Journal of Computational Physics (2021)

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

FBSDE based Neural Network Algorithms for High-Dimensional Quasilinear Parabolic PDEs

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

[a]*Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA*

ARTICLE INFO

Article history:
Received TBD
Received in final form TBD
Accepted TBD
Available online TBD

Communicated by TBD

Forward and backward SDEs, Feynman-Kac formula, Multi-scale deep neural network, Quasilinear parabolic equations, Pardoux-Peng theory. *2010 MSC:* 35Q68, 65N99, 68T07

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 quasilinear parabolic partial differential equations (PDEs), which are related to the FBSDEs by the Pardoux-Peng theory. The algorithms rely on a learning process by minimizing the pathwise difference between two discrete stochastic processes, defined by the time discretization of the FBSDEs and the DNN representation of the PDE solutions, respectively. The proposed algorithms are shown to generate DNN solutions for a 100-dimensional Black–Scholes–Barenblatt equation, accurate in a finite region in the solution space, and has a convergence rate similar to that of the Euler–Maruyama discretization used for the FBSDEs. As a result, a Richardson extrapolation technique over time discretizations can be used to enhance the accuracy of the DNN solutions. For time oscillatory solutions, a multi-scale DNN is shown to improve the performance of the FBSDE DNN for high frequencies.

1. Introduction

The relationship between stochastic processes and the solution of partial differential equations represents 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 PDEs as a result of the Kolmogorov backward equation for the generator of the stochastic process for the former [10] and the Dynkin formula for the latter [8]. The recent work by Pardoux–Peng [9] has extended the concept of the classic linear Feynman–Kac formula to a nonlinear version, which connects

*Corresponding author: cai@smu.edu

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 the mathematical finance in option pricing [4].

Meanwhile, in the field of scientific computing, this connection between SDEs and quasilinear PDEs has inspired new approaches of solving high dimensional parabolic partial differential equations (PDEs), which are ubiquitous in material sciences such as the Allen–Cahn equation for phase transition, and quantum mechanics such as the Schrodinger equation as well as the Black–Scholes equation for option pricing and the Hamilton-Jacobi-Bellman equation for optimal control. 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 the unknowns in the discretized systems for the PDEs grows exponentially in terms of the dimension of the problem. Recently, machine learning approaches based on the deep neural network 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 are discretized in time by the classic Euler–Maruyama scheme [6]. The first such an attempt was done in [2], 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 [11] 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, improved learning schemes will be proposed based on a similar approach in [11], but with clearer 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 relation between FBSDEs and quasilinear parabolic PDEs, with an emphasis on the relation between the classic Feynman–Kac formula and the nonlinear version represented by the Pardoux–Peng theory. Section 3 will first review the algorithms proposed in [2] and [11], 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. Enhanced numerical accuracy by Richardson extrapolations and multi-scale DNNs for PDEs with oscillatory solutions in time will also be discussed. Finally, a conclusion will be 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 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 [9], 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. Let $\{\mathcal{F}_t : 0 \le t \le T\}$ be its natural filtration on the time interval $[0, T]$. Then, we have the equations of stochastic 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 [8] 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 [5]. 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 [5], there exists a 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 $Z_t^\star$ [5].

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 [8] 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 neural network algorithms for quasilinear parabolic PDEs

The learning of the 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 unknowns,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 [2]

The Deep BSDE 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 been 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 has been shown to give convergent numerical results for various high dimensional parabolic equations [2] and a posteriori estimate suggests strong convergence of half order [3].

Remark 1. The Deep BSDE method from [2] trains the network for the specific initial data $X_0 = x_0$ and yield 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 parameter if higher accuracy of the PDE solution is desired.

*3.1.2. FBSNNs [11] (**Scheme 1**)*

The FBSNNs trains 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

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

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

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

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 [11] 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. Improved FBSDE based deep neural network algorithms for quasilinear parabolic PDEs

In this section, we propose improved algorithms for the FBSDEs based deep neural networks similar to the approach in [11], 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, calculate a reference quantity 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 reference 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 composite 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 $X^{(1)}_0 = X^{(2)}_0 = x_0$ and we compute

$$Y^{(1)}_0 = Y^{(2)}_0 = u_\theta(t_0, x_0), \quad Z^{(1)}_0 = Z^{(2)}_0 = \nabla u_\theta(t_0, x_0) \tag{33}$$

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

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

$$\begin{aligned}
X^{(1)}_{n+1} &= X^{(1)}_n + \mu(t_n, X^{(1)}_n, Y^{(1)}_n, Z^{(1)}_n)\Delta t_n + \sigma(t_n, X^{(1)}_n, Y^{(1)}_n)\Delta W_n,\\
Y^{(1)}_{n+1} &= u_\theta(t_{n+1}, X^{(1)}_{n+1}),\\
Z^{(1)}_{n+1} &= \nabla u_\theta(t_{n+1}, X^{(1)}_{n+1}),
\end{aligned} \tag{34}$$

$$\begin{aligned}
X^{(2)}_{n+1} &= X^{(2)}_n + \mu(t_n, X^{(2)}_n, Y^{(2)}_n, Z^{(2)}_n)\Delta t_n + \sigma(t_n, X^{(1)}_n, Y^{(1)}_n)\Delta W_n,\\
Y^{(2)}_{n+1} &= Y^{(2)}_n + \phi(t_n, X^{(2)}_n, Y^{(2)}_n, Z^{(2)}_n)\Delta t_n + (Z^{(2)}_n)^T\sigma(t_n, X^{(2)}_n, Y^{(2)}_n)\Delta W_n,\\
Z^{(2)}_{n+1} &= \nabla u_\theta(t_{n+1}, X^{(2)}_{n+1}).
\end{aligned} \tag{35}$$

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

$$L_3[u_\theta; x_0] = \frac{1}{M}\sum_{\omega}\left[\frac{1}{N}\sum_{n=1}^{N}\left\|Y^{(1)}_n - Y^{(2)}_n\right\|^2 + \beta_1\left\|Y^{(1)}_N - g(X^{(1)}_N)\right\|^2 + \beta_2\left\|Z^{(1)}_N - \nabla g(X^{(1)}_N)\right\|^2\right], \tag{36}$$

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

4. Numerical results

In this section, we will carry out several tests on Scheme 1 from [11] 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 [11] 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 [11]. 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 $\beta_1 = \beta_2 = 0.02$.

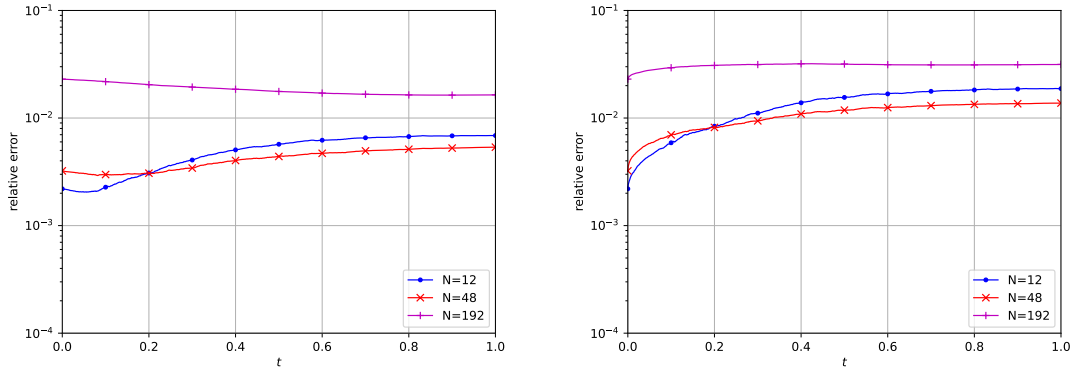
Illustration of the training results in the high-dimensional space is provided along the sample paths. When the training is finished, we randomly generate 1000 sample paths for verification of the accuracy, with a finer time discretization with time steps $\Delta t_n = 1/1000$. For each (discretized)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

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

The mean and the standard deviation (SD) of each e_n can also be calculated.

4.1.1. Scheme 1 from [11]

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).

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.

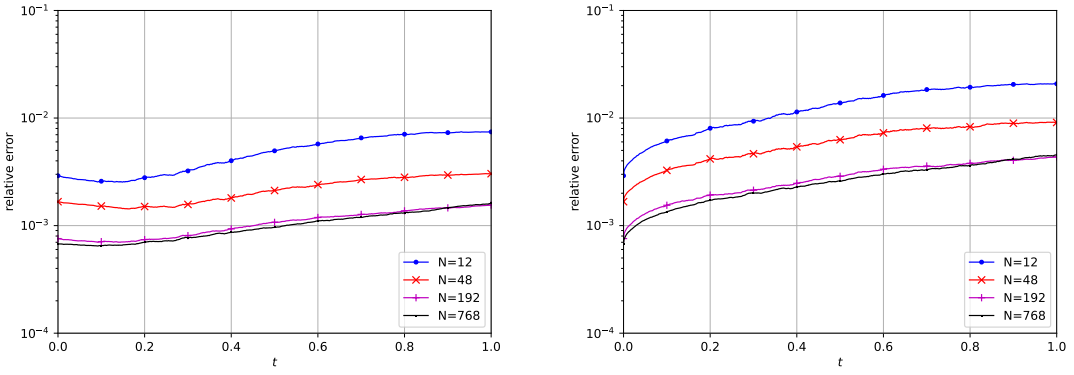
Both the results in Fig. 2 and Fig. 3 show the convergence of the new Scheme 2 and Scheme 3, respectively, in contrast to the degeneracy of the accuracy of Scheme 1 when the time discretization is refined. For both new schemes, we can see improvement of the accuracy from $N = 48$ to $N = 192$ is close to the one from $N = 12$ to $N = 48$, but the improvement of $N = 768$ over $N = 192$ is a little 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 $N = 192$.

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).

4.1.3. Richardson extrapolation for higher order accuracy

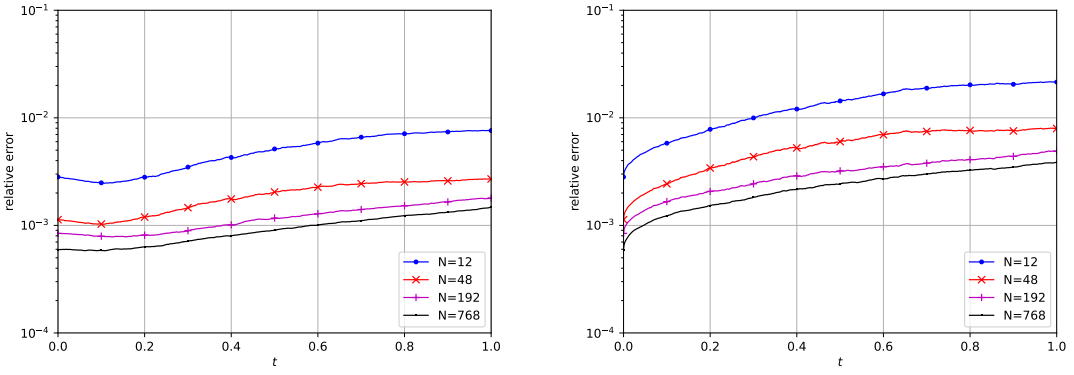
In Section 4.1.2 we have seen that Scheme 2 and Scheme 3 have the convergence behavior as the Euler–Maruyama scheme, so we can assume that the truncation error may have the following asymptotic ansatz

$$u_\theta^N - u = C_1 N^{-\frac{1}{2}} + C_2 N^{-1} + O(N^{-\frac{3}{2}}), \tag{42}$$



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

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



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

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

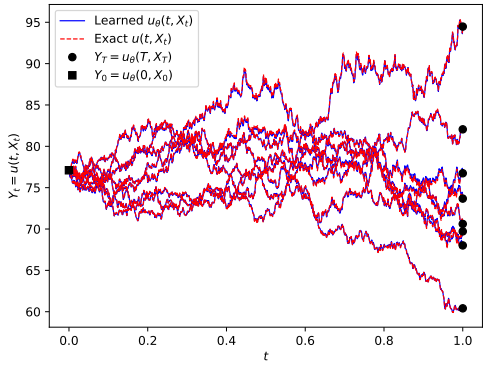
where the leading term $C_1 N^{-\frac{1}{2}}$ dominates the error when N is sufficiently large. If this holds for both u_θ^N and u_θ^{4N} for some constants C_1 and C_2, then we can define an extrapolated solution

$$\begin{aligned} u_{\text{ex}}^{4N} &= 2u_\theta^{4N} - u_\theta^N \\ &= u - \frac{C_2}{2} N^{-1} + O(N^{-\frac{3}{2}}) \end{aligned} \tag{43}$$

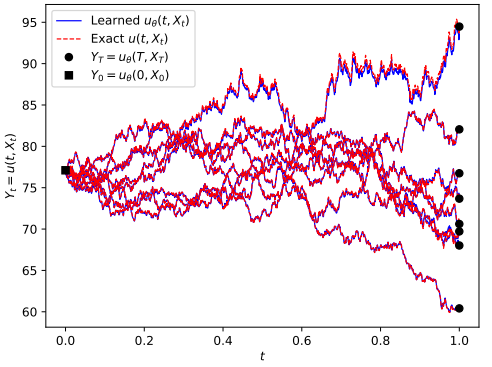
as an improved approximation to the solution.

For the model problem (37), the Richardson 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 1 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 both Scheme 2 and Scheme 3. Due to training difficulties, the improvement for using extrapolation on $N = 768$ is marginal, but still exists.

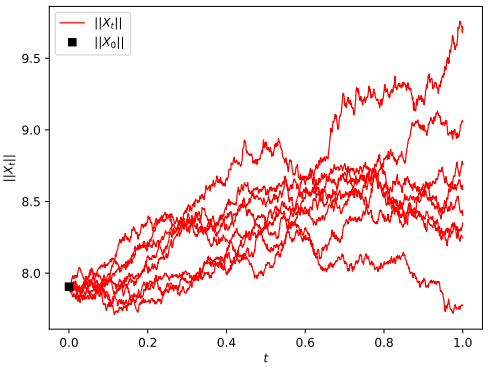
Note that the Richardson extrapolation approach usually may not work for the whole time interval along the entire sample paths. For instance, the values at $t = T$ are subject to explicit fitting of the terminal condition from the loss



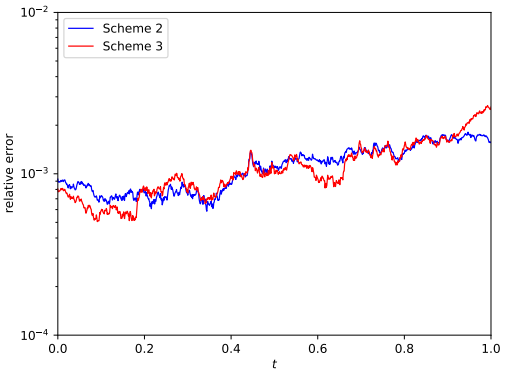
(a) Prediction by Scheme 2



(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, $N = 192$.

functions (32) and (36), so we cannot expect a general constant C_1 in (42) 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 $0 \le t \le 0.1$.

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

Table 1: Relative error of Y_0 from the network approximation and extrapolation.

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.,

$$(\tilde{x}_0)_j = (x_0)_j \cdot (1 + \varepsilon_j), \quad 1 \le j \le d = 100, \tag{44}$$

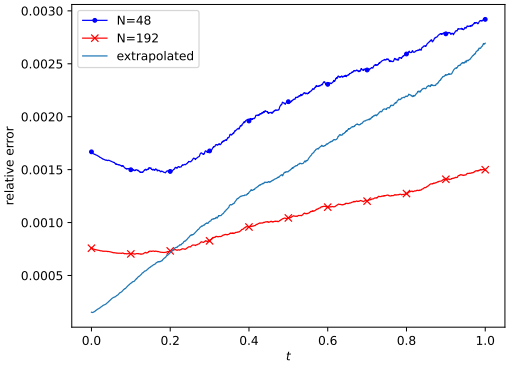


Fig. 5: Mean relative error of the extrapolation $u_\theta^{192}(t, X_t)$ for $0 \le t \le T$, using Scheme 2.

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 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 pathes 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 neighbourhood around x_0 for the whole time interval $0 \le t \le T$.

4.2. Multiscale DNN for the BSB equation with temporal oscillations

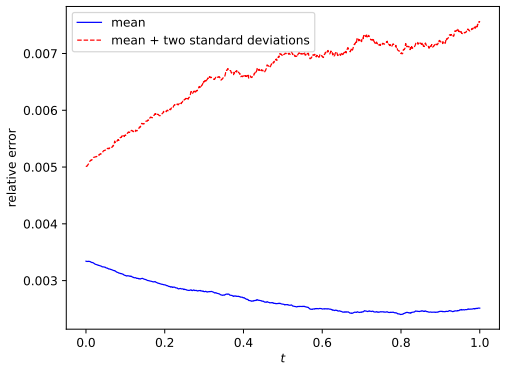
In a recent work [7], a multi-scale DNN (MscaleDNN) was proposed, which consists of a series of parallel normal sub-networks, each of which receiving a scaled version of the input, and outputs of the sub-networks are combined to form the final output of the MscaleDNN (see Fig. 7). The individual sub-networks 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 [7] 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 again we use the *sine* function for the activation function, i.e.,

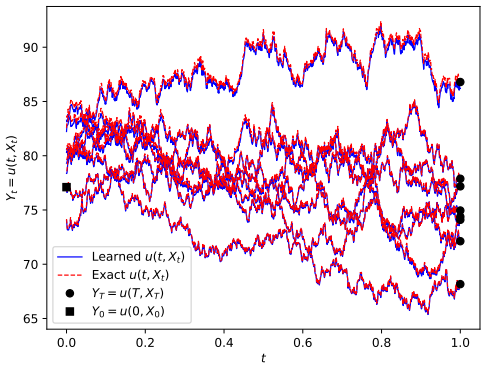
$$\sigma(x) = \sin(x). \tag{45}$$

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)

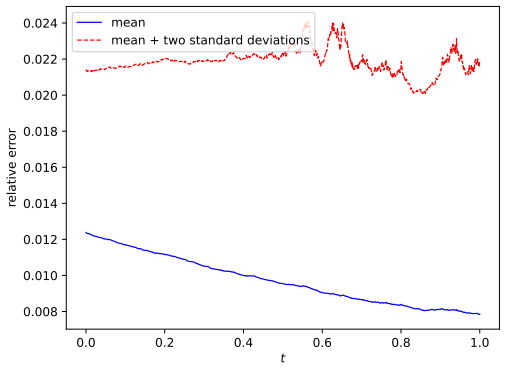
$$f(\boldsymbol{x}) \sim \sum_{i=1}^{M} \boldsymbol{W}_i^{[L]} f_{\theta^{n_i}}(\boldsymbol{\alpha}_i \cdot \boldsymbol{x}) + \boldsymbol{b}^{[L]}, \tag{46}$$



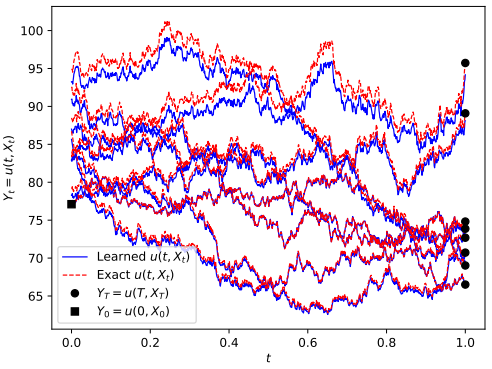
(a) Relative error, $R = 0.25$



(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, $N = 192$.

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 [7].

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 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 training difficulty:

$$\begin{aligned} \partial_t u + \frac{1}{2}\mathrm{Tr}[\sigma^2 \nabla\nabla u] &= \phi, \\ u(T, x) &= g(x), \end{aligned} \tag{47}$$

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

$$g(x) = \|x\|^2 \left(1 + \alpha \sin\left(\beta S_1 - \gamma T\right)\right), \tag{48}$$

$$\phi(t, x, u, \nabla u) = r(u - \nabla u \cdot x) + \alpha e^{(r+\sigma^2)(T-t)} P(t, x), \tag{49}$$

$$P(t, x) = (r\beta S_1 S_2 - \gamma S_2 + 2\sigma^2 \beta S_3)\cos(\beta S_1 - \gamma t) - \frac{\sigma^2 \beta^2}{2} S_2^2 \sin(\beta S_1 - \gamma t), \tag{50}$$

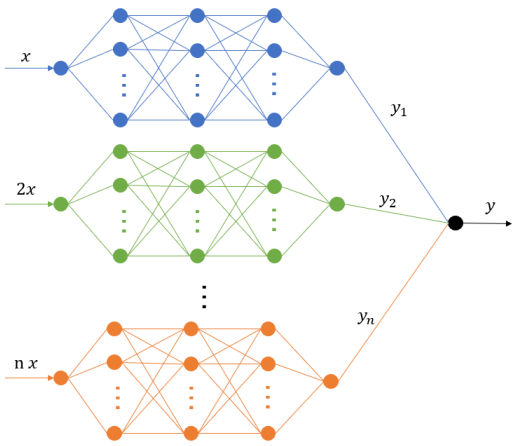


Fig. 7: Illustration of a MscaleDNN.

where each $S_j = \sum_{i=1}^{d} x_i^j$, and α, β and γ are parameters to be tuned. The modified PDE (47) has a solution

$$u(t,x) = e^{(r+\sigma^2)(T-t)}\|x\|^2 \left(1 + \alpha \sin\left(\beta S_1 - \gamma t\right)\right), \tag{51}$$

and corresponds to the FBSDEs

$$\begin{aligned} dX_t &= \sigma \mathrm{diag}(X_t) dW_t, \\ X_0 &= x_0, \\ dY_t &= \left(r(Y_t - Z_t \cdot X_t) + \alpha e^{(r+\sigma^2)(T-t)} P(t, X_t)\right) dt + \sigma Z_t^T \mathrm{diag}(X_t) dW_t, \\ Y_T &= g(X_T). \end{aligned} \tag{52}$$

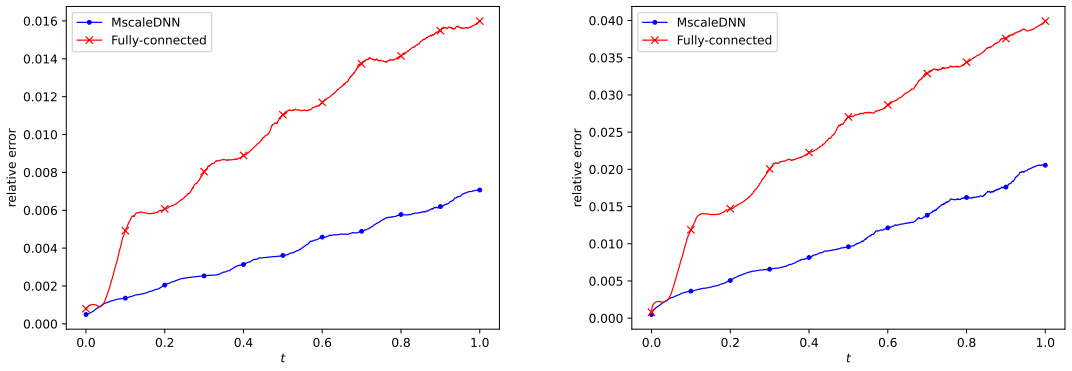
We apply $\alpha = 0.025, \beta = 0.25$ and $\gamma = 32$ to the above equation. During the training process, we use the same settings for the fully-connected DNN as in previous tests. For the MscaleDNN, the network is divided into 4 sub-networks, each having 5 hidden layers with 64 neurons per layer,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

$$(3^0 t, x), \quad (3^1 t, x), \quad (3^2 t, x), \quad (3^3 t, x), \tag{53}$$

so that a wider range of frequency of t can be captured with the MscaleDNN. When applying Scheme 2 and $N = 48$, 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. 9.

5. Conclusion

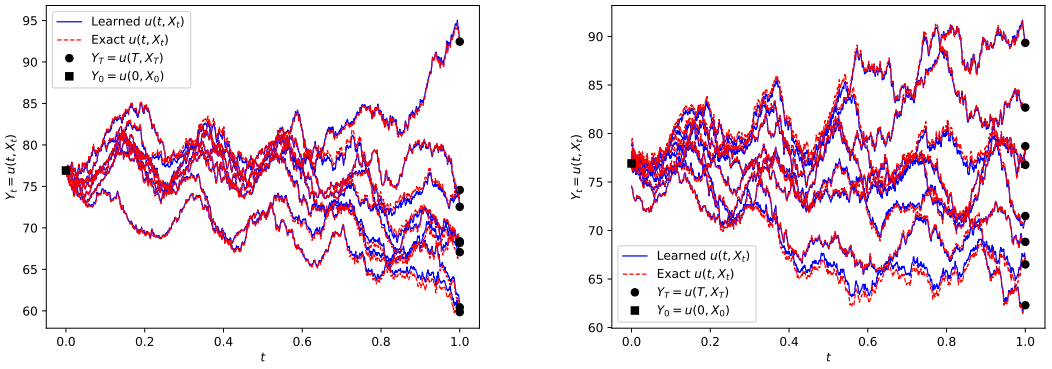
In this paper, we have proposed two FBSDE based DNN algorithms for high dimensional quasilinear parabolic equations. The key component of the proposed algorithms is the loss function used, consisting of, in addition to the terminal condition of the PDE, the pathwise difference of two convergent stochastic processes from either the



(a) Error Mean

(b) Error 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 $N = 48$.



(a) Starting from $(0, x_0)$.

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

Fig. 9: Prediction of 8 sample paths for problem with oscillation (47), using the MscaleDNN with Scheme 2 and $N = 192$.

discretized SDEs or the PDEs DNN solution. As the two stochastic processes converge to the same stochastic process in the Pardoux–Peng theory, the new algorithms are able to demonstrate nearly the half-order strong convergence of the underlying Euler–Maruyama scheme for the SDEs. As a result, the Richardson extrapolation method can be used, which confirms the convergence order of the DNN solutions and further enhances the resulting accuracy of the DNN solution of the PDE. For PDEs with time oscillatory solutions, the MscaleDNN is shown to provide an enhancement of the resulting accuracy as well.

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

Acknowledgments

The authors would like to thank Weinan E, Jiequn Han, and Maziar Raissi for helpful discussions on the topic of this work.

References

[1] Doob JL. Classical potential theory and its probabilistic counterpart: Advanced problems. Springer Science & Business Media; 2012 Dec 6.

[2] Han J, Jentzen A, Weinan E. Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences. 2018 Aug 21;115(34):8505-10.

[3] Han J, Long J. Convergence of the deep BSDE method for coupled FBSDEs. Probability, Uncertainty and Quantitative Risk. 2020 Dec;5(1):1-33.

[4] El Karoui N, Peng S, Quenez MC. Backward stochastic differential equations in finance. Mathematical finance. 1997 Jan;7(1):1-71.

[5] Karatzas I, Shreve SE. Brownian Motion and Stochastic Calculus 1998 (pp. 47-127). Springer, New York, NY.

[6] Kloeden PE, Platen E. Numerical solution of stochastic differential equations. Springer Science & Business Media; 2013 Apr 17.

[7] Liu, Z.Q., Wei Cai & Zhi-Qin John Xu, Multi-Scale Deep Neural Network (MscaleDNN) for Solving Poisson-Boltzmann Equation in Complex Domains. Communications in Computational Physics. 28(5), 1970-2001, 2020.

[8] Oksendal B. Stochastic differential equations. InStochastic differential equations 2003 (pp. 65-84). Springer, Berlin, Heidelberg.

[9] Pardoux E, Peng S. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In Stochastic partial differential equations and their applications 1992 (pp. 200-217). Springer, Berlin, Heidelberg.

[10] Pavliotis G. Stochastic Processes and Applications, 2014, Springer.

[11] Raissi M. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. arXiv preprint arXiv:1804.07010. 2018 Apr 19.