Journal of Computational Physics 470 (2022) 111557

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

FBSDE based neural network algorithms for high-dimensional quasilinear parabolic PDEs

Wenzhong Zhang [a,b], Wei Cai [b,*]

[a] Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou, Jiangsu 215000, China
[b] Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA

ARTICLE INFO

Article history:
Received 6 May 2021
Received in revised form 9 May 2022
Accepted 18 August 2022
Available online 24 August 2022

Keywords:
Forward and backward SDEs
Feynman-Kac formula
Multi-scale deep neural network
Quasilinear parabolic equations
Pardoux-Peng theory

ABSTRACT

In this paper, we propose forward and backward stochastic differential equations (FBSDEs) based deep neural network (DNN) learning algorithms for the solution of high dimensional quasi-linear parabolic partial differential equations (PDEs), which is related to the FBSDEs from the Pardoux-Peng theory. The algorithms rely on a learning process by minimizing the path-wise difference between two discrete stochastic processes, which are defined by the time discretization of the FBSDEs and the DNN representation of the PDE solution, respectively. The proposed algorithms are shown to generate DNN solution for a 100-dimensional Black–Scholes–Barenblatt equation, which is accurate in a finite region in the solution space, and has a convergence rate close to that of the Euler–Maruyama scheme used for discretizing the FBSDEs.

1. Introduction

The relationship between stochastic processes and the solution of partial differential equations is one of the high achievements of probability theory in potential theory research [1], represented by the celebrated Feynman–Kac formula in linear parabolic and elliptic partial differential equations (PDEs) as a result of the Kolmogorov backward equation for the generator of the stochastic process for the former [13] and Dynkin formula for the latter [11]. The recent work by Pardoux–Peng [12] has extended the concept of classic linear Feynman–Kac formula to a nonlinear version, which connects the solution of a quasilinear parabolic PDE to a coupled pair of forward and backward stochastic processes. This extraordinary development has made much impact in many areas including the mathematical finance in option pricing [6].

In particular, in the field of scientific computing, this connection between stochastic differential equations (SDEs) and quasi-linear PDEs has inspired new approaches of solving high dimensional parabolic PDEs which are ubiquitous in material sciences such as the Allen–Cahn equations for phase transitions, and quantum mechanics such as Schrodinger equations as well as option pricing and stochastic controls such as the Black–Scholes and Hamilton-Jacobi-Bellman equations. For PDEs in high dimensions, the main challenge of the traditional numerical methods, such as finite element, finite difference and spectral methods, is the curse of dimensionality, namely, the number of unknowns in the discretized systems for the PDEs grows exponentially in terms of the dimension of the problem. Recently, machine learning approaches using deep neural networks have taken advantage of the Pardoux–Peng's theory for forward and backward stochastic differential equations (FBSDEs) and PDEs. The solution to the PDEs can be learned by sampling the paths of involved stochastic processes, which

* Corresponding author.
E-mail address: cai@smu.edu (W. Cai).

https://doi.org/10.1016/j.jcp.2022.111557
0021-9991/

are discretized in time by the classic Euler–Maruyama scheme [7]. The first such an attempt was done in the work of [3], where neural network was used as an approximator to the gradient of the PDEs solutions, while the PDE's solution follows the dynamics of the FBSDEs, and the learning was carried out by imposing the terminal condition provided by the parabolic PDEs. Another approach [14] is to approximate the PDE's solution itself by a deep neural network, which also provides the gradient of the solution as required by the FBSDEs, the learning is then carried out by minimizing the difference between the solution given by the discretized SDEs and that given by the DNN at all discretization time stations. In this paper, an improved learning scheme will be proposed based on a similar approach in [14] but with more sound mathematical reasoning for the learning processes, to ensure the numerical methods' mathematical consistency and improved convergence for the PDEs' solutions.

The rest of the paper is organized as follows. In Section 2, we will review the Pardoux–Peng's theory, which establishes the connection between FBSDEs and quasilinear parabolic PDEs, with an emphasis on the relation between the classic Feynman–Kac formula and the nonlinear version given by the Pardoux–Peng theory. Section 3 will first review the algorithm proposed in [14], and then two new improved methods will be proposed. Section 4 will present numerical results of the new schemes for solving a 100-dimensional Black–Scholes–Barenblatt equation. Finally, a conclusion is given in Section 5.

2. Pardoux–Peng theory on FBSDEs and quasilinear parabolic PDEs

In this paper, we consider the scalar solution $u(t,x)$, $t\in[0,T]$, $x\in\mathbb{R}^d$ for the following d-dimensional parabolic PDE

$$\partial_t u+\frac{1}{2}\mathrm{Tr}[\sigma\sigma^T\nabla\nabla u]+\mu\cdot\nabla u=\phi \tag{1}$$

with a terminal condition

$$u(T,x)=g(x), \tag{2}$$

where $\sigma=\sigma(t,x,u)$, $\phi=\phi(t,x,u,\nabla u)$, $\mu=\mu(t,x,u,\nabla u)$ are functions with ranges in spaces with dimensions $d\times d$, 1 and d, respectively. We are interested in finding the initial value $u(0,x_0)$ given $x_0\in\mathbb{R}^d$. Therefore, in some sense our problem is similar to a time reverse problem for a time reversed version of (1) with an initial data at $t=0$.

Following Pardoux–Peng in [12], under certain regularity conditions, the forward-backward SDE reformulation gives a nonlinear implicit Feynman–Kac formula for the solution of the parabolic PDE (1). The FBSDEs are proposed as follows. Let $W_t=(W_t^1,\cdots,W_t^d)$ where each W_t^j is a standard Brownian motion, and $\{\mathcal{F}_t:0\le t\le T\}$ be its natural filtration on the time interval $[0,T]$. Then, we have the following stochastic equations for processes X_t, Y_t and Z_t in d, 1 and d dimensions that are adaptive to the filtration $\{\mathcal{F}_t:0\le t\le T\}$, respectively,

$$\begin{aligned} dX_t&=\mu(t,X_t,Y_t,Z_t)dt+\sigma(t,X_t,Y_t)dW_t,\\ X_0&=x_0, \end{aligned} \tag{3}$$

$$\begin{aligned} dY_t&=\phi(t,X_t,Y_t,Z_t)dt+Z_t^T\sigma(t,X_t,Y_t)dW_t,\\ Y_T&=g(X_T). \end{aligned} \tag{4}$$

If μ and σ do not explicitly depend on Y_t or Z_t, the FBSDEs are *decoupled*.

We can easily show that processes defined by

$$Y_t=u(t,X_t),\quad Z_t=\nabla u(t,X_t) \tag{5}$$

in fact satisfy the above equations (3) and (4). By using the Ito's formula [11] and the forward SDE of X_t, we have

$$\begin{aligned} dY_t=du(t,X_t)&=\partial_t u dt+\nabla u\cdot dX_t+\frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\partial_{ij}u d[X^i,X^j]_t\\ &=\partial_t u dt+\nabla u\cdot(\mu dt+\sigma\cdot dW_t)+\frac{1}{2}\mathrm{Tr}[\sigma\sigma^T\nabla\nabla u]dt\\ &=\left(\partial_t u+\nabla u\cdot\mu+\frac{1}{2}\mathrm{Tr}[\sigma\sigma^T\nabla\nabla u]\right)dt+Z_t^T\sigma dW_t, \end{aligned} \tag{6}$$

which gives the PDE (1) by comparing (6) with the backward SDE (4) for Y_t.

The determination of the third stochastic process Z_t from the two SDEs in (3) and (4) makes use of the martingale representation theory [8]. Consider the following special case of the backward SDE (4) as an example:

$$Y_t+\int_t^T f(s,X_s)ds+\int_t^T Z_s\cdot dW_s=g(X_T),\quad 0\le t\le T, \tag{7}$$

i.e. $\mu(t,x,u,\nabla u) = f(t,x)$, and $\sigma(t,x,u) = I_{d\times d}$ is the identity matrix. By taking the conditional expectation with respect to $\mathcal{F}_t$, we have

$$Y_t = \mathbb{E}\left[\, Y_t \,|\, \mathcal{F}_t\right] = \mathbb{E}\left[g(X_T) - \int_t^T f(s, X_s) ds \,\middle|\, \mathcal{F}_t \right], \quad 0 \le t \le T. \tag{8}$$

Next, we define the following martingale

$$L_t = \mathbb{E}\left[g(X_T) - \int_0^T f(s, X_s) ds \,\middle|\, \mathcal{F}_t \right], \quad 0 \le t \le T, \tag{9}$$

where $L_0 = Y_0$. By the martingale representation theorem [8], there exists a stochastic process $Z_t^\star$ such that

$$L_t = Y_0 + \int_0^t Z_s^\star \cdot dW_s, \quad 0 \le t \le T. \tag{10}$$

The stochastic process $Z_t^\star$ is unique in the sense that

$$\int_0^T \left\| Z_t^\star - Z_t^* \right\|^2 dt = 0, \quad \text{a.s.} \tag{11}$$

if Z_t^* satisfies the same condition (10) as $Z_t^\star$ [8]. Meanwhile, we can show that $Z_t = Z_t^\star$ solves the backward SDE (7),

$$\begin{aligned}
&Y_t + \int_t^T f(s, X_s) ds + \int_t^T Z_s^\star \cdot dW_s - g(X_T) \\
&= \mathbb{E}\left[g(X_T) - \int_t^T f(s, X_s) ds \,\middle|\, \mathcal{F}_t \right] + \left(\int_0^T - \int_0^t \right) f(s, X_s) ds + (L_T - L_t) - g(X_T) \\
&= \int_0^T f(s, X_s) ds + L_T - g(X_T) \\
&= L_T - \mathbb{E}\left[g(X_T) - \int_0^T f(s) ds \,\middle|\, \mathcal{F}_T \right] \\
&= 0.
\end{aligned}$$

Connection with the classic Feynman–Kac formula is interpreted as follows. If in the parabolic PDE (1), ϕ has a linear dependence on u, i.e.

$$\phi(t,x,u,\nabla u) = c(t,x)u(t,x) + f(t,x), \tag{12}$$

then, the backward SDE (4) has an explicit solution

$$Y_t = e^{-\int_t^T c(s,X_s)ds} g(X_T) - \int_t^T e^{-\int_t^s c(\tau,X_\tau)d\tau} f(s, X_s) ds - \int_t^T e^{-\int_t^s c(\tau,X_\tau)d\tau} Z_s^T \sigma(s, X_s, Y_s) dW_s. \tag{13}$$

By taking the conditional expectation on both sides, we arrive at

$$Y_t = \mathbb{E}\left[e^{-\int_t^T c(s,X_s)ds} g(X_T) - \int_t^T e^{-\int_t^s c(\tau,X_\tau)d\tau} f(s, X_s) ds \,\middle|\, \mathcal{F}_t \right]. \tag{14}$$

For $(t,x) \in [0,T] \times \mathbb{R}^d$, using $X_t = x$ as the initial condition of the forward SDE (3) on the time interval $[t,T]$ *instead of* $X_0 = x_0$, the traditional Feynman–Kac formula [11] is recovered,

$$u(t,x)=\mathbb{E}\left[e^{-\int_t^T c(s,X_s)ds}g(X_T)-\int_t^T e^{-\int_t^s c(\tau,X_\tau)d\tau}f(s,X_s)ds\,\middle|\,X_t=x\right]. \tag{15}$$

For a general parabolic equation with a nonlinear function $\phi(s,x,u,\nabla u)$, we have

$$Y_t=\mathbb{E}\left[g(X_T)-\int_t^T \phi(s,X_s,Y_s,Z_s)ds\,\middle|\,\mathcal{F}_t\right],$$

and for given $(t,x)\in[0,T]\times\mathbb{R}^d$, the following nonlinear equation for $u(t,x)$ is obtained

$$u(t,x)=\mathbb{E}\left[g(X_T)-\int_t^T \phi(s,X_s,u(s,X_s),\nabla u(s,X_s))ds\,\middle|\,X_t=x\right]. \tag{16}$$

3. FBSDE based deep neural network algorithms for quasilinear parabolic PDEs

The learning of the PDEs' solution will be based on the sample paths of the FBSDEs, which are linked to the PDE solution in (5). Paths of the FBSDEs will be produced by a time discretization algorithm with samples of the Brownian motion W_t.

Let $0=t_0<\cdots<t_N=T$ be a uniform partition of $[0,T]$. On each interval $[t_n,t_{n+1}]$, define time and Brownian motion increments as

$$\Delta t_n=t_{n+1}-t_n,\quad \Delta W_n=W_{t_{n+1}}-W_{t_n}. \tag{17}$$

Denoting X_{t_n}, Y_{t_n} and Z_{t_n} by X_n, Y_n and Z_n, respectively, and applying the Euler–Maruyama scheme to the FBSDEs (3) and (4), respectively, we have

$$X_{n+1}\approx X_n+\mu(t_n,X_n,Y_n,Z_n)\Delta t_n+\sigma(t_n,X_n,Y_n)\Delta W_n, \tag{18}$$

$$Y_{n+1}\approx Y_n+\phi(t_n,X_n,Y_n,Z_n)\Delta t_n+Z_n^T\sigma(t_n,X_n,Y_n)\Delta W_n. \tag{19}$$

Due to the relationship with the parabolic PDE, the solution to the parabolic PDE provides an alternative representation for Y_{n+1} and Z_{n+1},

$$Y_{n+1}=u(t_{n+1},X_{n+1}), \tag{20}$$

$$Z_{n+1}=\nabla u(t_{n+1},X_{n+1}). \tag{21}$$

In this paper, fully connected networks of L hidden layers will be used, which are given in the following form,

$$f_{\boldsymbol{\theta}}(\boldsymbol{x})=\boldsymbol{W}^{[L-1]}\sigma\circ(\cdots(\boldsymbol{W}^{[1]}\sigma\circ(\boldsymbol{W}^{[0]}(\boldsymbol{x})+\boldsymbol{b}^{[0]})+\boldsymbol{b}^{[1]})\cdots)+\boldsymbol{b}^{[L-1]}, \tag{22}$$

where $W^{[1]},\cdots,W^{[L-1]}$ and $b^{[1]},\cdots,b^{[L-1]}$ are the weight matrices and bias parameters, respectively, denoted collectively by θ, to be optimized via the training, $\sigma(x)$ is the activation function and $\circ$ is the application of the activation function σ applied to a vector quantity component-wisely.

3.1. Existing FBSDE based neural network algorithms

3.1.1. Deep BSDE [3]

The deep BSDE method trains a network to approximate the random value Y_N at time $t=T$, where $X_0=x_0$ is the input. Y_0, Z_0 are trainable variables and Y_0 is the targeted quantity of the algorithm. $W_n, X_n, 0\le n\le N$ can be obtained similarly as before. The algorithm can be organized as follows.

(1) The initial value $X_0=x_0$ is given. Trainable variables Y_0 and Z_0 are randomly initialized.
(2) On each time interval $[t_n,t_{n+1}]$, use the Euler–Maruyama scheme to calculate X_{n+1} and Y_{n+1} as in (18) and (19). Then, train a fully connected feedforward network

$$f_\theta^{(n+1)}(\cdot)\approx\nabla u(t_{n+1},\cdot) \tag{23}$$

where $f_\theta^{(n+1)}(\cdot)$ is a fully connected neural network of H hidden layers of the form given in (22). Activation functions including ReLU, Tanh, Sigmoid, etc. can be used.
(3) Connect all quantities (subnetworks $f_\theta^{(n)}(\cdot)$, etc) at $\{t_n\}$ to form a network that outputs Y_N, which is expected to be an approximation of $u(t_N,X_N)$.

(4) The loss function is then defined by a Monte Carlo approximation of

$$\mathbb{E}\left\|Y_N - g(X_N)\right\|^2. \tag{24}$$

The deep BSDE method has been shown to give convergent numerical results for various high dimensional parabolic equations [3] and a posteriori estimate suggests strong convergence of half order [4].

Remark 1. The deep BSDE method from [3] trains the network for the specific initial data $X_0 = x_0$ and yields only an approximation to the PDE solution $Y_0 = u(0, x_0)$. Therefore, once the desired initial data is changed, a new training may have to be carried out. Also, the total size of N individual sub-networks used to approximate $Z_n = \nabla u(t_n, X_n)$, $n = 1, \cdots, N-1$ will grow linearly in terms of time discretization steps N, resulting in large amount of training parameters if higher accuracy of the PDE solution is desired.

3.1.2. FBSNNs [14] (Scheme 1)

The FBSNNs train a network $u_\theta(t, x)$ that directly approximates the solution to the PDE (1) in some region in the (t, x) space. The network has a fixed size of number of hidden layers and neurons per layer. The algorithm can be organized as follows.

(1) The initial value $X_0 = x_0$ is given. Evaluate Y_0 and Z_0 using the network

$$Y_0 = u_\theta(t_0, X_0), \quad Z_0 = \nabla u_\theta(t_0, X_0). \tag{25}$$

The gradient above is calculated by an automatic differentiation.

(2) On each time interval $[t_n, t_{n+1}]$, use the Euler–Maruyama scheme (18) to calculate X_{n+1}, and use the network for Y_{n+1} and Z_{n+1}, i.e.

$$\begin{aligned} X_{n+1} &= X_n + \mu(t_n, X_n, Y_n, Z_n)\Delta t_n + \sigma(t_n, X_n, Y_n)\Delta W_n, \\ Y_{n+1} &= u_\theta(t_{n+1}, X_{n+1}), \\ Z_{n+1} &= \nabla u_\theta(t_{n+1}, X_{n+1}). \end{aligned} \tag{26}$$

On the other hand, calculate a reference value $Y^\star_{n+1}$ using the Euler–Maruyama scheme (19)

$$Y^\star_{n+1} = Y_n + \phi(t_n, X_n, Y_n, Z_n)\Delta t_n + Z_n^T \sigma(t_n, X_n, Y_n)\Delta W_n. \tag{27}$$

(3) The loss function is taken as a Monte Carlo approximation of

$$\mathbb{E}\left[\sum_{n=1}^{N}\left\|Y_n - Y^\star_n\right\|^2 + \left\|Y_N - g(X_N)\right\|^2 + \left\|Z_N - \nabla g(X_N)\right\|^2\right]. \tag{28}$$

In this paper, we will name the above numerical method Scheme 1. In order to compare the training results using different values of N, the loss function for Scheme 1 is modified as

$$L_1[u_\theta; x_0] = \frac{1}{M}\left[\sum_{\omega}\frac{1}{N}\sum_{n=1}^{N}\left\|Y_n - Y^\star_n\right\|^2 + \beta_1\left\|Y_N - g(X_N)\right\|^2 + \beta_2\left\|Z_N - \nabla g(X_N)\right\|^2\right] \tag{29}$$

where M serves as the batch size of the training and ω denotes any instance of sampling of the discretized Brownian motion $W_n, 0 \le n \le N-1$, and β_1, β_2 are the penalty parameters for the terminal conditions. The averaging factor $1/N$ is introduced for consistency consideration as the reduction of the loss function as N increases, when applied to the exact solution, is expected.

Remark 2. The FBSNNs algorithm proposed in [14] relies on a loss function involving the difference between sequences $\{Y_n\}$ and $\{Y^*_n\}$, which carry the information inside the time interval $(0, T)$. While the discrete stochastic process $\{Y_n\}$ can be expected to approach a continuous stochastic process as defined in the backward SDE (4), the question whether the discrete sequence of random variables $\{Y^*_n\}$ will converge to the same stochastic process is not clear. As a result, the rate and extent for the difference between $\{Y_n\}$ and $\{Y^*_n\}$, thus the loss function, approaching to zero is not certain. Our numerical test will provide some evidence for this concern.

3.2. FBSDE based deep neural network algorithms for PDEs

In this section, we propose improved algorithms for the FBSDEs based deep neural networks similar to Scheme 1 above first proposed in [14], but are mathematically consistent in the definition of the loss function and the discretization of both forward and backward SDEs related to the PDE solutions. Specifically, the loss will be made of the difference of *two discrete stochastic processes*, which will approach the same process given by the backward SDEs if the overall scheme converges.

3.2.1. FBSDE based algorithms - Scheme 2

Based on the Remark 2 from Section 3.1.2, we would like to design a new scheme whose loss function is expected to show the strong convergence rate of the Euler–Maruyama scheme for the discretization of the FBSDEs. A key factor will be to make the loss function as the pathwise differences between two stochastic processes, which will converge to the same continuous adapted diffusion process if the time discretization of FBSDEs and DNN approximations converge.

Scheme 2. Train a DNN $u_\theta(t,x)$ to approximate the solution $u(t,x)$ of the parabolic PDE (1).

(1) Given $X_0 = x_0$ and let $Y_0 = u_\theta(t_0, X_0)$, $Z_0 = \nabla u_\theta(t_0, X_0)$.

(2) On each time interval $[t_n, t_{n+1}]$, calculate X_{n+1} and Y_{n+1} using the Euler–Maruyama scheme (18) and (19), respectively, and calculate Z_{n+1} using the network, i.e.

$$\begin{aligned} X_{n+1} &= X_n + \mu(t_n, X_n, Y_n, Z_n)\Delta t_n + \sigma(t_n, X_n, Y_n)\Delta W_n, \\ Y_{n+1} &= Y_n + \phi(t_n, X_n, Y_n, Z_n)\Delta t_n + Z_n^T \sigma(t_n, X_n, Y_n)\Delta W_n, \\ Z_{n+1} &= \nabla u_\theta(t_{n+1}, X_{n+1}). \end{aligned} \tag{30}$$

Next, define a second variable by the DNN representation of the PDE solution,

$$Y^\star_{n+1} = u_\theta(t_{n+1}, X_{n+1}). \tag{31}$$

(3) For a batch size M with ω denoting any of the M sample paths, the loss function is given as

$$L_2[u_\theta; x_0] = \frac{1}{M}\sum_{\omega}\left[\frac{1}{N}\sum_{n=1}^{N}\left\|Y_n - Y^\star_n\right\|^2 + \beta_1\left\|Y^\star_N - g(X_N)\right\|^2 + \beta_2\left\|Z_N - \nabla g(X_N)\right\|^2\right], \tag{32}$$

where β_1, β_2 are the penalty parameters of the terminal condition.

The quantity $Y^\star_N$ is used in the terminal term in the loss function $L_2[u_\theta; x_0]$, because here it is a straightforward output of the neural network u_θ.

3.2.2. FBSDE based algorithms - Scheme 3

In the Scheme 2 above, the discrete process (31) is defined through the composition function using the DNN representation of the PDE solution $u_\theta(t,x)$. An alternative way is given below where both discrete processes are obtained from an Euler–Maruyama discretization of the SDEs.

Scheme 3. Train a DNN $u_\theta(t,x)$ to approximate the solution $u(t,x)$ of the parabolic PDE (1).

(1) Given the initial values $X_0^{(1)} = X_0^{(2)} = x_0$ and we compute

$$Y_0^{(1)} = Y_0^{(2)} = u_\theta(t_0, x_0), \quad Z_0^{(1)} = Z_0^{(2)} = \nabla u_\theta(t_0, x_0) \tag{33}$$

from the network $u_\theta(t,x)$.

(2) On each time interval $[t_n, t_{n+1}]$, calculate $X_{n+1}^{(1)}, Y_{n+1}^{(1)}$ and $Z_{n+1}^{(1)}$ as in (26) of Scheme 1, then $X_{n+1}^{(2)}, Y_{n+1}^{(2)}$ and $Z_{n+1}^{(2)}$ as in (30) of Scheme 2, i.e.

$$\begin{aligned} X_{n+1}^{(1)} &= X_n^{(1)} + \mu(t_n, X_n^{(1)}, Y_n^{(1)}, Z_n^{(1)})\Delta t_n + \sigma(t_n, X_n^{(1)}, Y_n^{(1)})\Delta W_n, \\ Y_{n+1}^{(1)} &= u_\theta(t_{n+1}, X_{n+1}^{(1)}), \\ Z_{n+1}^{(1)} &= \nabla u_\theta(t_{n+1}, X_{n+1}^{(1)}), \end{aligned} \tag{34}$$

$$\begin{aligned} X_{n+1}^{(2)} &= X_n^{(2)} + \mu(t_n, X_n^{(2)}, Y_n^{(2)}, Z_n^{(2)})\Delta t_n + \sigma(t_n, X_n^{(1)}, Y_n^{(1)})\Delta W_n, \\ Y_{n+1}^{(2)} &= Y_n^{(2)} + \phi(t_n, X_n^{(2)}, Y_n^{(2)}, Z_n^{(2)})\Delta t_n + (Z_n^{(2)})^T \sigma(t_n, X_n^{(2)}, Y_n^{(2)})\Delta W_n, \\ Z_{n+1}^{(2)} &= \nabla u_\theta(t_{n+1}, X_{n+1}^{(2)}). \end{aligned} \tag{35}$$

(3) For a batch size M with ω denoting any of the M sample paths, the loss function is defined by

$$L_3[u_\theta; x_0] = \frac{1}{M}\sum_{\omega}\left[\frac{1}{N}\sum_{n=1}^{N}\left\|Y_n^{(1)} - Y_n^{(2)}\right\|^2 + \beta_1\left\|Y_N^{(1)} - g(X_N^{(1)})\right\|^2 + \beta_2\left\|Z_N^{(1)} - \nabla g(X_N^{(1)})\right\|^2\right], \tag{36}$$

where β_1, β_2 are the penalty parameters of the terminal condition.

Remark 3. Formulation of the deep BSDE method for solving PDEs in [3] is closely related to deep learning approximation of stochastic control problems using DNN [2] [5], and Z_n, approximated by individual subnetworks $f_\theta^{(n)}(\cdot)$ in [3], serves as the policy at time t_n in the context of reinforcement learning [15][9]. On the other hand, the policy Z_n for the Scheme 1, 2, 3 is given analytically by the gradient of the DNN used to represent the PDE solution itself. Moreover, there is some difference in the loss functions used in the stochastic gradient optimizations for finding the optimal policy function parameterized by the deep neural networks. In the deep BSDE method, the mis-match of the terminal condition predicted by the BSDE is used as the loss function while Schemes 1, 2 and 3 use, in addition, the mis-match of two stochastic processes, one generated by the BSDEs and one by a function composition of stochastic processes.

4. Numerical results

In this section, we will carry out several tests on Scheme 1 from [14] and the new Scheme 2 and Scheme 3, for a 100-dimensional Black–Scholes–Barenblatt equation and its variants.

4.1. 100-dimensional Black–Scholes–Barenblatt equation

Consider the following 100-dimensional Black–Scholes–Barenblatt (BSB) equation from [14] as the model problem: for $t \in [0, T]$ and $x \in \mathbb{R}^d$, the scalar function $u(t, x)$ satisfies

$$\begin{aligned} u_t + \frac{1}{2}\mathrm{Tr}\left[\sigma^2 \mathrm{diag}(xx^T)\nabla\nabla u\right] &= r(u - \nabla u \cdot x), \\ u(T, x) &= \|x\|^2. \end{aligned} \tag{37}$$

The PDE is linked to the FBSDEs

$$\begin{aligned} dX_t &= \sigma\, \mathrm{diag}(X_t) dW_t, \\ X_0 &= x_0, \\ dY_t &= r(Y_t - Z_t \cdot X_t)dt + \sigma\, Z_t^T \mathrm{diag}(X_t) dW_t, \\ Y_T &= g(X_T), \end{aligned} \tag{38}$$

where $g(x) = \|x\|^2$, and $x_0 \in \mathbb{R}^d$ is the position where we like to get the initial value $u(0, x_0)$. The exact solution to the PDE (37) is given in a closed form by

$$u(t, x) = e^{(r+\sigma^2)(T-t)} \|x\|^2, \tag{39}$$

so that we can test the accuracy of the DNN schemes. Parameters are given by $d = 100$, $T = 1.0$, $\sigma = 0.4$, $r = 0.05$ and

$$x_0 = (1, 0.5, 1, 0.5, \cdots, 1, 0.5). \tag{40}$$

We use a 6-layer fully connected feedforward neural network for $u_\theta(t, x)$ with 5 hidden layers, each having 256 neurons. The activation function is the sine function as suggested by [14]. We train the network with the Adam optimizer with descending learning rates 1e-3, 1e-4, 1e-5, 1e-6 and 1e-7, each for 10000 steps. The batch size is $M = 100$.

In the loss functions (29), (32) and (36), the penalty parameters are chosen as $\beta_1 = \beta_2 = 0.02$.

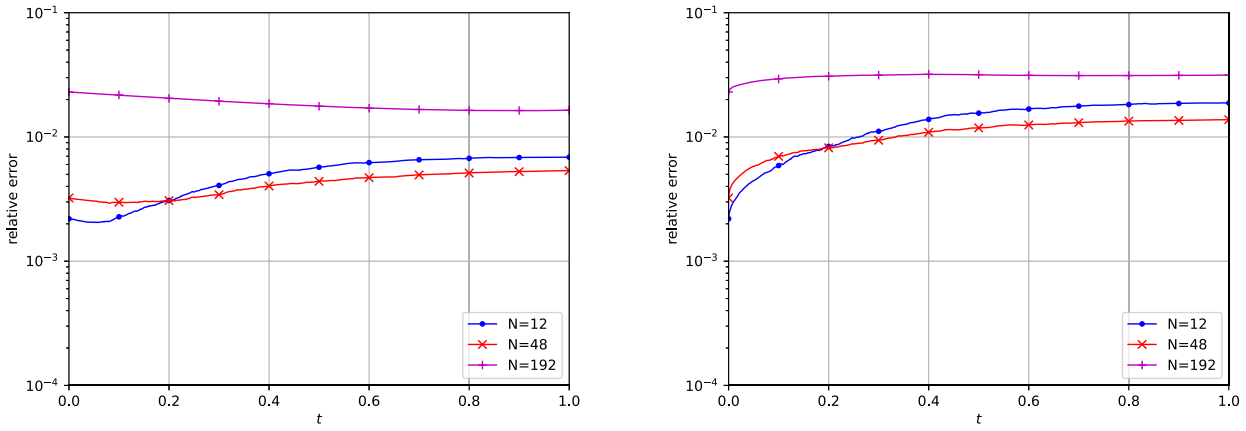
Illustration of the training results in the high-dimensional space is provided along the sample paths. When the training is finished, we randomly generate 1000 sample paths for verification of the accuracy, with a finer time discretization with time steps $\Delta t_n = 1/1000$. For each (discretized) sample path ω and for $0 \le n \le 1000$, the relative error at $(t_n, X_n(\omega))$ (or at $(t_n, X_n^{(2)}(\omega))$ when using Scheme 3) is defined by

$$e_n^N(\omega) = \frac{|u_\theta(t_n, X_n(\omega)) - u(t_n, X_n(\omega))|}{|u(t_n, X_n(\omega))|}. \tag{41}$$

The mean and the standard deviation (SD) of each e_n^N can also be calculated, denoted by Mean(e_n^N) and SD(e_n^N), respectively.

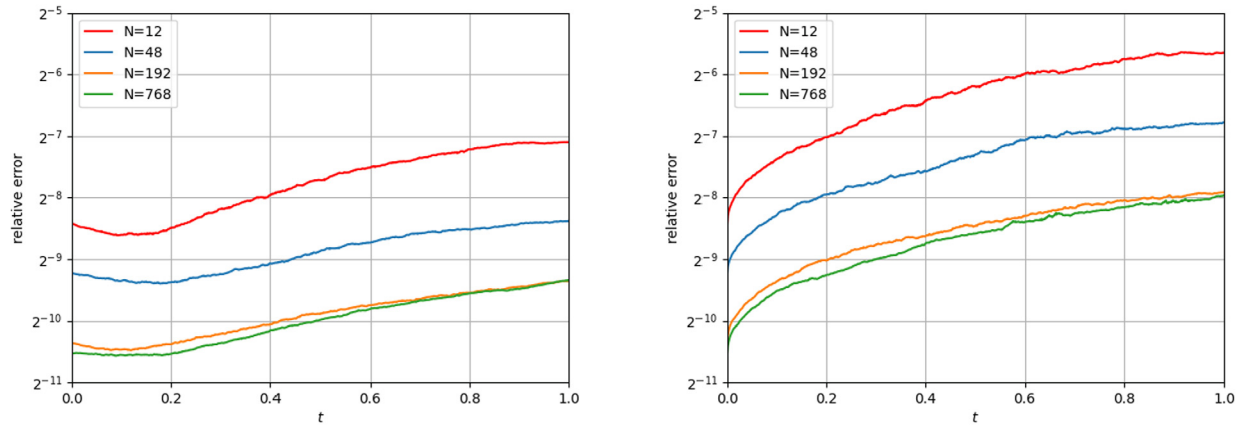
4.1.1. Scheme 1 from [14]

Fig. 1 shows the relative error of Scheme 1 for $N = 12$, 48 and 192, where the mean error and the mean error plus two standard deviations of the error are presented. We can see the reduction of the errors from $N = 12$ to $N = 48$, however, the error increases from $N = 48$ to $N = 192$. This degeneracy in accuracy is an indication that as the time discretization is refined, the two quantities in the definition of loss function (28) do not approach the same continuous stochastic process. In fact, as it is defined by (27), $\{Y_n^\star\}$ may not converge to a continuous stochastic process at all.



(a) Error Mean (b) Error Mean plus two SDs

Fig. 1. (Non-convergence) Relative error of Scheme 1 for $N = 12$ (middle), 48 (bottom) and 192 (top).



(a) Error Mean (b) Erro Mean plus two SDs

Fig. 2. Relative error of Scheme 2 for $N = 12, 48, 192$ and 768.

4.1.2. Scheme 2 and Scheme 3

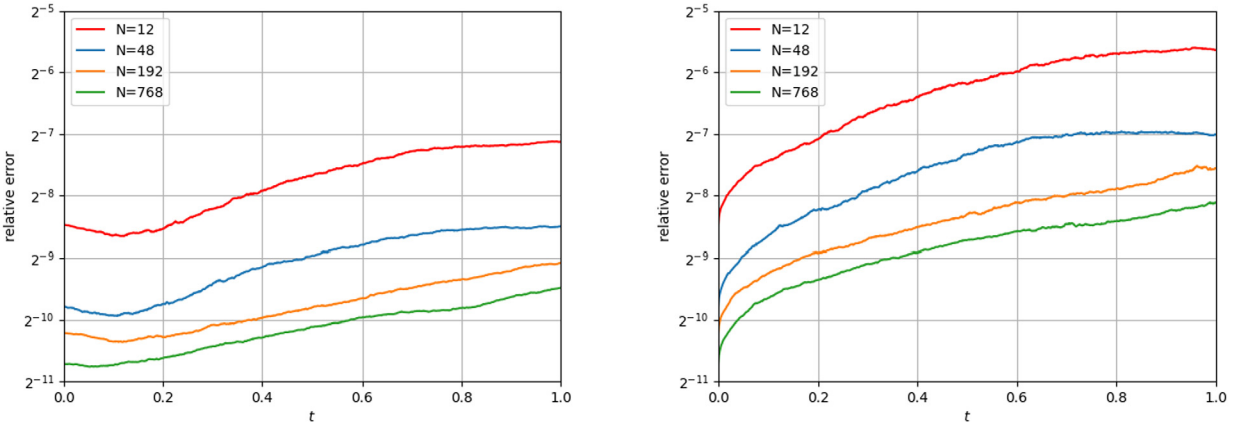
Fig. 2 and Fig. 3 show the mean error and mean error plus two standard derivations of the error for Scheme 2 and Scheme 3 for $N = 12$, $N = 48$, $N = 192$ and $N = 768$, respectively. Errors in these figures are measured in a log base 2 scale to reflect the expected convergence rate as N increases, to be elaborated below.

Both the results in Fig. 2 and Fig. 3 show the convergence of the new Scheme 2 and Scheme 3, respectively, in contrast to the degeneracy of the accuracy of Scheme 1 when the time discretization is refined. For both new schemes, we can see improvement of the accuracy from $N = 48$ to $N = 192$ is close to the one from $N = 12$ to $N = 48$, but the improvement of $N = 768$ over $N = 192$ is less. This indicates the network training might dominate the error compared to the time discretization error. In fact, the terminal parts of the loss function failed to halve in the $N = 768$ cases compared to $N = 192$. The convergence rates as N increases are expected to follow the order of the Euler–Maruyama scheme. To show the convergence rates, we define

$$\alpha_N^{(1)} = \log_4 \max_{0\le n\le 1000} \frac{\text{Mean}(e_n^{4N})}{\text{Mean}(e_n^N)}, \quad \alpha_N^{(2)} = \log_4 \max_{0\le n\le 1000} \frac{\text{Mean}(e_n^{4N}) + 2\text{SD}(e_n^{4N})}{\text{Mean}(e_n^N) + 2\text{SD}(e_n^N)}, \tag{42}$$

which reflect the worst scenery of convergence rates for $t \in [0, 1]$. One can see from Table 1 that Scheme 2 approximately follows the half order of convergence from $N = 12$ to $N = 192$, while the improvement from $N = 192$ to $N = 768$ for Scheme 2 degenerates due to the presence of other errors from the DNN and similar situation happens for Scheme 3.

Fig. 4 (a) (b) show the prediction of trained networks using Scheme 2 and Scheme 3 with $N = 192$ along 8 sampled test paths depicted in Fig. 4 (c), in comparison with the exact solution, where the average error of the prediction is given in Fig. 4 (d).



(a) Error Mean (b) Error Mean plus two SDs

Fig. 3. Relative error of Scheme 3 for $N = 12, 48, 192$ and 768.

Table 1
Convergence rate as N increases.

	Scheme 2		Scheme 3	
N-$4N$	$\alpha_N^{(1)}$	$\alpha_N^{(2)}$	$\alpha_N^{(1)}$	$\alpha_N^{(2)}$
12-48	0.363	0.401	0.585	0.551
48-192	0.401	0.496	0.181	0.212
192-768	-6.93e-3	1.900e-2	0.146	0.179

4.1.3. Extrapolation for higher order accuracy in solution Y_0

In Section 4.1.2 we have seen that Scheme 2 and Scheme 3 have the convergence behavior as the Euler–Maruyama scheme, so we can assume that the truncation error may have the following asymptotic ansatz

$$u_\theta^N - u = C_1 N^{-\frac{1}{2}} + C_2 N^{-1} + O(N^{-\frac{3}{2}}), \tag{43}$$

where the leading term $C_1 N^{-\frac{1}{2}}$ dominates the error when N is sufficiently large. If this holds for both u_θ^N and u_θ^{4N} for some constants C_1 and C_2, then we can define an extrapolated solution

$$\begin{aligned} u_{\text{ex}}^{4N} &= 2u_\theta^{4N} - u_\theta^N \\ &= u - \frac{C_2}{2} N^{-1} + O(N^{-\frac{3}{2}}) \end{aligned} \tag{44}$$

as an improved approximation to the solution.

For the model problem (37), the extrapolation is valid for the approximation of $Y_0 = u(0, x_0)$ and $u(0, x)$ in a neighborhood near x_0, as shown by Table 2 and Fig. 5. In terms of the accuracy of Y_0, by training the DNNs only with $N = 12$ and $N = 48$, the extrapolated result $u_{\text{ex}}^{48}(0, x_0)$ has its accuracy outperforming those using $N = 768$ which takes more than 10 times longer time to train, when using either Scheme 2 or Scheme 3. Due to training difficulties, the improvement for using extrapolation on $N = 768$ is marginal, but still exists.

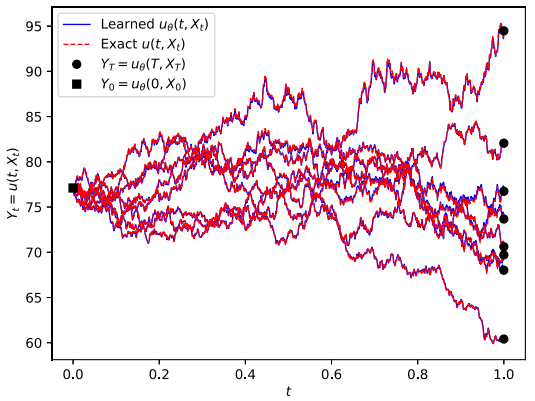
Note that the extrapolation approach usually may not work for the whole time interval along the entire sample paths. For instance, the values at $t = T$ are subject to explicit fitting of the terminal condition from the loss functions (32) and (36), so we cannot expect a general constant C_1 in (43) for $u_\theta^N(T, x)$ and $u_\theta^{4N}(T, x)$. The result in Fig. 5 shows that the extrapolation technique can be used for a time interval $0 \le t \le 0.1$. For the value of Y_0, the extrapolation with Scheme 2 has the order of convergence approximately at 0.744 between $N = 48$ and $N = 192$.

4.1.4. Region of validity of DNN $u_\theta(t, x)$ near X_0

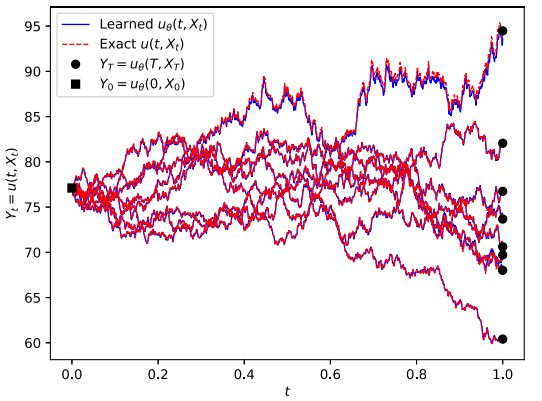
In this section, we will verify the validity of the networks $u_\theta(t, x)$ in a region that are larger than the one sampled during the training process. For this purpose, we randomly sample the initial value $X_0 = \tilde{x}_0$ from a *cubic* neighborhood of x_0 with halved edge length R, i.e.,

$$(\tilde{x}_0)_j = (x_0)_j \cdot (1 + \varepsilon_j), \quad 1 \le j \le d = 100, \tag{45}$$

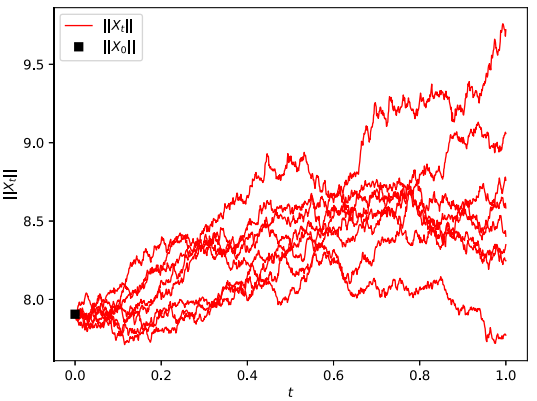
where ε_j are i.i.d. random variables with uniform distribution on $(-R, R)$. For the network trained with Scheme 2 and $N = 192$, we compare the resulting error using the same measurement with $R = 0.25$ and $R = 0.5$, while keeping one



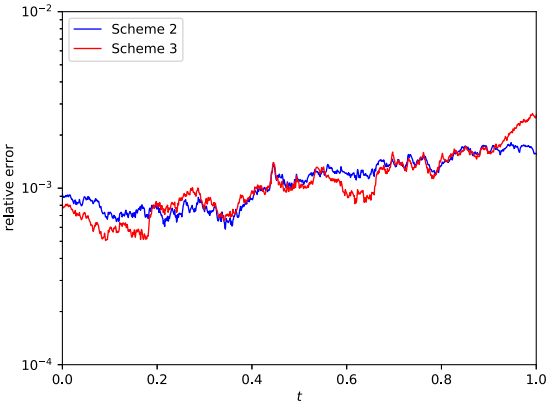
(a) Prediction by Scheme 2



(b) Prediction by Scheme 3



(c) $\|X_t\|$ along 8 sample paths



(d) Averaged relative error on $[0, T]$

Fig. 4. Prediction of 8 test sample paths from training results of Scheme 2 and Scheme 3, $N = 192$.

Table 2
Relative error of Y_0 from the network approximation and extrapolation.

	Scheme 2		Scheme 3	
N - $4N$	u_θ^N	u_{ex}^N	u_θ^N	u_{ex}^N
12	2.91e-03		2.82e-03	
12-48	1.67e-03	**4.29e-04**	1.13e-03	5.57e-04
48-192	7.58e-04	**1.53e-04**	8.43e-04	5.55e-04
192-768	6.77e-04	5.97e-04	5.96e-04	3.49e-04

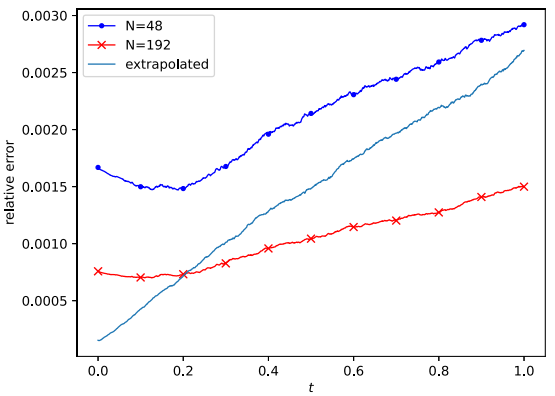
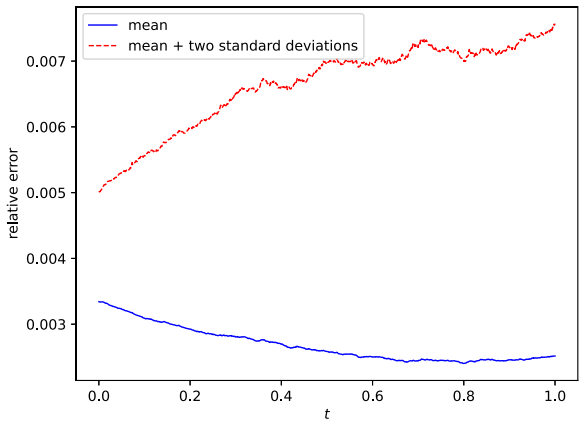
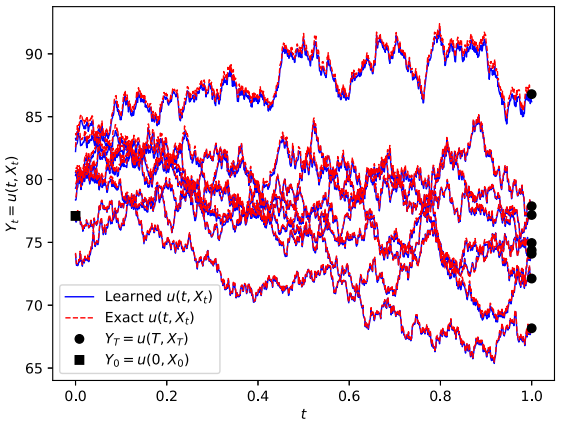


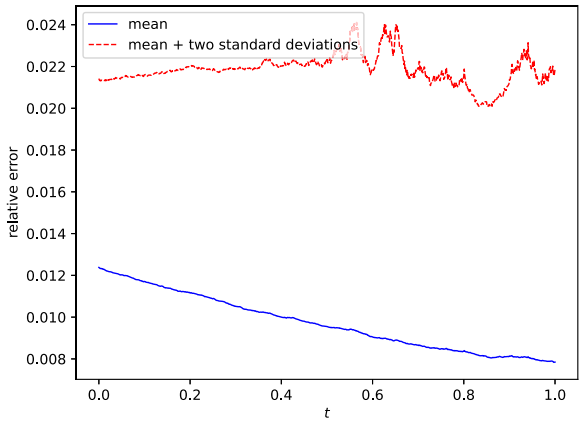
Fig. 5. Mean relative error of the extrapolation $u_\theta^{192}(t, X_t)$ for $0 \le t \le T$, using Scheme 2.



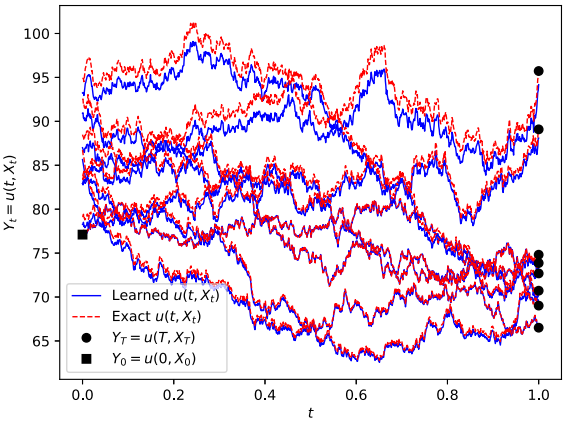
(a) Relative error, $R = 0.25$



(b) Prediction of 8 sample paths



(c) Relative error, $R = 0.5$



(d) Prediction of 8 sample paths

Fig. 6. Training error verified with initial value $\tilde{x}_0$ from a neighborhood of x_0, using Scheme 2, $N = 192$.

sample starting exactly from x_0 (for the sake of plotting), see Fig. 6. The averaged relative error is slightly larger at $t = 0$ because during the training process these regions are less likely to be visited since we fixed the initial value for all training paths at $X_0 = x_0$. If we look at the overall maximum for $t \in [0, T]$, we can still have an averaged relative error of 0.34% for $R = 0.25$ and 1.25% for $R = 0.5$. Also, it is noted that, in comparison with the non-perturbed result, the trained network fits the solution of the PDE better when Y_t has a value below 80.

This result shows that the DNN we trained for $x = x_0$ is in fact can be used in a local neighborhood around x_0 for the whole time interval $0 \le t \le T$.

4.2. MultiscaleDNN for the BSB equation with temporal oscillations

In a recent work [10], a multi-scale DNN was proposed, which consists of a series of parallel normal sub-networks, each of which receiving a scaled version of the input, and outputs of the sub-networks are combined to form the final output of the MscaleDNN (see Fig. 7). The individual sub-networks in the MscaleDNN with a scaled input is designed to approximate a segment of frequency content of the targeted function, and the effect of the scaling is to convert a specific high frequency content to a lower frequency range so that the learning can be accomplished more quickly, which is shown by the recent work [10] on the frequency dependence of the DNN convergence.

Fig. 7 shows the schematics of a MscaleDNN consisting of n sub-networks. Each scaled input passing through a fully-connected sub-network, which can be expressed in the formula (22), here we use the *sine* function for the activation function, i.e.,

$$\sigma(x) = \sin(x). \tag{46}$$

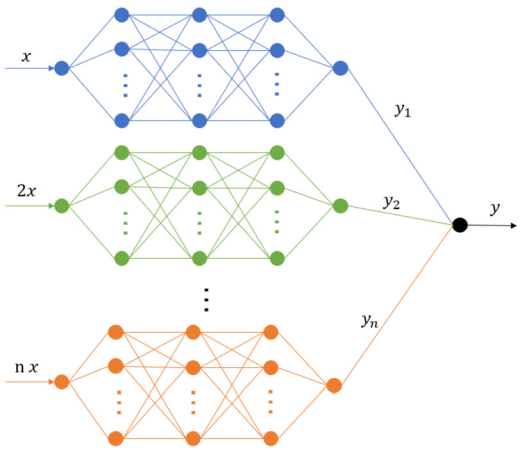


Fig. 7. Illustration of a MscaleDNN.

Mathematically, the final output of a MscaleDNN solution is represented by the following sum of sub-networks $f_{\theta^{n_i}}$ with network parameters denoted by θ^{n_i} (i.e. weight matrices and bias)

$$f(\boldsymbol{x}) \sim \sum_{i=1}^{M} \boldsymbol{W}_i^{[L]} f_{\theta^{n_i}}(\boldsymbol{\alpha}_i \cdot \boldsymbol{x}) + \boldsymbol{b}^{[L]}, \tag{47}$$

where $\boldsymbol{\alpha}_i$ is the chosen scale vector for the i-th sub-network in Fig. 7. For more details on the design of the MscaleDNN, refer to [10].

For the input scales, the general idea is to adopt various scaling factors for different components of the input, depending on the complexity of the solution of the PDE to be solved.

The MscaleDNN is tested with the following model problem, modified from the BSB equation above with an oscillatory factor to effectively increase the learning difficulty:

$$\begin{aligned} \partial_t u + \frac{1}{2}\mathrm{Tr}[\sigma^2 \nabla\nabla u] &= \phi, \\ u(T,x) &= g(x), \end{aligned} \tag{48}$$

where the dimension $d = 100$, $T = 1.0$, $\sigma = 0.4$ and $r = 0.05$ are unchanged parameters compared to (37),

$$g(x) = \|x\|^2 \left(1 + \alpha \sin(\beta S_1 - \gamma T)\right), \tag{49}$$

$$\phi(t,x,u,\nabla u) = r(u - \nabla u \cdot x) + \alpha e^{(r+\sigma^2)(T-t)} P(t,x), \tag{50}$$

$$P(t,x) = (r\beta S_1 S_2 - \gamma S_2 + 2\sigma^2 \beta S_3)\cos(\beta S_1 - \gamma t) - \frac{\sigma^2 \beta^2}{2} S_2^2 \sin(\beta S_1 - \gamma t), \tag{51}$$

where each $S_j = \sum_{i=1}^{d} x_i^j$, and α, β and γ are parameters to be tuned. The modified PDE (48) has a solution

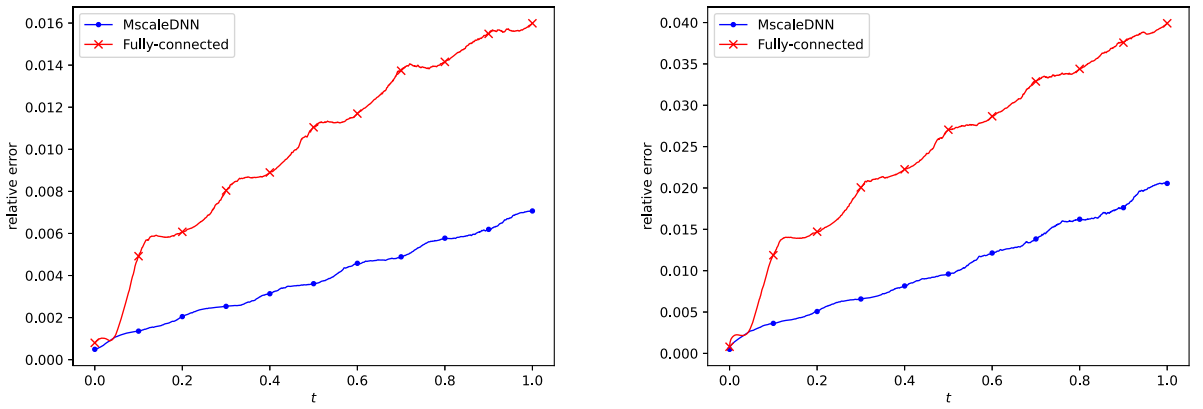
$$u(t,x) = e^{(r+\sigma^2)(T-t)} \|x\|^2 \left(1 + \alpha \sin(\beta S_1 - \gamma t)\right), \tag{52}$$

and corresponds to the FBSDEs

$$\begin{aligned} dX_t &= \sigma\, \mathrm{diag}(X_t) dW_t, \\ X_0 &= x_0, \\ dY_t &= \left(r(Y_t - Z_t \cdot X_t) + \alpha e^{(r+\sigma^2)(T-t)} P(t, X_t)\right) dt + \sigma Z_t^T \mathrm{diag}(X_t) dW_t, \\ Y_T &= g(X_T). \end{aligned} \tag{53}$$

We apply $\alpha = 0.025$, $\beta = 0.25$ and $\gamma = 32$ to the above equation. During the training process, we use the same settings for the fully-connected DNN as in previous tests. For the MscaleDNN, the network is divided into 4 sub-networks, each having 5 hidden layers with 64 neurons per layer, so that sizes of the networks in the comparison are matching. The scaled inputs for the sub-networks are given by

$$(3^0 t, x), \quad (3^1 t, x), \quad (3^2 t, x), \quad (3^3 t, x), \tag{54}$$



(a) Mean (b) Mean plus two SDs

Fig. 8. Relative training error for fully-connected DNN and MscaleDNN for the model problem with oscillation, using Scheme 2 and $N = 48$.

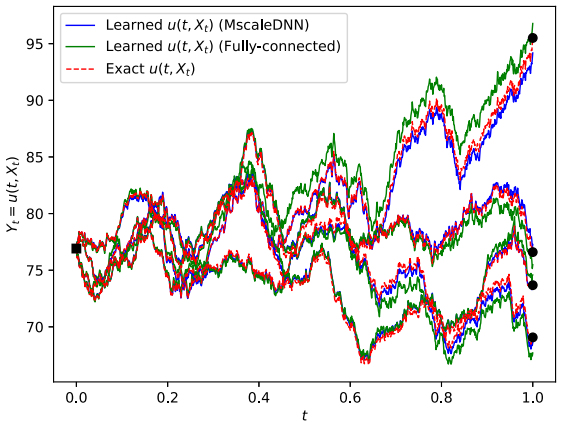
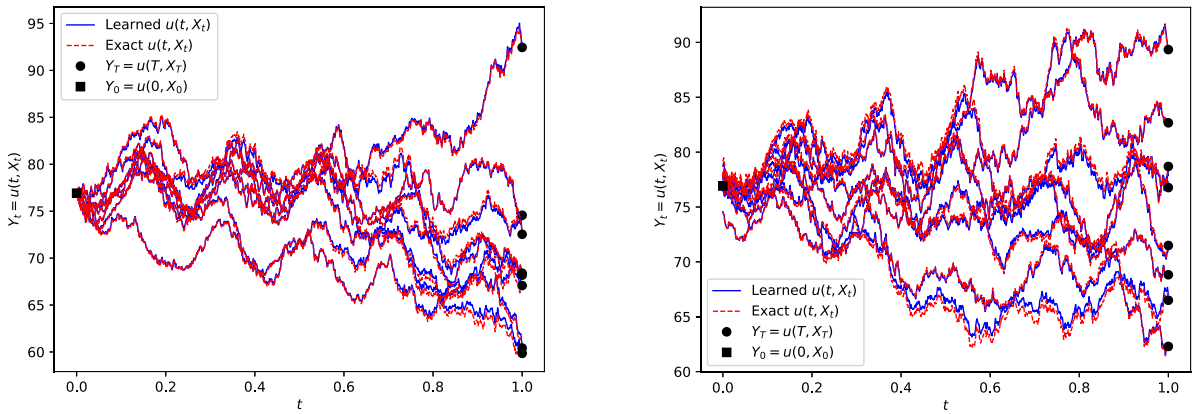


Fig. 9. Comparison between MscaleDNN and fully-connected DNN for the prediction of 4 sample paths for the model problem with oscillation, using Scheme 2 and $N = 48$.



(a) Starting from $(0, x_0)$. (b) Starting from a cubic neighborhood of x_0, $R = 0.125$.

Fig. 10. Prediction of 8 sample paths for problem with oscillation (48), using the MscaleDNN with Scheme 2 and $N = 192$.

so that a wider range of frequency of t can be captured with the MscaleDNN. When applying Scheme 2 and $N = 48$, from Fig. 8 and Fig. 9 we can see the MscaleDNN halves the overall error compared to the fully-connected network. One can also predict sample paths with better accuracy using the MscaleDNN, too, see Fig. 10 in the neighborhood of x_0.

5. Conclusion

In this paper, we have proposed two FBSDE based DNN algorithms for high dimensional quasilinear parabolic equations. The key component of the proposed algorithms is the loss function used, consisting of, in addition to the terminal condition of the PDE, the path-wise difference of two convergent stochastic processes from either discretized SDEs or the PDEs network solution. As the two stochastic processes converge to the same stochastic processes in the Pardoux–Peng theory, the new algorithms are able to demonstrate a nearly half-order strong convergence rate of the underlying Euler–Maruyama scheme for the SDEs. We also show that the extrapolation method verifies the convergence order of the DNN solutions to some extent and further enhances the resulting accuracy of the estimate on the initial value of the PDE. For PDEs with time oscillatory solutions, we demonstrated that the MscaleDNN is shown to provide an enhancement of the resulting accuracy.

Future research will be done to improve the convergence of the networks and the overall schemes, including MscaleDNN for PDEs with spatially oscillatory solutions.

CRediT authorship contribution statement

Wenzhong Zhang: Conceptualization, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Wei Cai:** Conceptualization, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank Jiequn Han, Weinan E, and Maziar Raissi for helpful discussions on the topic of this work. The work of W. Cai is supported by the US National Science Foundation grant DMS-2207449.

References

[1] J.L. Doob, Classical Potential Theory and Its Probabilistic Counterpart: Advanced Problems, Springer Science & Business Media, Dec 2012.
[2] J. Han, E. Weinan, Deep learning approximation for stochastic control problems, in: Deep Reinforcement Learning Workshop, NIPS, 2016.
[3] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. USA 115 (34) (Aug 2018) 8505–8510.
[4] J. Han, J. Long, Convergence of the deep BSDE method for coupled FBSDEs, Probab. Uncertain. Quant. Risk 5 (1) (Dec 2020) 1–33.
[5] S. Ji, S. Peng, Y. Peng, X. Zhang, Three algorithms for solving high-dimensional fully coupled FBSDEs through deep learning, IEEE Intell. Syst. 35 (3) (Feb 2020) 71–84.
[6] N. El Karoui, S. Peng, M.C. Quenez, Backward stochastic differential equations in finance, Math. Finance 7 (1) (Jan 1997) 1–71.
[7] P.E. Kloeden, E. Platen, Numerical Solution of Stochastic Differential Equations, Springer Science & Business Media, Apr 2013.
[8] I. Karatzas, S.E. Shreve, Brownian Motion and Stochastic Calculus, Springer, New York, NY, 1998, pp. 47–127.
[9] V. François-Lavet, P. Henderson, R. Islam, M.G. Bellemare, J. Pineau, An introduction to deep reinforcement learning, arXiv preprint, arXiv:1811.12560, Nov 2018.
[10] Z.Q. Liu, W. Cai, Zhi-Qin John Xu, Multi-scale deep neural network (MscaleDNN) for solving Poisson-Boltzmann equation in complex domains, Commun. Comput. Phys. 28 (5) (2020) 1970–2001.
[11] B. Oksendal, Stochastic differential equations, in: Stochastic Differential Equations, Springer, Berlin, Heidelberg, 2003, pp. 65–84.
[12] E. Pardoux, S. Peng, Backward stochastic differential equations and quasilinear parabolic partial differential equations, in: Stochastic Partial Differential Equations and Their Applications, Springer, Berlin, Heidelberg, 1992, pp. 200–217.
[13] G. Pavliotis, A. Stuart, Multiscale Methods: Averaging and Homogenization, Springer Science & Business Media, Jan 2008.
[14] M. Raissi, Forward-backward stochastic neural networks: deep learning of high-dimensional partial differential equations, arXiv preprint, arXiv:1804.07010, Apr 2018.
[15] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Nov 2018.