\ell \right)_{\ell=1}^{L}$. The x-axis is x_{11} and the y-axis is the average $\frac{1}{21} \sum_{i=1}^{21} x_i$. This corresponds to $v(t, x)$ at the midpoint $x = \frac{L}{2}$ and the average $\frac{1}{L} \int_0^L v(t, x) dx$, respectively.

The average percent error over the entire space is 0.1%.

[3] The PDE (5.5) has a semi-analytic solution which satisfies a Riccati equation. The Riccati equation can be solved using an iterative method.

Lastly, we close this section by mentioning that in the recent paper [15] (see also [2]) the authors develop a machine learning algorithm that provides the value at a single point in time and space of the solution to a class of HJB equations which admit explicit solution that can be obtained through the Cole–Hopf transformation. Their method relies on characterizing the solution via backward stochastic differential equations (BSDE). In contrast, the current work (a) does not rely on BSDE type representations through nonlinear Feynman–Kac formulas, and (b) allows to recover the whole object (i.e. the solution across all points in time and space).

6. Burgers' equation

It is often of interest to find the solution of a PDE over a range of problem setups (e.g., different physical conditions and boundary conditions). For example, this may be useful for the design of engineering systems or uncertainty quantification. The problem setup space may be high-dimensional and therefore may require solving many PDEs for many different problem setups, which can be computationally expensive.

Let the variable p represent the problem setup (i.e., physical conditions, boundary conditions, and initial conditions). The variable p takes values in the space $\mathcal{P}$, and we are interested in the solution of the PDE $u(t, x; p)$. (This is sometimes called a "parameterized class of PDEs".) In particular, suppose $u(t, x; p)$ satisfies the PDE

$$\begin{aligned} \frac{\partial u}{\partial t}(t, x; p) &= \mathcal{L}_p u(t, x; p), \quad (t, x) \in [0, T] \times \Omega, \\ u(t, x; p) &= g_p(x), \quad (t, x) \in [0, T] \times \partial\Omega, \\ u(t = 0, x; p) &= h_p(x), \quad x \in \Omega. \end{aligned} \tag{6.1}$$

A traditional approach would be to discretize the $\mathcal{P}$-space and re-solve the PDE many times for many different points p. However, the total number of grid points (and therefore the number of PDEs that must be solved) grows exponentially with the number of dimensions, and $\mathcal{P}$ is typically high-dimensional.

We propose to use the DGM algorithm to approximate the *general solution* to the PDE (6.1) for different boundary conditions, initial conditions, and physical conditions. The deep neural network is trained using stochastic gradient descent on a sequence of random time, space, and problem setup points (t, x, p). Similar to before,

- Initialize θ.
- Repeat until convergence:
  - Generate random samples (t, x, p) from $[0, T] \times \Omega \times \mathcal{P}$, $(\tilde{t}, \tilde{x})$ from $[0, T] \times \partial\Omega$, and $\hat{x}$ from Ω.
  - Construct the objective function

$$\begin{aligned} J(\theta) = &\left(\frac{\partial f}{\partial t}(t, x, p; \theta) - \mathcal{L}_p f(t, x, p; \theta) \right)^2 \\ &+ \left(g_p(\tilde{x}) - f(\tilde{t}, \tilde{x}, p; \theta) \right)^2 \\ &+ \left(h_p(\hat{x}) - f(0, \hat{x}, p; \theta) \right)^2. \end{aligned} \tag{6.2}$$

  - Update θ with a stochastic gradient descent step

$$\theta \longrightarrow \theta - \alpha \nabla_\theta J(\theta), \tag{6.3}$$

  where α is the learning rate.

If x is low-dimensional ($d \leq 3$), which is common in many physical PDEs, the first and second partial derivatives of f can be calculated via chain rule or approximated by finite difference. We implement our algorithm for Burgers' equation on a finite domain.

$$\begin{aligned} &\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - \alpha u \frac{\partial u}{\partial x}, \quad (t, x) \in [0, 1] \times [0, 1], \\ &u(t, x = 0) = a, \\ &u(t, x = 1) = b, \\ &u(t = 0, x) = g(x), \quad x \in [0, 1]. \end{aligned}$$

The problem setup space is $\mathcal{P} = (\nu, \alpha, a, b) \in \mathbb{R}^4$. The initial condition $g(x)$ is chosen to be a linear function which matches the boundary conditions $u(t, x = 0) = a$ and $u(t, x = 1) = b$. We train a *single neural network* to approximate the solution of $u(t, x; p)$ over the entire space $(t, x, \nu, \alpha, a, b) \in [0, 1] \times [0, 1] \times [10^{-2}, 10^{-1}] \times [10^{-2}, 1] \times [-1, 1] \times [-1, 1]$. We

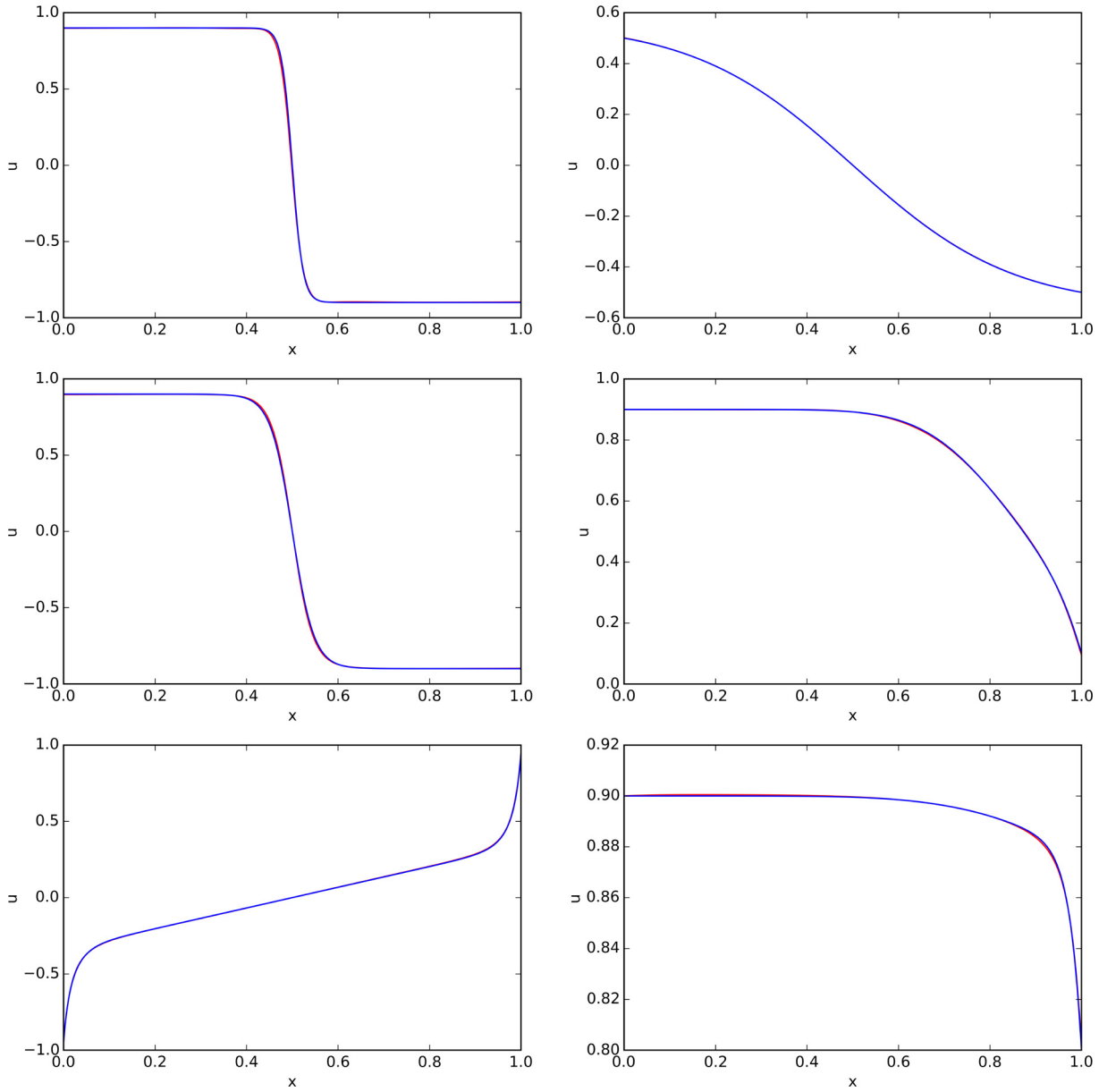

Fig. 5. The deep learning solution is in red. The "exact solution", found via finite difference, is in blue. Solutions are reported at time $t = 1$. The solutions are very close; generally, the two solutions are visibly indistinguishably. The problem setups, in counter-clockwise order, are $(\nu, \alpha, a, b) = (0.01, 0.95, 0.9, -0.9)$, $(0.02, 0.95, 0.9, -0.9)$, $(0.01, 0.95, -0.95, 0.95)$, $(0.02, 0.9, 0.9, 0.8)$, $(0.01, 0.75, 0.9, 0.1)$, and $(0.09, 0.95, 0.5, -0.5)$.

use a larger network (6 layers, 200 units per layer) than in the previous numerical examples. Fig. 5 compares the deep learning solution with the exact solution for several different problem setups p. The solutions are very close; generally, the two solutions are visibly indistinguishable. The deep learning algorithm is able to accurately capture the shock layers and boundary layers.

Fig. 6 presents the accuracy of the deep learning algorithm for different times t and different choices of ν. As ν becomes smaller, the solution becomes steeper. It also shows the shock layer forming over time. The contour plot (Fig. 7) reports the absolute error of the deep learning solution for different choices of b and ν.

7. Neural network approximation theorem for PDEs

Let the L^2 error $J(f)$ measure how well the neural network f satisfies the differential operator, boundary condition, and initial condition. Define $\mathfrak{C}^n$ as the class of neural networks with n hidden units and let f^n be a neural network with n hidden units which minimizes $J(f)$. We prove that

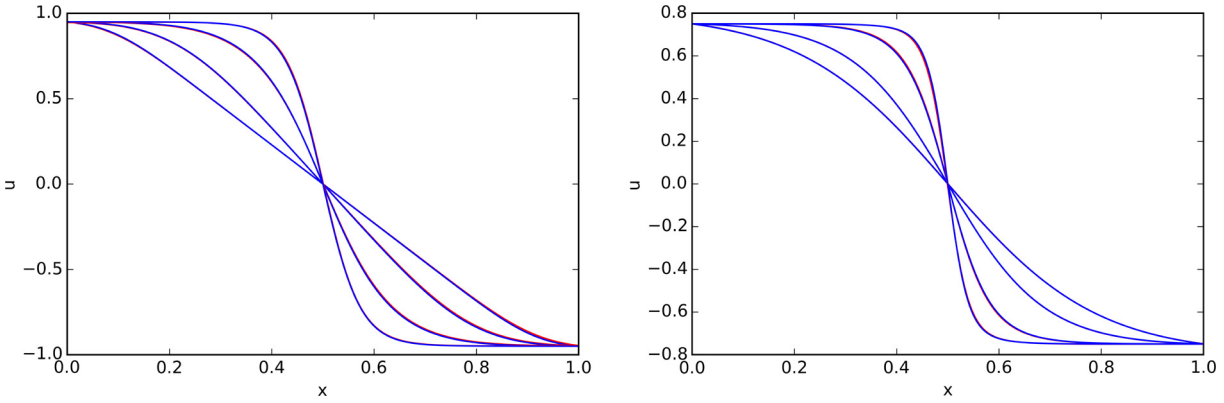

Fig. 6. The deep learning solution is in red. The "exact solution", found via finite difference, is in blue. **Left plot:** Comparison of solutions at times $t = 0.1, 0.25, 0.5, 1$ for $(\nu, \alpha, a, b) = (0.03, 0.9, 0.95, -0.95)$. **Right plot:** Comparison of solutions for $\nu = 0.01, 0.02, 0.05, 0.09$ at time $t = 1$ and with $(\alpha, a, b) = (0.8, 0.75, -0.75)$.

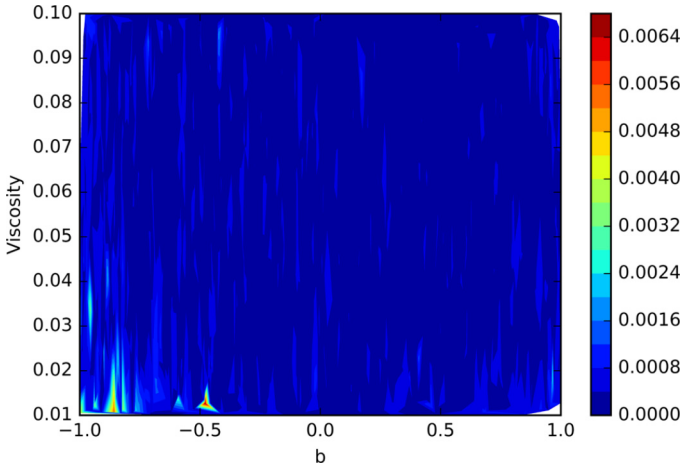

Fig. 7. Contour plot of the average absolute error of the deep learning solution for different b and ν (the viscosity). The absolute error is averaged across $x \in [0, 1]$ for time $t = 1$.

there exists $f^n \in \mathfrak{C}^n$ such that $J(f^n) \to 0$, as $n \to \infty$, and

$f^n \to u \quad \text{as} \quad n \to \infty,$

in the appropriate sense, for a class of quasilinear parabolic PDEs with the principle term in divergence form under certain growth and smoothness assumptions on the nonlinear terms. Our theoretical result only covers a class of quasilinear parabolic PDEs as described in this section. However, the numerical results of this paper indicate that the results are more broadly applicable.

The proof requires the joint analysis of the approximation power of neural networks as well as the continuity properties of partial differential equations. First, we show that the neural network can satisfy the differential operator, boundary condition, and initial condition arbitrarily well for sufficiently large n.

$$J(f^n) \to 0 \quad \text{as} \quad n \to \infty. \tag{7.1}$$

Let u be the solution to the PDE. The statement (7.1) does not necessarily imply that $f^n \to u$. One challenge to proving convergence is that we only have L^2 control of the error. We prove convergence for the case of homogeneous boundary data, i.e., $g(t, x) = 0$, by first establishing that each neural network $\{f^n\}_{n=1}^{\infty}$ satisfies a PDE with a source term $h^n(t, x)$. Importantly, the source terms $h^n(t, x)$ are only known to be vanishing in L^2. We are then able to prove that the convergence of $f^n \to u$ as $n \to \infty$ in the appropriate space holds using compactness arguments.

The precise statement of the theorem and the presentation of the proof is in the next two sections. Section 7.1 proves that $J(f^n) \to 0$ as $n \to \infty$. Section 7.2 contains convergence results of f^n to the solution u of the PDE as $n \to \infty$. The main result is Theorem 7.3. For readability purposes the corresponding proofs are in Appendix A.

7.1. Convergence of the L^2 error $J(f)$

In this section, we present a theorem guaranteeing the existence of multilayer feed forward networks f able to universally approximate solutions of quasilinear parabolic PDEs in the sense that there is f that makes the objective function $J(f)$ arbitrarily small. To do so, we use the results of [26] on universal approximation of functions and their derivatives and make appropriate assumptions on the coefficients of the PDEs to guarantee that a classical solution exists (since then the results of [26] apply).

Consider a bounded set $\Omega \subset \mathbb{R}^d$ with a smooth boundary $\partial\Omega$ and denote $\Omega_T = (0,T] \times \Omega$ and $\partial\Omega_T = (0,T] \times \partial\Omega$. In this subsection we consider the class of quasilinear parabolic PDE's of the form

$$\begin{aligned} &\partial_t u(t,x) - \operatorname{div}\left(\alpha(t,x,u(t,x),\nabla u(t,x))\right) + \gamma(t,x,u(t,x),\nabla u(t,x)) = 0, \text{ for } (t,x) \in \Omega_T \\ &u(0,x) = u_0(x), \text{ for } x \in \Omega \\ &u(t,x) = g(t,x), \text{ for } (t,x) \in \partial\Omega_T \end{aligned} \tag{7.2}$$

For notational convenience, let us write the operator of (7.2) as $\mathcal{G}$. Namely, let us denote

$$\mathcal{G}[u](t,x) = \partial_t u(t,x) - \operatorname{div}\left(\alpha(t,x,u(t,x),\nabla u(t,x))\right) + \gamma(t,x,u(t,x),\nabla u(t,x)).$$

Notice that we can write

$$\mathcal{G}[u](t,x) = \partial_t u(t,x) - \sum_{i,j=1}^{d} \frac{\partial \alpha_i(t,x,u(t,x),\nabla u(t,x))}{\partial u_{x_j}} \partial_{x_i,x_j} u(t,x) + \hat{\gamma}(t,x,u(t,x),\nabla u(t,x)),$$

where

$$\hat{\gamma}(t,x,u,p) = \gamma(t,x,u,p) - \sum_{i=1}^{d} \frac{\partial \alpha_i(t,x,u,p)}{\partial u} \partial_{x_i} u - \sum_{i=1}^{d} \frac{\partial \alpha_i(t,x,u,p)}{\partial x_i}.$$

For the purposes of this section, we consider equations of the type (7.2) that have classical solutions.

In particular we assume that there is a unique $u(t,x)$ solving (7.2) such that

$$u(t,x) \in \mathcal{C}(\bar{\Omega}_T) \bigcap \mathcal{C}^{1+\eta/2,2+\eta}(\Omega_T) \text{ with } \eta \in (0,1) \text{ and that } \sup_{(t,x)\in\Omega_T} \sum_{k=1}^{2} |\nabla_x^{(k)} u(t,x)| < \infty. \tag{7.3}$$

We refer the interested reader to Theorems 5.4, 6.1 and 6.2 of Chapter V in [28] for specific general conditions on α, γ guaranteeing the validity of the aforementioned statement.

Universal approximation results for single functions and their derivatives have been obtained under various assumptions in [11,25,26]. In this paper, we use Theorem 3 of [26]. Let us recall the setup appropriately modified for our case of interest. Let ψ be an activation function, e.g., of sigmoid type, of the hidden units and define the set

$$\mathfrak{C}^n(\psi) = \left\{ \zeta(t,x) : \mathbb{R}^{1+d} \mapsto \mathbb{R} : \zeta(t,x) = \sum_{i=1}^{n} \beta_i \psi\left(\alpha_{1,i} t + \sum_{j=1}^{d} \alpha_{j,i} x_j + c_j\right) \right\}, \tag{7.4}$$

where $\theta = \left(\beta_1, \cdots, \beta_n, \alpha_{1,1}, \cdots, \alpha_{d,n}, c_1, c_1, \cdots, c_n\right) \in \mathbb{R}^{2n+n(1+d)}$ compose the elements of the parameter space. Then we have the following result.

Theorem 7.1. *Let $\mathfrak{C}^n(\psi)$ be given by (7.4) where ψ is assumed to be in $\mathcal{C}^2(\mathbb{R}^d)$, bounded and non-constant. Set $\mathfrak{C}(\psi) = \bigcup_{n\geq 1} \mathfrak{C}^n(\psi)$. Assume that Ω_T is compact and consider the measures ν_1, ν_2, ν_3 whose support is contained in Ω_T, Ω and $\partial\Omega_T$ respectively. In addition, assume that the PDE (7.2) has a unique classical solution such that (7.3) holds. Also, assume that the nonlinear terms $\frac{\partial \alpha_i(t,x,u,p)}{\partial p_j}$ and $\hat{\gamma}(t,x,u,p)$ are locally Lipschitz in (u,p) with Lipschitz constant that can have at most polynomial growth on u and p, uniformly with respect to t,x. Then, for every $\epsilon > 0$, there exists a positive constant $K > 0$ that may depend on $\sup_{\Omega_T} |u|$, $\sup_{\Omega_T} |\nabla_x u|$ and $\sup_{\Omega_T} |\nabla_x^{(2)} u|$ such that there exists a function $f \in \mathfrak{C}(\psi)$ that satisfies*

$$J(f) \leq K\epsilon.$$

The proof of this theorem is in the Appendix.

7.2. Convergence of the neural network to the PDE solution

We now prove, under stronger conditions, the convergence of the neural networks f^n to the solution u of the PDE

$$\begin{aligned} &\partial_t u(t,x) - \operatorname{div}(\alpha(t,x,u(t,x),\nabla u(t,x))) + \gamma(t,x,u(t,x),\nabla u(t,x)) = 0, \text{ for } (t,x) \in \Omega_T \\ &u(0,x) = u_0(x), \text{ for } x \in \Omega \\ &u(t,x) = 0, \text{ for } (t,x) \in \partial\Omega_T, \end{aligned} \tag{7.5}$$

as $n \to \infty$. Notice that we have restricted the discussion to homogeneous boundary data. We do this for both presentation and mathematical reasons.[4]

The objective function is

$$J(f) = \|\mathcal{G}[f]\|^2_{2,\Omega_T} + \|f\|^2_{2,\partial\Omega_T} + \|f(0,\cdot) - u_0\|^2_{2,\Omega}$$

Recall that the norms above are $L^2(X)$ norms in the respective space $X = \Omega_T, \partial\Omega_T$ and Ω respectively. From Theorem 7.1, we have that

$$J(f^n) \to 0 \text{ as } n \to \infty.$$

Each neural network f^n satisfies the PDE

$$\begin{aligned} &\mathcal{G}[f^n](t,x) = h^n(t,x), \text{ for } (t,x) \in \Omega_T \\ &f^n(0,x) = u_0^n(x), \text{ for } x \in \Omega \\ &f^n(t,x) = g^n(t,x), \text{ for } (t,x) \in \partial\Omega_T \end{aligned} \tag{7.6}$$

for some h^n, u_0^n, and g^n such that

$$\left\|h^n\right\|^2_{2,\Omega_T} + \left\|g^n\right\|^2_{2,\partial\Omega_T} + \left\|u_0^n - u_0\right\|^2_{2,\Omega} \to 0 \text{ as } n \to \infty. \tag{7.7}$$

For the purposes of this section, we make the following set of assumptions.

Condition 7.2.

- There is a constant $\mu > 0$ and positive functions $\kappa(t,x), \lambda(t,x)$ such that for all $(t,x) \in \Omega_T$ we have

  $$\|\alpha(t,x,u,p)\| \le \mu(\kappa(t,x) + \|p\|), \text{ and } |\gamma(t,x,u,p)| \le \lambda(t,x)\|p\|,$$

  with $\kappa \in L^2(\Omega_T)$, $\lambda \in L^{d+2+\eta}(\Omega_T)$ for some $\eta > 0$.
- $\alpha(t,x,u,p)$ and $\gamma(t,x,u,p)$ are Lipschitz continuous in $(t,x,u,p) \in \Omega_T \times \mathbb{R} \times \mathbb{R}^d$ uniformly on compacts of the form $\{(t,x) \in \bar{\Omega}_T, |u| \le C, |p| \le C\}$.
- $\alpha(t,x,u,p)$ is differentiable with respect to (x,u,p) with continuous derivatives.
- There is a positive constant $\nu > 0$ such that

  $$\alpha(t,x,u,p)p \ge \nu|p|^2$$

  and

  $$\langle \alpha(t,x,u,p_1) - \alpha(t,x,u,p_2), p_1 - p_2 \rangle > 0, \text{ for every } p_1, p_2 \in \mathbb{R}^d, p_1 \ne p_2.$$

- $u_0(x) \in \mathcal{C}^{0,2+\xi}(\bar{\Omega})$ for some $\xi > 0$[5] with itself and its first derivative bounded in $\bar{\Omega}$.
- Ω is a bounded, open subset of $\mathbb{R}^d$ with boundary $\partial\Omega \in \mathcal{C}^2$.
- For every $n \in \mathbb{N}$, $f^n \in \mathcal{C}^{1,2}(\bar{\Omega}_T)$. In addition, $(f^n)_{n\in\mathbb{N}} \in L^2(\Omega_T)$.

[4] We set $u(t,x) = 0$, for $(t,x) \in \partial\Omega_T$, i.e., $g = 0$, to circumvent certain technical difficulties arising due to inhomogeneous boundary conditions. If $g \ne 0$ such that g is the trace of some appropriately smooth function, say ϕ, then one can reduce the inhomogeneous boundary conditions on $\partial\Omega_T$ to the homogeneous one by introducing in place of u the new function $u - \phi$, see Section 4 of Chapter V in [28] or Chapter 8 of [20] for details on such considerations. We do not explore this here, because our goal is not to prove the most general result possible, but to provide a concrete setup in which we can prove the validity of the approximation results of interest.

[5] In general, the Hölder space $\mathcal{C}^{0,\xi}(\bar{\Omega})$ is the Banach space of continuous functions in $\bar{\Omega}$ having continuous derivatives up to order $[\xi]$ in $\bar{\Omega}$ with finite corresponding uniform norms and finite uniform $\xi - [\xi]$ Hölder norm. Analogously, we also define the Hölder space $\mathcal{C}^{0,\xi,\xi/2}(\bar{\Omega}_T)$ which in addition has finite $[\xi]/2$ and $(\xi - [\xi])/2$ regular and Hölder derivatives norms in time respectively. These spaces are denoted by $H^{\xi}(\bar{\Omega})$ and $H^{\xi,\xi/2}(\bar{\Omega}_T)$ respectively in [28].

Theorem 7.3. *Assume that Condition 7.2 and (7.7) hold. Then, problem (7.5) has a unique bounded solution in* $\mathcal{C}^{0,\delta,\delta/2}(\bar{\Omega}_T) \cap L^2\left(0,T;W_0^{1,2}(\Omega)\right) \cap W_0^{(1,2),2}(\Omega'_T)$ *for some* $\delta > 0$ *and any interior subdomain* Ω'_T *of* Ω_T.[6] *In addition,* f^n *converges to* u*, the unique solution to (7.5), strongly in* $L^\rho(\Omega_T)$ *for every* $\rho < 2$*. If, in addition, the sequence* $\{f^n(t,x)\}_{n\in\mathbb{N}}$ *is uniformly bounded in* n *and equicontinuous then the convergence to* u *is uniform in* Ω_T*.*

The proof of this theorem is in the Appendix. We conclude this section with some remarks and an example.

Remark 7.4. Despite the restriction made to the zero boundary data case, we do expect that our results are also valid for reasonably smooth inhomogeneous boundary data. In addition, if we make further assumptions on the nonlinearities $\alpha(t,x,u,p)$ and $\gamma(t,x,u,p)$ and on the initial data $u_0(x)$, then one can establish existence and uniqueness of classical solutions, see for example Section 6 of Chapter V in [28] for details. As a matter of fact the results of Chapter V.6 in [28] show that with assuming a little bit more on the growth of the derivatives of the nonlinear functions $\alpha(t,x,u,p), \gamma(t,x,u,p)$ will lead to $\nabla_x u \in \mathcal{C}^{0,\delta',\delta'/2}(\Omega_T)$ for some $\delta' > 0$. Furthermore, we remark here that stronger claims can be made if more properties are known in regard to the given approximating family $\{f^n\}$ such as, for example, a-priori bounds on appropriate Sobolev norms, but we do not explore this further here.

Remark 7.5. The uniform, in n, L^2 bound for the sequence $\{f^n\}_{n\in\mathbb{N}}$ is easily satisfied for a bounded neural network approximation sequence $f^n(t,x)$. However, we believe that it is true for a wider class of models, after all one expects that to be true if f^n indeed converges in L^ρ for $\rho < 2$. The condition on equicontinuity for $\{f^n(t,x)\}$ allows to both simplify the proof and make a stronger claim as well. However, it is only a sufficient condition and not necessary. The paper, [8], see Theorems 19 and 20 therein, discusses structural restrictions (a-priori boundedness and summability) that can be imposed on the unknown weights of feedforward neural networks, belonging in the class $\mathfrak{C}(\psi) = \bigcup_{n\geq 1}\mathfrak{C}^n(\psi)$ as defined by (7.4), which then guarantee both equicontinuity and universal approximation properties of the neural network for continuous and bounded functions. As it is also discussed in [8], equicontinuity is also related to fault-tolerance properties of neural networks, a subject worthy of further study in the context of PDEs. However, we do not discuss this further here as this would be a topic for a different paper.

Let us present the case of linear parabolic PDEs in Example 7.6 below.

Example 7.6 *(Linear case).* Let us assume that the operator $\mathcal{G}$ is linear in u and ∇u. In particular, let us set

$$\alpha_i(t,x,u,p) = \sum_{j=1}^{n}\left(\sigma\sigma^T\right)_{i,j}(t,x)p_j, i = 1,\cdots d$$

and

$$\gamma(t,x,u,p) = -\langle b(t,x), p\rangle + \sum_{i,j=1}^{d}\frac{\partial}{\partial x_i}\left(\sigma\sigma^T\right)_{i,j}(t,x)p_j - c(t,x)u.$$

Assume that there are positive constants $\nu, \mu > 0$ such that for every $\xi \in \mathbb{R}^d$ the matrix $\left[\left(\sigma\sigma^T\right)_{i,j}(t,x)\right]_{i,j=1}^{d}$ satisfies

$$\nu|\xi|^2 \leq \sum_{i,j=1}^{d}\left(\sigma\sigma^T\right)_{i,j}(t,x)\xi_i\xi_j \leq \mu|\xi|^2$$

and that the coefficients b and c are such that

$$\left\|\sum_{i=1}^{d} b_i^2\right\|_{q,r,\Omega_T} + \|c\|_{q,r,\Omega_T} \leq \mu, \text{ for some } \mu > 0$$

where we recall for example $\|c\|_{q,r,\Omega_T} = \left(\int_0^T\left(\int_\Omega |c(t,x)|^q dx\right)^{r/q}\right)^{1/r}$ and r, q satisfy the relations

$$\frac{1}{r} + \frac{d}{2q} = 1$$
$$q \in (d/2,\infty], r \in [1,\infty), \text{ for } d \geq 2,$$
$$q \in [1,\infty], r \in [1,2], \text{ for } d = 1.$$

[6] Here $W_0^{(1,2),2}(\Omega'_T)$ denotes the Banach space which is the closure of $C_0^\infty(\Omega'_T)$ with elements from $L^2(\Omega'_T)$ having generalized derivatives of the form $D_t^r D_x^s$ with r, s such that $2r + s \leq 2$ with the usual Sobolev norm.

In particular, the previous bounds always hold in the case of coefficients b and c that are bounded in Ω_T. Under these conditions, standard results for linear PDE's, see for instance Theorem 4.5 of Chapter III of [28] for a related result, show that approximation results analogous to that of Theorem 7.3 hold.

8. Conclusion

We believe that deep learning could become a valuable approach for solving high-dimensional PDEs, which are important in physics, engineering, and finance. The PDE solution can be approximated with a deep neural network which is trained to satisfy the differential operator, initial condition, and boundary conditions. We prove that the neural network converges to the solution of the partial differential equation as the number of hidden units increases.

Our deep learning algorithm for solving PDEs is meshfree, which is key since meshes become infeasible in higher dimensions. Instead of forming a mesh, the neural network is trained on batches of randomly sampled time and space points. The approach is implemented for a class of high-dimensional free boundary PDEs in up to 200 dimensions with accurate results. We also test it on a high-dimensional Hamilton–Jacobi–Bellman PDE with accurate results.

The DGM algorithm can be easily modified to apply to hyperbolic, elliptic, and partial–integral differential equations. The algorithm remains essentially the same for these other types of PDEs. However, numerical performance for these other types of PDEs remains to be investigated.

It is also important to put the numerical results in Sections 4, 5 and 6 in a proper context. PDEs with highly non-monotonic or oscillatory solutions may be more challenging to solve and further developments in architecture will be necessary. Further numerical development and testing is therefore required to better judge the usefulness of deep learning for the solution of PDEs in other applications. However, the numerical results of this paper demonstrate that there is sufficient evidence to further explore deep neural network approaches for solving PDEs.

In addition, it would be of interest to establish results analogous to Theorem 7.3 for PDEs beyond the class of quasilinear parabolic PDEs considered in this paper. Stability analysis of deep learning and machine learning algorithms for solving PDEs is also an important question. It would certainly be interesting to study machine learning algorithms that use a more direct variational formulation of the involved PDEs. We leave these questions for future work.

Appendix A. Proofs of convergence results

In this section we have gathered the proofs of the theoretical results of Section 7.

Proof of Theorem 7.1. By Theorem 3 of [26] we know that there is a function $f \in \mathfrak{C}(\psi)$ that is uniformly 2-dense on compacts of $\mathcal{C}^2(\mathbb{R}^{1+d})$. This means that for $u \in \mathcal{C}^{1,2}([0,T]\times\mathbb{R}^d)$ and $\epsilon > 0$, there is $f \in \mathfrak{C}(\psi)$ such that

$$\sup_{(t,x)\in\Omega_T} |\partial_t u(t,x) - \partial_t f(t,x;\theta)| + \max_{|a|\le 2} \sup_{(t,x)\in\bar{\Omega}_T} |\partial_x^{(a)} u(t,x) - \partial_x^{(a)} f(t,x;\theta)| < \epsilon \tag{A.1}$$

We have assumed that $(u,p) \mapsto \hat{\gamma}(t,x,u,p)$ is locally Lipschitz continuous in (u,p) with Lipschitz constant that can have at most polynomial growth in u and p, uniformly with respect to t,x. This means that

$$\left|\hat{\gamma}(t,x,u,p) - \hat{\gamma}(t,x,v,s)\right| \le \left(|u|^{q_1/2} + |p|^{q_2/2} + |v|^{q_3/2} + |s|^{q_4/2}\right)(|u-v| + |p-s|),$$

for some constants $0 \le q_1, q_2, q_3, q_4 < \infty$. Therefore we obtain, using Hölder inequality with exponents r_1, r_2,

$$\begin{aligned}
&\int_{\Omega_T} \left|\hat{\gamma}(t,x,f,\nabla_x f) - \hat{\gamma}(t,x,u,\nabla_x u)\right|^2 d\nu_1(t,x) \le \\
&\le \int_{\Omega_T} \left(|f(t,x;\theta)|^{q_1} + |\nabla_x f(t,x;\theta)|^{q_2} + |u(t,x)|^{q_3} + |\nabla_x u(t,x)|^{q_4}\right) \\
&\qquad \times \left(|f(t,x;\theta) - u(t,x)|^2 + |\nabla_x f(t,x;\theta) - \nabla_x u(t,x)|^2\right) d\nu_1(t,x) \\
&\le \left(\int_{\Omega_T} \left(|f(t,x;\theta)|^{q_1} + |\nabla_x f(t,x;\theta)|^{q_2} + |u(t,x)|^{q_3} + |\nabla_x u(t,x)|^{q_4}\right)^{r_1} d\nu_1(t,x)\right)^{1/r_1} \\
&\qquad \times \left(\int_{\Omega_T} \left(|f(t,x;\theta) - u(t,x)|^2 + |\nabla_x f(t,x;\theta) - \nabla_x u(t,x)|^2\right)^{r_2} d\nu_1(t,x)\right)^{1/r_2}
\end{aligned}$$

$$\leq K \left(\int_{\Omega_T} \left(|f(t,x;\theta) - u(t,x)|^{q_1} + |\nabla_x f(t,x;\theta) - \nabla_x u(t,x)|^{q_2} + |u(t,x)|^{q_1 \vee q_3} + |\nabla_x u(t,x)|^{q_2 \vee q_4} \right)^{r_1} d\nu_1(t,x) \right)^{1/r_1}$$
$$\times \left(\int_{\Omega_T} \left(|f(t,x;\theta) - u(t,x)|^2 + |\nabla_x f(t,x;\theta) - \nabla_x u(t,x)|^2 \right)^{r_2} d\nu_1(t,x) \right)^{1/r_2}$$
$$\leq K \left(\epsilon^{q_1} + \epsilon^{q_2} + \sup_{\Omega_T} |u|^{q_1 \vee q_3} + \sup_{\Omega_T} |\nabla_x u|^{q_2 \vee q_4} \right) \epsilon^2 \tag{A.2}$$

where the unimportant constant $K < \infty$ may change from line to line and for two numbers $q_1 \vee q_3 = \max\{q_1, q_3\}$. In the last step we used (A.1).

In addition, we have also assumed that for every $i, j \in \{1, \cdots d\}$, the mapping $(u,p) \mapsto \frac{\partial \alpha_i(t,x,u,p)}{\partial p_j}$ is locally Lipschitz in (u,p) with Lipschitz constant that can have at most polynomial growth on u and p, uniformly with respect to t, x. This means that

$$\left| \frac{\partial \alpha_i(t,x,u,p)}{\partial p_j} - \frac{\partial \alpha_i(t,x,v,s)}{\partial s_j} \right| \leq \left(|u|^{q_1/2} + |p|^{q_2/2} + |v|^{q_3/2} + |s|^{q_4/2} \right) (|u-v| + |p-s|),$$

for some constants $0 \leq q_1, q_2, q_3, q_4 < \infty$. Denote for convenience

$$\xi(t,x,u,\nabla u, \nabla^2 u) = \sum_{i,j=1}^{d} \frac{\partial \alpha_i(t,x,u(t,x),\nabla u(t,x))}{\partial u_{x_j}} \partial_{x_i,x_j} u(t,x).$$

Then, similarly to (A.2) we have after an application of Hölder inequality, for some constant $K < \infty$ that may change from line to line,

$$\int_{\Omega_T} \left| \xi(t,x,f,\nabla_x f, \nabla_x^2 f) - \xi(t,x,u,\nabla_x u, \nabla_x^2 u) \right|^2 d\nu_1(t,x) \leq$$
$$\leq \int_{\Omega_T} \left| \sum_{i,j=1}^{d} \left(\frac{\partial \alpha_i(t,x,f(t,x;\theta),\nabla f(t,x;\theta))}{\partial f_{x_j}} - \frac{\partial \alpha_i(t,x,u(t,x),\nabla u(t,x))}{\partial u_{x_j}} \right) \partial_{x_i,x_j} u(t,x) \right|^2 d\nu_1(t,x)$$
$$+ \int_{\Omega_T} \left| \sum_{i,j=1}^{d} \frac{\partial \alpha_i(t,x,f(t,x;\theta),\nabla f(t,x;\theta))}{\partial f_{x_j}} \left(\partial_{x_i,x_j} f(t,x;\theta) - \partial_{x_i,x_j} u(t,x) \right) \right|^2 d\nu_1(t,x)$$
$$\leq K \sum_{i,j=1}^{d} \left(\int_{\Omega_T} \left| \partial_{x_i,x_j} u(t,x) \right|^{2p} d\nu_1(t,x) \right)^{1/p} \times$$
$$\times \left(\int_{\Omega_T} \left| \frac{\partial \alpha_i(t,x,f(t,x;\theta),\nabla f(t,x;\theta))}{\partial f_{x_j}} - \frac{\partial \alpha_i(t,x,u(t,x),\nabla u(t,x))}{\partial u_{x_j}} \right|^{2q} d\nu_1(t,x) \right)^{1/q} +$$
$$+ K \sum_{i,j=1}^{d} \left(\int_{\Omega_T} \left| \frac{\partial \alpha_i(t,x,f,\nabla f)}{\partial f_{x_j}} \right|^{2p} d\nu_1(t,x) \right)^{1/p} \left(\int_{\Omega_T} \left| \partial_{x_i,x_j} f(t,x;\theta) - \partial_{x_i,x_j} u(t,x) \right|^{2q} d\nu_1(t,x) \right)^{1/q}$$
$$\leq K \sum_{i,j=1}^{d} \left(\int_{\Omega_T} \left| \partial_{x_i,x_j} u(t,x) \right|^{2p} d\nu_1(t,x) \right)^{1/p} \times$$
$$\times \left(\int_{\Omega_T} \left(|f(t,x;\theta) - u(t,x)|^{q_1} + |\nabla_x f(t,x;\theta) - \nabla_x u(t,x)|^{q_2} + |u(t,x)|^{q_1 \vee q_3} + |\nabla_x u(t,x)|^{q_2 \vee q_4} \right)^{qr_1} d\nu_1(t,x) \right)^{1/(qr_1)}$$

$$\times \left(\int_{\Omega_T} \left(|f(t,x;\theta) - u(t,x)|^2 + |\nabla_x f(t,x;\theta) - \nabla_x u(t,x)|^2 \right)^{qr_2} d\nu_1(t,x) \right)^{1/(qr_2)}$$

$$+ K \sum_{i,j=1}^{d} \left(\int_{\Omega_T} \left| \frac{\partial \alpha_i(t,x,f,\nabla f)}{\partial f_{x_j}} \right|^{2p} d\nu_1(t,x) \right)^{1/p} \left(\int_{\Omega_T} \left| \partial_{x_i,x_j} f(t,x;\theta) - \partial_{x_i,x_j} u(t,x) \right|^{2q} d\nu_1(t,x) \right)^{1/q}$$

$$\leq K\epsilon^2, \tag{A.3}$$

where in the last step we followed the computation in (A.2) and used (A.1).

Using (A.1) and (A.2)–(A.3) we subsequently obtain for the objective function (note that $\mathcal{G}[u](t,x) = 0$ for u that solves the PDE)

$$\begin{aligned}
J(f) &= \|\mathcal{G}[f](t,x)\|^2_{\Omega_T,\nu_1} + \|f(t,x;\theta) - g(t,x)\|^2_{\partial\Omega_T,\nu_2} + \|f(0,x;\theta) - u_0(x)\|^2_{\Omega,\nu_3} \\
&= \|\mathcal{G}[f](t,x) - \mathcal{G}[u](t,x)\|^2_{\Omega_T,\nu_1} + \|f(t,x;\theta) - g(t,x)\|^2_{\partial\Omega_T,\nu_2} + \|f(0,x;\theta) - u_0(x)\|^2_{\Omega,\nu_3} \\
&\leq \int_{\Omega_T} |\partial_t u(t,x) - \partial_t f(t,x;\theta)|^2 d\nu_1(t,x) + \int_{\Omega_T} \left| \xi(t,x,f,\nabla f,\nabla^2 f) - \xi(t,x,u,\nabla u,\nabla^2 u) \right|^2 d\nu_1(t,x) \\
&+ \int_{\Omega_T} \left| \gamma(t,x,f,\nabla_x f) - \gamma(t,x,u,\nabla_x u) \right|^2 d\nu_1(t,x) + \int_{\partial\Omega_T} |f(t,x;\theta) - u(t,x)|^2 d\nu_2(t,x) + \\
&\quad + \int_{\Omega} |f(0,x;\theta) - u(0,x)|^2 d\nu_3(t,x) \\
&\leq K\epsilon^2
\end{aligned}$$

for an appropriate constant $K < \infty$. The last step completes the proof of the Theorem after rescaling ϵ. □

Proof of Theorem 7.3. Existence, regularity and uniqueness for (7.5) follows from Theorem 2.1 [40] combined with Theorems 6.3–6.5 of Chapter V.6 in [28] (see also Theorem 6.6 of Chapter V.6 of [28]). Boundedness follows from Theorem 2.1 in [40] and Chapter V.2 in [28]. The convergence proof follows by the smoothness of the neural networks together with compactness arguments as we explain below.

Let us first consider problem (7.6) with $g^n(t,x) = 0$ and let us denote the solution to this problem by $\hat{f}^n(t,x)$. Due to Condition 7.2, Lemma 4.1 of [40] applies and gives that $\{\hat{f}^n\}_{n\in\mathbb{N}}$ is uniformly bounded with respect to n in at least $L^\infty\left(0,T;L^2(\Omega)\right) \cap L^2\left(0,T;W_0^{1,2}(\Omega)\right)$ (in regard to such uniform energy bound results we also refer the reader to Theorem 2.1 and Remark 2.14 of [5] for the case $\gamma = 0$ and to [34,37] for related results in more general cases). As a matter of fact $\hat{f}^n$ is more regular than stated, see Section 6, Chapter V of [28], but we will not make use of this fact in the convergence proof of $\hat{f}^n$ to u. These uniform energy bounds imply that we can extract a subsequence, denoted also by $\{\hat{f}^n\}_{n\in\mathbb{N}}$, which converges to some u in the weak-* sense in $L^\infty\left(0,T;L^2(\Omega)\right)$ and weakly in $L^2\left(0,T;W_0^{1,2}(\Omega)\right)$ and to some v weakly in $L^2(\Omega)$ for every fixed $t \in (0,T]$.

Next let us set $q = 1 + \frac{d}{d+4} \in (1,2)$ and note that for conjugates, $r_1, r_2 > 1$ such that $1/r_1 + 1/r_2 = 1$

$$\begin{aligned}
\int_{\Omega_T} \left| \gamma(t,x,\hat{f}^n,\nabla_x \hat{f}^n) \right|^q dtdx &\leq \int_{\Omega_T} |\lambda(t,x)|^q \, |\nabla_x \hat{f}^n(t,x)|^q dtdx \\
&\leq \left(\int_{\Omega_T} |\lambda(t,x)|^{r_1 q} dtdx \right)^{1/r_1} \left(\int_{\Omega_T} |\nabla_x \hat{f}^n(t,x)|^{r_2 q} dtdx \right)^{1/r_2}.
\end{aligned} \tag{A.4}$$

Let us choose $r_2 = 2/q > 1$. Then we calculate $r_1 = \frac{r_2}{r_2 - 1} = \frac{2}{2-q}$. Hence, we have that $r_1 q = d + 2$. Recalling the assumption $\lambda \in L^{d+2}(\Omega_T)$ and the uniform bound on the $\left\| \nabla_x \hat{f}^n \right\|_2$ we subsequently obtain that for $q = 1 + \frac{d}{d+4}$, there is a constant $C < \infty$ such that

$$\int_{\Omega_T} \left| \gamma(t,x,\hat{f}^n,\nabla_x \hat{f}^n) \right|^q dtdx \leq C.$$

The latter estimate together with the growth assumptions on $\alpha(\cdot)$ from Condition 7.2, imply that $\{\partial_t \hat{f}^n\}_{n\in\mathbb{N}}$ is bounded uniformly with respect to n in $L^{1+d/(d+4)}(\Omega_T)$ and in $L^2(0,T;W^{-1,2}(\Omega))$. Consider the conjugates $1/\delta_1 + 1/\delta_2 = 1$ with $\delta_2 > \max\{2,d\}$. Due to the embedding

$$W^{-1,2}(\Omega) \subset W^{-1,\delta_1}(\Omega), \quad L^q(\Omega) \subset W^{-1,\delta_1}(\Omega), \text{ and } L^2(\Omega) \subset W^{-1,\delta_1}(\Omega),$$

we have that $\{\partial_t \hat{f}^n\}_{n\in\mathbb{N}}$ is bounded uniformly with respect to n in $L^1(0,T;W^{-1,\delta_1}(\Omega))$. Define now the spaces $X = W_0^{1,2}(\Omega)$, $B = L^2(\Omega)$ and $Y = W^{-1,\delta_1}(\Omega)$, and notice that

$$X \subset B \subset Y$$

with the first embedding being compact. Then, Corollary 4 of [48] yields relative compactness of $\{\hat{f}^n\}_{n\in\mathbb{N}}$ in $L^2(\Omega_T)$, which means that $\{\hat{f}^n\}_{n\in\mathbb{N}}$ converges strongly to u in that space. Thus, up to subsequences, $\{\hat{f}^n\}_{n\in\mathbb{N}}$ converges almost everywhere to u in Ω_T.

The nonlinearity of the α and γ functions with respect to the gradient prohibits us from passing to the limit directly in the respective weak formulation. However, the uniform boundedness of $\{\hat{f}_n\}_{n\in\mathbb{N}}$ in $L^\sigma\left(0,T;W_0^{1,\sigma}(\Omega)\right)$ with $\sigma > 1$ (in fact here $\sigma = 2$) and its weak convergence to u in that space, allows us to conclude, as in Theorem 3.3 of [4], that

$$\nabla \hat{f}^n \to \nabla u \text{ almost everywhere in } \Omega_T.$$

Hence, we obtain that $\{\hat{f}^n\}_{n\in\mathbb{N}}$ converges to u strongly also in $L^\rho\left(0,T;W_0^{1,\rho}(\Omega)\right)$ for every $\rho < 2$.

In preparation to passing to the limit as $n \to \infty$ in the weak formulation, we need to study the behavior of the nonlinear terms. Recalling the assumptions on $\alpha(t,x,u,p)$ we have for $\rho < 2$ and for a measurable set $A \subset \Omega_T$ (the constant $K < \infty$ may change from line to line)

$$\begin{aligned}
\int_A \left|\alpha(t,x,\hat{f}^n,\nabla\hat{f}^n)\right|^\rho dtdx &\le K\left[\int_A |\kappa(t,x)|^\rho dtdx + \int_A |\nabla\hat{f}^n(t,x)|^\rho dtdx\right] \\
&\le K\left[\int_A |\kappa(t,x)|^\rho dtdx + \left(\int_{\Omega_T} |\nabla\hat{f}^n(t,x)|^2 dtdx\right)^{\rho/2} |A|^{1-\rho/2}\right] \\
&\le K\left[\int_A |\kappa(t,x)|^\rho dtdx + |A|^{1-\rho/2}\right].
\end{aligned}$$

In the latter display we used Höder inequality with exponent $2/\rho > 1$. By Vitali's theorem we then conclude that

$$\alpha(t,x,\hat{f}^n,\nabla\hat{f}^n) \to \alpha(t,x,u,\nabla u) \text{ strongly in } L^\rho(\Omega_T)$$

as $n \to \infty$, for every $1 < \rho < 2$. For the same reason, an analogous estimate to (A.4), gives

$$\int_A \left|\gamma(t,x,\hat{f}^n,\nabla_x\hat{f}^n)\right|^q dtdx \le K\left(\int_A |\lambda(t,x)|^{d+2} dtdx\right)^{(2-q)/2} \le K|A|^{\frac{\eta}{d+2+\eta}}$$

implying, via Vitali's theorem, that

$$\gamma(t,x,\hat{f}^n,\nabla\hat{f}^n) \to \gamma(t,x,u,\nabla u) \text{ strongly in } L^q(\Omega_T)$$

as $n \to \infty$, for $q = 1 + \frac{d}{d+4}$.

Notice also that by construction we have that the initial condition u_0^n converges to u_0 strongly in $L^2(\Omega)$. The weak formulation of the PDE (7.6) with $g^n = 0$ reads as follows. For every $t_1 \in (0,T]$

$$\begin{aligned}
&\int_{\Omega_{t_1}} \left[-\hat{f}^n\partial_t\phi + \left\langle\alpha(t,x,\hat{f}^n,\nabla\hat{f}^n),\nabla\phi\right\rangle + (\gamma(t,x,\hat{f}^n,\nabla\hat{f}^n) - h^n)\phi\right](t,x)dxdt \\
&\quad + \int_\Omega \hat{f}^n(t_1,x)\phi(t_1,x)dx - \int_\Omega u_0^n(x)\phi(0,x)dx = 0
\end{aligned}$$

for every $\phi \in C_0^\infty(\Omega_T)$. Using the above convergence results, we then obtain that the limit point u satisfies for every $t_1 \in (0,T]$ the equation

$$\int_{\Omega_{t_1}} \left[-u\partial_t\phi + \langle \alpha(t,x,u,\nabla u), \nabla\phi\rangle + \gamma(t,x,u,\nabla u)\phi\right](t,x)dxdt + \int_{\Omega} u(t_1,x)\phi(t_1,x)dx - \int_{\Omega} u_0(x)\phi(0,x)dx = 0,$$

which is the weak formulation of equation (7.5).

It remains to discuss the convergence of $f^n - \hat{f}^n$ to zero, where we recall that f^n is the neural network approximation satisfying (7.6) and $\hat{f}^n$ satisfies (7.6) with $g^n = 0$. The functions $f^n \in C^{1,2}(\bar{\Omega}_T)$ and $\bar{\Omega}_T$ is compact. We have also assumed that $\{f^n\}_n$ is uniformly bounded in $L^2(\Omega_T)$. This implies that, up to a subsequence, f^n will converge at least weakly in $L^2(\Omega_T)$. Moreover, the boundary values $g^n(t,x)$ (which is nothing else by $f^n(t,x)$ evaluated at the smooth boundary $\partial\Omega_T$) in (7.6) converge to zero strongly in L^2. It is then a standard result that g^n, i.e., f^n evaluated at the boundary, converges to zero, at least, almost uniformly along a subsequence, see for example Lemma 2.1 in Chapter II of [28]. As it then follows, for example, by the proof of Theorems 6.3–6.4–6.5 of Chapter V.6 in [28], using smoothness and uniqueness, f^n will differ from the solution to the PDE (7.6) with $g^n = 0$, $\hat{f}^n(t,x)$, by a negligible amount as $n \to \infty$ in the almost everywhere sense. The assumed uniform L^2 bound for $\{f^n\}_{n\in\mathbb{N}}$ together with the previously derived uniform $L^2(\Omega_T)$ bound for $\{\hat{f}^n\}_{n\in\mathbb{N}}$ yield uniform $L^2(\Omega_T)$ boundedness for $\{f^n - \hat{f}^n\}_{n\in\mathbb{N}}$. Then, by Vitali's theorem again, we get that $\{f^n - \hat{f}^n\}_{n\in\mathbb{N}}$ goes to zero strongly in $L^\rho(\Omega_T)$ for every $\rho < 2$.

The previously derived strong convergence of $\{f^n - \hat{f}^n\}_{n\in\mathbb{N}}$ to zero in $L^\rho(\Omega_T)$ for every $\rho < 2$, together with the strong $L^2(\Omega_T)$ convergence of $\{\hat{f}^n\}_{n\in\mathbb{N}}$ to u, conclude the proof of the convergence in $L^\rho(\Omega_T)$ for every $\rho < 2$ using triangle inequality.

If $\{f^n(t,x)\}$ is equicontinuous then, Lemma 3.2 of [17] gives uniform convergence of g^n to zero. Hence, by the previous analysis, it will certainly be true that $\{f^n\}_{n\in\mathbb{N}}$ converges to u in $L^\rho(\Omega_T)$ for every $\rho < 2$. The L^ρ convergence to zero together with boundedness and equicontinuity of the sequence $\{f^n(t,x)\}$ results then in uniform convergence due to the well known Arzelà–Ascoli theorem. □

References

[1] S. Asmussen, P. Glynn, Stochastic Simulation: Algorithms and Analysis, Springer, 2007.
[2] C. Beck, W. E, A. Jentzen, Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations, arXiv:1709.05963, 2017.
[3] D. Bertsekas, J. Tsitsiklis, Gradient convergence in gradient methods via errors, SIAM J. Optim. 10 (3) (2000) 627–642.
[4] L. Boccardo, A. Dall'Aglio, T. Gallouët, L. Orsina, Nonlinear parabolic equations with measure data, J. Funct. Anal. 147 (1997) 237–258.
[5] L. Boccardo, M.M. Porzio, A. Primo, Summability and existence results for nonlinear parabolic equations, Nonlinear Anal., Theory Methods Appl. 71 (304) (2009) 1–15.
[6] H. Bungartz, A. Heinecke, D. Pfluger, S. Schraufstetter, Option pricing with a direct adaptive sparse grid approach, J. Comput. Appl. Math. 236 (15) (2012) 3741–3750.
[7] H. Bungartz, M. Griebel, Sparse grids, Acta Numer. 13 (2004) 174–269.
[8] P. Chandra, Y. Singh, Feedforward sigmoidal networks – equicontinuity and fault-tolerance properties, IEEE Trans. Neural Netw. 15 (6) (2004) 1350–1366.
[9] P. Chaudhari, A. Oberman, S. Osher, S. Soatto, G. Carlier, Deep Relaxation: Partial Differential Equations for Optimizing Deep Neural Networks, 2017.
[10] S. Cerrai, Stationary Hamilton–Jacobi equations in Hilbert spaces and applications to a stochastic optimal control problem, SIAM J. Control Optim. 40 (3) (2001) 824–852.
[11] G. Cybenko, Approximation by superposition of a sigmoidal function, Math. Control Signals Syst. 2 (1989) 303–314.
[12] A. Davie, J. Gaines, Convergence of numerical schemes for the solution of parabolic stochastic partial differential equations, Math. Comput. 70 (233) (2000) 121–134.
[13] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. Le, A. Ng, Large scale distributed deep networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1223–1231.
[14] A. Debussche, M. Fuhrman, G. Tessitore, Optimal control of a stochastic heat equation with boundary-noise and boundary-control, ESAIM Control Optim. Calc. Var. 13 (1) (2007) 178–205.
[15] W. E, J. Han, A. Jentzen, Deep Learning-Based Numerical Methods for High-Dimensional Parabolic Partial Differential Equations and Backward Stochastic Differential Equations, Communications in Mathematics and Statistics, Springer, 2017.
[16] M. Fujii, A. Takahashi, M. Takahashi, Asymptotic expansion as prior knowledge in deep learning method for high dimensional BSDEs, arXiv:1710.07030, 2017.
[17] A.M. Garcia, E. Rodemich, H. Rumsey Jr., M. Rosenblatt, A real variable lemma and the continuity of paths of some Gaussian processes, Indiana Univ. Math. J. 20 (6) (December 1970) 565–578.
[18] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.
[19] J. Gaines, Numerical Experiments with SPDEs, London Mathematical Society Lecture Note Series, 1995, pp. 55–71.
[20] D. Gilbarg, N.S. Trudinger, Elliptic Partial Differential Equations of Second Order, second edition, Springer-Verlag, Berlin, Heidelberg, 1983.
[21] I. Gyöngy, Lattice approximations for stochastic quasi-linear parabolic partial differential equations driven by space–time white noise I, Potential Anal. 9 (1) (1998) 1–25.
[22] A. Heinecke, S. Schraufstetter, H. Bungartz, A highly parallel Black–Scholes solver based on adaptive sparse grids, Int. J. Comput. Math. 89 (9) (2012) 1212–1238.
[23] M. Haugh, L. Kogan, Pricing American options: a duality approach, Oper. Res. 52 (2) (2004) 258–270.
[24] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[25] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Netw. 3 (5) (1990) 551–560.
[26] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. 4 (1991) 251–257.
[27] D. Kingma, J. Ba, ADAM: a method for stochastic optimization, arXiv:1412.6980, 2014.

[28] O.A. Ladyzenskaja, V.A. Solonnikov, N.N. Ural'ceva, Linear and Quasi-Linear Equations of Parabolic Type, Translations of Mathematical Monographs Reprint, vol. 23, American Mathematical Society, 1988.
[29] I. Lagaris, A. Likas, D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[30] I. Lagaris, A. Likas, D. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (5) (2000) 1041–1049.
[31] H. Lee, Neural algorithm for solving differential equations, J. Comput. Phys. 91 (1990) 110–131.
[32] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. 807 (2016) 155–166.
[33] F. Longstaff, E. Schwartz, Valuing American options by simulation: a simple least-squares approach, Rev. Financ. Stud. 14 (2001) 113–147.
[34] M. Magliocca, Existence results for a Cauchy–Dirichlet parabolic problem with a repulsive gradient term, Nonlinear Anal. 166 (2018) 102–143.
[35] A. Malek, R. Beidokhti, Numerical solution for high order differential equations using a hybrid neural network-optimization method, Appl. Math. Comput. 183 (1) (2006) 260–271.
[36] F. Masiero, HJB equations in infinite dimensions, J. Evol. Equ. 16 (4) (2016) 789–824.
[37] R. Di Nardo, F. Feo, O. Guibé, Existence result for nonlinear parabolic equations with lower order terms, Anal. Appl. (Singap.) 09 (02) (2011) 161–186.
[38] P. Petersen, F. Voigtlaender, Optimal approximation of piecewise smooth functions using deep ReLU neural networks, arXiv:1709.05289v4, 2017.
[39] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1999) 143–195.
[40] M.M. Porzio, Existence of solutions for some "noncoercive" parabolic equations, Discrete Contin. Dyn. Syst. 5 (3) (1999) 553–568.
[41] M. Raissi, P. Perdikaris, G. Karniadakis, Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations, arXiv: 1711.10561, 2017.
[42] M. Raissi, P. Perdikaris, G. Karniadakis, Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations, arXiv:1711.10566, 2017.
[43] C. Reisinger, G. Wittum, Efficient hierarchical approximation of high-dimensional option pricing problems, SIAM J. Sci. Comput. 29 (1) (2007) 440–458.
[44] C. Reisinger, Analysis of linear difference schemes in the sparse grid combination technique, IMA J. Numer. Anal. 33 (2) (2012) 544–581.
[45] L.C.G. Rogers, Monte-Carlo valuation of American options, Math. Finance 12 (3) (2002) 271–286.
[46] K. Rudd, Solving Partial Differential Equations Using Artificial Neural Networks, PhD Thesis, Duke University, 2013.
[47] R. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, in: Advances in Neural Information Processing Systems, 2015, pp. 2377–2385.
[48] J. Simon, Compact sets in the space $L^p(0, T; B)$, Ann. Mat. Pura Appl. 146 (1987) 65–96.
[49] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin, Accelerating Eulerian fluid simulation with convolutional networks, in: Proceedings of Machine Learning Research, vol. 70, 2017, pp. 3424–3433.