Received 26 April 2016; | Revised 6 June 2016; | Accepted 6 June 2016

DOI: xxx/xxxx

ARTICLE TYPE

Artificial neural networks for solving elliptic differential equations with boundary layer

Dongfang Yuan[1] | Wenhui Liu[2] | Yongbin Ge[3] | Guimei Cui[2] | Lin Shi[1] | Fujun Cao*[1]

[1]School of Science, Inner Mongolia University of Science and Technology, Baotou, P.R.China
[2]School of Information Engineering, Inner Mongolia University of Science and Technology, Baotou, P.R.China
[3]School of Mathematics and Statistics, Ningxia University, Yinchuan, P.R.China

Correspondence
*Corresponding author. Email: caofujun@imust.edu.cn

Present Address
No.7, alding street, Baotou, Inner Mongolia

Summary

In this paper, we consider the artificial neural networks for solving the elliptic differential equation with boundary layer, in which the gradient of the solution changes sharply near the boundary layer. The solution of the boundary layer problems poses a huge challenge to both traditional numerical methods and artificial neural network methods. By theoretical analyzing the changing rate of the weights of first hidden layer near the boundary layer, a mapping strategy is added in traditional neural network to improve the convergence of the loss function. Numerical examples are carried out for the 1D and 2D convection-diffusion equation with boundary layer. The results demonstrate that the modified neural networks significantly improve the ability in approximating the solutions with sharp gradient.

KEYWORDS:
Artificial neural network ; Boundary layer problems ; Elliptic differential equation

1 | INTRODUCTION

Differential equations, especially partial differential equations, can be used to describe the basic physical laws of a given system, so they play an important role in many disciplines. The method for solving differential equations has also evoked extensively attention of researchers in last decades. Various numerical methods, such as finite difference method (FDM), finite volume method (FVM), finite element method (FEM), etc, have been developed to approximate the solution of differential equations. However, numerical solution for high-dimensional partial differential equations have been a longstanding challenge, owing to the explosion in the number of grid points and the demand for reducing time step size[1]. With the growth and development of machine learning algorithms, the artificial neural networks (ANNs) has been becoming a powerful tool and emerged as an alternative method for solving differential equations arising in various fields including fluid dynamics and quantitative finance, especially where large data sets are involved.

ANNs can be used to approximate the solutions of partial differential equations (PDEs), due to their ability to efficiently approximate arbitrary functions, named as the universal approximation theorem[2], which states that an artificial neural network containing a single hidden layer can approximate any arbitrarily complex function with enough neurons. The main idea of the approach lies on fitting the governing equations[3,4,5] or the energy minimization formulation of the governing equation[6,7] and the boundary conditions at randomly selected points in the domain and on the boundary. ANNs is trained on a set of matching inputs and outputs by seeking to minimize an error function formulated in terms of the differential operator. The ANNs approach has many advantages over the traditional numerical method, such as, using random points in the domain and avoiding to generate spatio-temporal grids, approximating the trial solutions directly and regardless of the dimension of the problem, etc. The classical artificial feed-forward neural network[3,4,5,8?] employs a feed-forward neural network as the basic approximation element, whose parameters (weights and biases) are adjusted to minimize an appropriate error function by defining a loss function which minimizes the residuals of the governing equations at a chosen set of training points. Further, some functional link artificial neural network is developed to solve differential equation with initial and boundary value problems[11,12,13,14]. In these methods, the hidden layer is replaced by using orthogonal polynomials, such as Chebyshev[12,13], Legendre[14], to improve computationally efficient and to achieve higher convergence speed. Rizaner[10] proposed an unsupervised radial basis function networks and achieve high accuracy for the first order initial value problems. Recently, the mesh-free deep learning algorithm with multi-layer neural networks are proposed to solve high dimensional PDEs and approximate the unknown solution[6,1,15,16,17,18]. The deep learning based mesh-free method also be applied to solve elliptic PDE with complex geometries[9] and the interface problems with discontinuous and high-contrast coefficient[7]. Raissi et al.[19] introduced physics informed neural networks to solve the general nonlinear partial differential equations. They also considered the data-driven discovery of the parameterized partial differential equation[20]. Dwivedi et al.[21] proposed a physics informed extreme learning machine and applied to stationary and time-dependent linear partial differential equations. However, current neural network architectures for such implicit neural representations are incapable of modeling signals with fine detail, and fail to represent a signal's spatial and temporal derivatives, despite the fact that these are essential to many physical signals defined implicitly as the solution to partial differential equations. Sitzmann et al.[22] proposed a sinusoidal periodic activation functions for implicit neural representations to solve challenging boundary value problems, such as particular Eikonal equations, the Poisson equation, and the Helmholtz and wave equations.

In this paper, we consider the differential equation with boundary layer, in which the gradient changes dramatically and greatly challenge the performance of numerical methods. The theoretical analysis and numerical experiments show that the traditional neural network method is difficult to converge for this kind of problem. By analyzing the changing rate of the weights near the boundary layer, a mapping strategy coupling with the initialization strategy is proposed, which can improve the convergence rate of neural network even if there is a large gradient. Numerical experiments of one dimensional and two dimensional convection-diffusion equation with boundary layer are tested to verify the performance of the present method.

The rest of this paper is organized as follows. In Section 2, we formulate the process of artificial neural network for solving differential equation. Sections 3 is devoted to analyze the factors affecting the convergence of neural network. Numerical examples are carried out in Section 4 to demonstrate the efficiency of the presented neural network. A concluding remark is given in Section 5.

2 | PROBLEM FORMULATION

The general elliptic linear differential equation has the form

$$G(\vec{x}, u(\vec{x}), \nabla u(\vec{x}), \nabla^2 u(\vec{x})) = 0, \quad \forall \vec{x} \in D, \tag{1}$$

where $\vec{x} \in \mathcal{R}^n$ is the independent variable over the domain $D \in \mathcal{R}^n$, and $u(\vec{x})$ is the unknown solution. The essential boundary conditions (B.Cs) (for instance Dirichlet and/or Neumann), are specified on the boundary of the domain ∂D.

The above differential equation is relaxed to a discretized version and for a discretization of the domain $\mathbb{D} = \{x^{(i)} \in D; i = 1, 2, \cdots, m\}$, then (1) is relaxed to hold only at these points

$$\hat{G}(\vec{x}^{(i)}, u(\vec{x}^{(i)}), \nabla u(\vec{x}^{(i)}), \nabla^2 u(\vec{x}^{(i)})) = 0, \quad \forall i = 1, \cdots, m, \tag{2}$$

subject to the constraints imposed by the B.Cs.

Consider an artificial feed-forward network with n inputs, m outputs and L hidden layers with k^l units, $l = 1, 2, \cdots, L$. Let $\mathbb{N} = \{N(\cdot, \theta) | \theta \in \Theta\}$ denote the set of all expressible functions by the neural network parameterized by $\theta \in \Theta$, where Θ is set of θ. Denote $u_t(\vec{x}, \theta)$ as the trial solution with adjustable parameters θ, then the partial differential equation (1) can be transformed into a constrained optimization problem

$$\min_{\theta} \sum_{\vec{x}^i \in \mathbb{D}} \hat{G}\Big(\vec{x}^{(i)}, u_t(\vec{x}^{(i)}, \theta), \nabla u_t(\vec{x}^{(i)}, \theta), \nabla^2 u_t(\vec{x}^{(i)}, \theta)\Big)^2, \tag{3}$$

subject to the constraints imposed by the B.Cs.

Once the domain is discretized into a finte number of training points $\vec{x}^i$, then the approximations to the solutions, $u_t(\vec{x})$ can be identified by finding the set of weights and biases θ, such that the neural network loss function is minimized on the training points. The full loss function that we use is

$$\mathcal{L}(\theta) = \omega_1 \sum_i \hat{G}(\vec{x}^{(i)}, u_t(\vec{x}^{(i)}), \nabla u_t(\vec{x}^{(i)}), \nabla^2 u_t(\vec{x}^{(i)}))^2 \\ + \omega_2 \sum_{B.C.} (\nabla^\theta u_t(\vec{x}^{(i)}) - g(\vec{x}))^2, \tag{4}$$

where the second term represents the sum of the squares of the boundary conditions, defined at the boundaries $\vec{x}_b$. ω_1 and ω_2 are weighting coefficients to adjust the influence of the governing equation and boundary conditions , when there is boundary layer near the boundary. An objective function measures how well a neural network approximation satisfies the differential equation and is used as a compass to train the neural network. The objective function is selected either directly from the differential equation or an equivalent formulation. This is analogous to a finite difference method directly discretizing the differential equation and a finite element method using the variational formulation.

We hope to minimize the error of the loss function $\mathcal{L}(\theta)$ by learning a set of parameters on the training set $\mathbb{D}^{train}$,

$$\theta^* = \underset{\theta}{argmin} \mathcal{L}(\theta), \quad x \in \mathbb{D}^{train}. \tag{5}$$

The optimal parameters can be obtained numerically by a number of different optimization methods, such as back propagation or the quasi-Newton Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS). Regardless of the method, once the parameters θ^* have been attained, the trial solution $u_t(\vec{x}, \theta^*)$ is a smooth approximation to the true solution that can be evaluated continuously on the domain.

The strategy for defining the approximate solution $u_t(\vec{x}, \theta)$ is a trial solution composed of two or three parts to satisfy the Dirichlet and Neumann boundaries[4], respectively.

$$u_t(\vec{x}, \theta) = A_D(\vec{x}) + A_N(\vec{x}, N) + \mathsf{G}(\vec{x}, N), \tag{6}$$

where A_D is designed to satisfy the Dirichlet B.Cs, A_N ensures satisfaction of B.Cs on the Neumann boundary while not interfering with the Dirichlet B.Cs, the term G to return zero on the boundary while being a function of the ANN output N for all points inside the domain.

3 | STRUCTURE OF NEURAL NETWORK

3.1 | Feed-forward networks

A feed-forward network consists of an input layer, an output layer and an arbitrary number of intermediary hidden layers, where all the neurons (units) in adjacent layers are connected with each other. It can be used to represent a function $u : \mathcal{R}^n \to \mathcal{R}^m$ by using n neurons in input layer and m neurons in the output layer. Index the input layer as 0, and the output layer as L, and denotes the number of neurons in each layer by $k_0, k_1, k_2, \cdots, k_L$. To each connection between the i-th neuron in layer $l-1$ and the j-th neuron in layer l, we associate a weight w^l_{ji} and to each neuron in the layers $0 < l \leq L$, we associate a bias b^l_j, $j = 1, \cdots, k_L$, as in Fig.1 . Moreover, we define an nonlinear activation function $\sigma^l : \mathcal{R} \to \mathcal{R}$ between the layers l and $l-1$. Two frequently used activation functions are the rectified linear unit (ReLU) and the Sigmoid functions, which are defined as $\sigma(x) = \max(0, x)$ and $\sigma(x) = (1 + e^{-x})^{-1}$, respectively. We use the sigmiod activation function which is preferable due to its smoothness, and its first order derivative is given as $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. Then, the values at each neuron can be written in terms of the activation function applied to a linear combination of the neurons in the previous layer given by the corresponding weights and biases, i.e.,

$$u^l_k = \sigma^l(\sum_{j=1}^{k_{l-1}} w^l_{kj} u^{l-1}_j + b^l_k), \quad k = 1, 2, \cdots, k_L. \tag{7}$$

This can be written more compactly in matrix form as:

$$\mathbf{u}^l = \sigma^l(\mathbf{W}^l \mathbf{u}^{l-1} + \mathbf{b}^l), \quad \mathbf{z}^l = \mathbf{W}^l \mathbf{u}^{l-1} + \mathbf{b}^l, \quad l = 1, 2, \cdots, L,$$

where $\mathbf{W}^l$ is a matrix of weights corresponding to the connections between $l-1$ and l, $\mathbf{u}^l = [u^l_j]$ and $\mathbf{b}^l = [b^l_j]$ are column vectors and the activation function is applied element-wise.

Equipped with those definitions, we are able to define a continuous function $N(x)$ by a composition of linear transforms and activation functions,

$$N(x) = \mathbf{W}^L \sigma\big(\mathbf{W}^{L-1} \circ\circ\circ \sigma(\mathbf{W}^1 \mathbf{X} + \mathbf{b}^1) + \circ\circ\circ + \mathbf{b}^{L-1}\big). \tag{8}$$

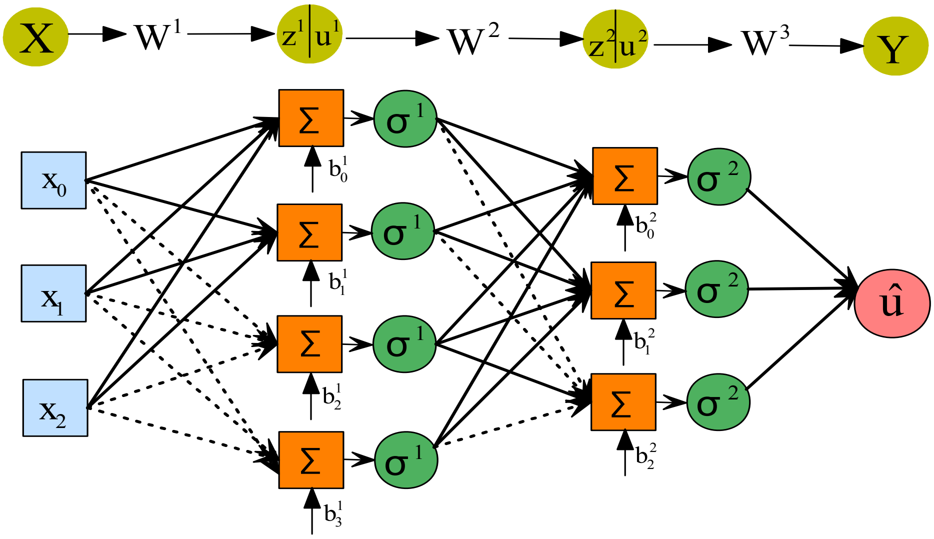

FIGURE 1 Partitioning of ANN nodes and weight

Denoting all the undetermined coefficients (e.g., $\mathbf{W}^l$ and $\mathbf{b}^l$) in (8) as $\theta \in \Theta$, where $\theta = \{\bigcup_{l=1,2,\cdots,L} (\mathbf{W}^l \bigcup \mathbf{b}^l)\}$ and Θ is the space of θ. The neural network representation of a continuous function can be viewed as

$$N = N(x, \theta). \tag{9}$$

Notice that θ is a high-dimensional vector and θ_k is k-th component of θ.

3.2 | The back-propagation algorithm

The back-propagation algorithm enables the application of gradient-based minimization algorithms, where a loss function based on the output of the neural network is to be minimized. The idea is to perform the chain rule starting with the last layer and store the intermediary values in a computational graph where the order of the layers is reversed. The optimization problem that one encounters often takes the form as in (4). After we get the approximation of the gradient with respect to θ_k, we can update each component of θ_k as

$$\theta_k^{n+1} = \theta_k^n - \eta \frac{\partial \mathcal{L}(N(x, \theta_k))}{\partial \theta_k}\Big|_{\theta_k = \theta_k^n}. \tag{10}$$

Considering the partial derivatives of the $N(X, \theta)$ respect to inputs X and the wight coefficients between inputs and the first hidden layer, $\mathbf{W^1}$, we consider

$$\frac{\partial \mathcal{L}(N(x, \theta_k))}{\partial \theta_k} = \frac{\partial \mathcal{L}(N(x, \theta_k))}{\partial N} \frac{\partial N}{\partial \theta_k}. \tag{11}$$

$$\frac{\partial N}{\partial w^1_{i,j}} = \frac{\partial N}{\partial \vec{u}^2} \frac{\partial \vec{u}^2}{\partial \vec{z}^2} \frac{\partial \vec{z}^2}{\partial \vec{u}^1_i} \frac{\partial \vec{u}^1_i}{\partial \vec{z}^1_i} \frac{\partial \vec{z}^1_i}{\partial w^1_{i,j}}. \tag{12}$$

$$\frac{\partial N}{\partial x_j} = \frac{\partial N}{\partial \vec{u}^2} \frac{\partial \vec{u}^2}{\partial \vec{z}^2} \frac{\partial \vec{z}^2}{\partial \vec{u}^1} \frac{\partial \vec{u}^1}{\partial \vec{z}^1} \frac{\partial \vec{z}^1}{\partial x_j}. \tag{13}$$

Consider the activation function σ and its first and second derivatives are bounded. And the convergence of (10) requires that the the gradient of loss function respect to weights are bounded and tends to zero as the iteration process increasing, e.g.,

$$\frac{\partial \mathcal{L}(N(x, \theta_k))}{\partial \theta_k} \to 0, \quad \frac{\partial N}{\partial w^1_{i,j}} = \beta \to 0. \tag{14}$$

On the other hand, when there is a boundary layer in the problem, $x_j = \alpha$. The gradient near the boundary layer will change dramatically and the gradient value inevitably tends to be very large, e.g.,

$$\frac{\partial N}{\partial x_j}\Big|_{x_j = \alpha} = \kappa \to 10^n, \quad n \geq 2. \tag{15}$$

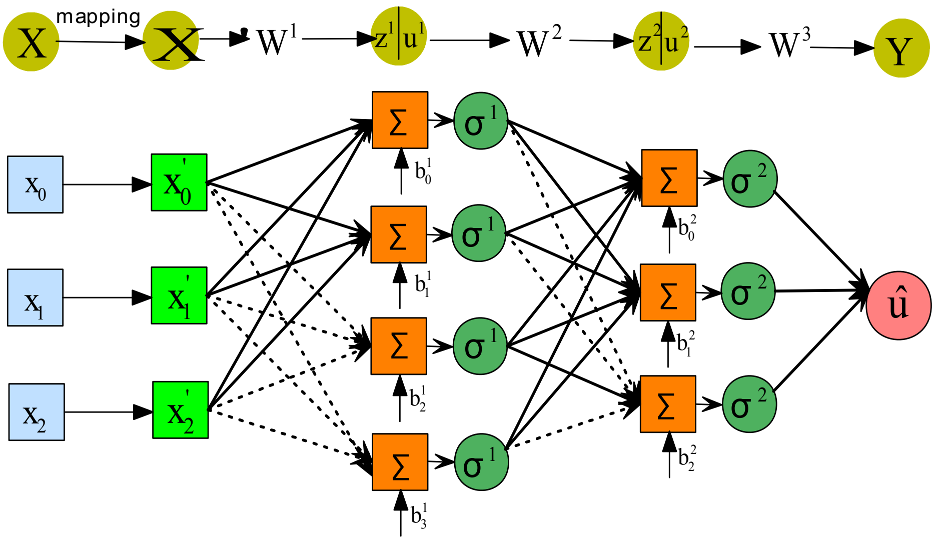

FIGURE 2 Partitioning of ANN with mapping

The changing rate of the coefficients near the boundary layer can be expressed as

$$\frac{\partial w^1_{i,j}}{\partial x_j} = \frac{\partial N}{\partial x_j} \Big/ \frac{\partial N}{\partial w^1_{i,j}} = \frac{\kappa}{\beta} \to 10^n, \quad n > 2. \tag{16}$$

It is in contradiction with the objection of the loss function converge to 0. Therefore, when there is a the boundary layer and the gradient near the boundary layer increases sharply, the conventional neural network model will not converge or the convergence efficiency will be seriously reduced. Rewrite the above equation as

$$\frac{\partial w^1_{i,j}}{\partial x_j} = \frac{\vec{W}^3 \sigma(\vec{u}^2)(1-\sigma(\vec{u}^2))\vec{W}^2 \sigma(\vec{u}^1_j)(1-\sigma(\vec{u}^1_j))\vec{W}^1_{:,j}}{\vec{W}^3 \sigma(\vec{u}^2)(1-\sigma(\vec{u}^2))\vec{W}^2_{i,:} \sigma(\vec{u}^1_i)(1-\sigma(\vec{u}^1_i))x_j}. \tag{17}$$

It can be seen that the changing rate of the weights is related to x_j. Because of the weights and the activation functions are bounded, the growth rate of the weights tends to infinity and only when x_j tends to zero.

3.3 | Mapping and initialization

In view of above analysis, a mapping strategy is proposed which maps the boundary layer position to 0. This strategy is expected to make the neural network converge near the boundary layer and improve the convergence efficiency. It is means that a mapping layer will be added between the input layer and first hidden layer compare to the traditional neural network, as shown in Fig.2 .

The simple mapping model is used

$$x'_{mapping} = x - x_\alpha, \tag{18}$$

where α is the location of boundary layer.

Meanwhile, considering the traditional neural network initialize the weight by the random sampling in $[0, 1]$, named as $\vec{W}^1_{unifom}$. From formula (4), in order to balance the gradient variability between the weights and the boundary layer, it is necessary to readjust and increase the initial value of the weights. Thus,

$$\vec{W}^1_{readjust} = \lambda \vec{W}^1_{unifom}, \tag{19}$$

where λ is adjustment coefficient, $\lambda \in [50, 100]$.

In view of the gradient varies intensively near the boundary layer, in order to distinguish the gradient variation effectively, it is necessary to increase the grid density near the boundary layer. We use a linear transformation function to generate non-uniform points as follows

$$x_i = \frac{i}{N_{train}} + \frac{\gamma}{\pi} \sin \frac{\pi i}{N_{train}}, \quad i = 1, 2, \cdots, N_{train}, \tag{20}$$

TABLE 1 The comparison of L_2 norm, average and variance of weights in three layers for Example 1

Weight	W^1			W^2			W^3		
	L_2	Aver	Var	L_2	Aver	Var	L_2	Aver	Var
Direct NN	7.89	0.69	2.34	8.77	0.01	1.28	4.52	0.007	1.08
Present NN	331.6	-9.67	104.4	8.29	-0.31	1.25	5.00	-0.13	1.34

where γ is adjustable coefficient to control the location of the clustered points.

4 | NUMERICAL EXPERIMENTS

To verify the performance of the present method, the one dimensional and two dimensional convection-diffusion equation with boundary layer are tested. The network we used to solve these problem is of two hidden layers with 10 units in each layer. There are a total of 140 parameters in model for 1D problems and 150 parameters in model for 2D problems, respectively. We randomly choose 20 points as the train set and 150 points as the test set for 1D problems. and 20×20 points as the train set and 100×100 points as the test set for 2D problems, respectively. For optimization, we use the Adam (adaptive momentum) optimizer which is based on stochastic gradient descent followed by a quasi-Newton method (L-BFGS) which builds an approximated Hessian at each gradient-descent step. The TensorFlow provides an efficient tool to calculate the partial derivatives in (10), which will be used in our implementation.

Example 1 We first consider the one dimensional convection-diffusion equation with boundary layer at $x = 1$,

$$\begin{cases} -\varepsilon \frac{d^2u}{dx^2} + \frac{du}{dx} + (1+\varepsilon)u(x) = 0, \quad x \in [0,1], \\ u(0) = 1 + e^{\frac{-(1+\varepsilon)}{\varepsilon}}, \quad u(1) = 1 + e^{-1}. \end{cases}$$

The analytic solution is

$$u(x) = e^{-x} + e^{\frac{(1+\varepsilon)(x-1)}{\varepsilon}}.$$

The loss function of this problem is

$$\begin{aligned} \mathcal{L}(\theta) = {} & \omega_1 \sum_{i=1}^{N_{int}} \Big(-\varepsilon \frac{d^2 N(x,\theta)}{dx^2} + \frac{dN(x,\theta)}{dx} - f(x) \Big)^2 \\ & + \omega_2 \sum_{x \in B.C.} N(x,\theta)^2, \end{aligned}$$

where $\omega_1 = 1$,$\omega_2 = 200$ and $\varepsilon = 0.005$. The stretching parameter is $\lambda = 0.9$, which controls the density distribution of sampling points in train set.

Fig. 3 displays the analytic and numerical solutions by direct neural networks and the present method, respectively. It is clear that the numerical solution from the direct neural networks, in Fig. 3 (a), is far from the analytic solution, which means that the direct NN method cannot effectively approximate the solution of the sharp gradient problem. By contrast, the numerical solution by presented method is consistent with the analytic solution on test set, as in Fig. 3 (b).

Fig. 4 compares the first and second derivatives of the present neural network with those by analytic solution. It is shown that the first and second derivatives of the approximate solution in the present neural network are very consistent with that of the analytical solution, respectively.

Fig. 5 (a) shows the absolute error between the solution by present neural network and the analytic solution. The average error on test set is about 4×10^{-3}. Fig. 5 (b) plots the convergence error of the loss function by the direct NN solution and the present method during the training process. It is obvious that the error of the direct NN method cannot be reduced as the number of iteration steps increases. On the other hand, the error of the presented method decreases with the number of iteration step increases.

Table 1 displays the L_2 norm, average and variance of weights on three layers. It is shown that the L_2 norm and variance of the weights of the first hidden layer are very different from those without mapping.

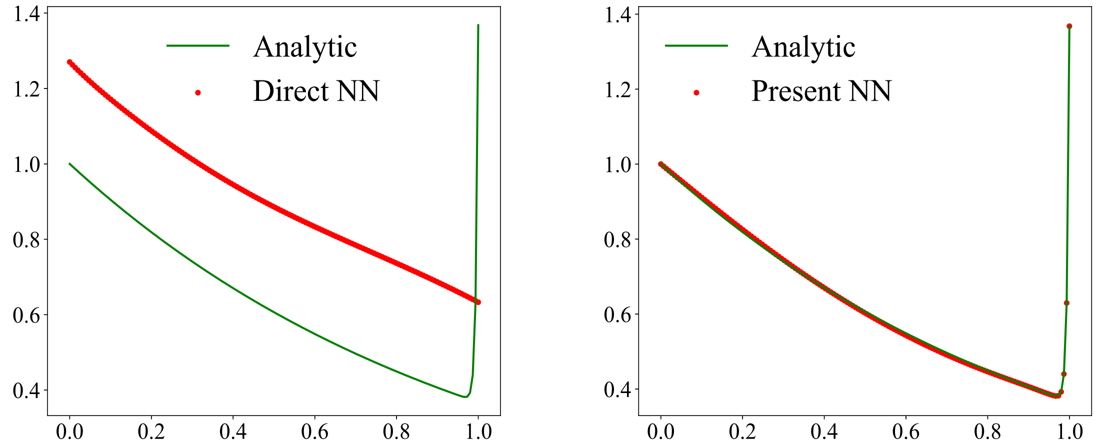

FIGURE 3 Comparison of numerical and analytical solution by direct NN method (a) and present method (b)

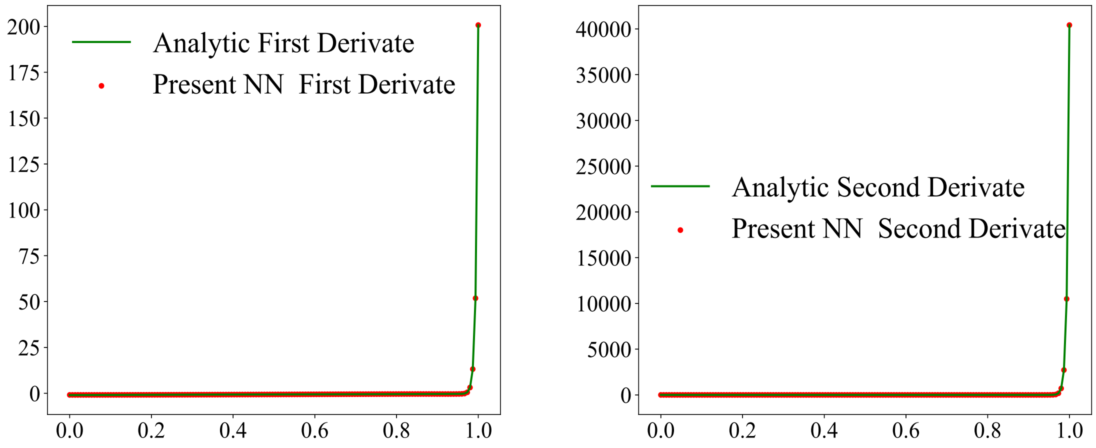

FIGURE 4 Comparison of first and second derivative between present method with analytic solution

Example 2 We consider another one-dimensional convection-diffusion equation with boundary layer at $x = 1$,

$$\begin{cases} -\varepsilon \frac{d^2u}{dx^2} + \frac{du}{dx} = \varepsilon\pi^2 \sin(\pi x) + \pi \cos(\pi x), \quad x \in [0, 1], \\ u(0) = 0, \quad u(1) = 1. \end{cases}$$

The analytic solution is

$$u(x) = \frac{e^{x/\varepsilon} - 1}{e^{1/\varepsilon} - 1} + \sin(\pi x).$$

The stretching parameters λ and loss function are the same as in Example 1.

Fig. 6 gives the analytic solution and the numerical solutions by direct NN method and the present method. Form Fig. 6 (a), we can see that the numerical solution of the direct NN method on test set is obviously violated from the analytical solution. On the contrast, the numerical solution of the present NN method is able to approximate to the analytical solution, as in Fig. 6 (b). Fig. 7 displays the first and second derivatives of the present neural network with those by analytic solution. It can be seen that the present NN method can not only approximate to the solution of the problem, but also effectively approximate to its first and second derivatives, respectively.

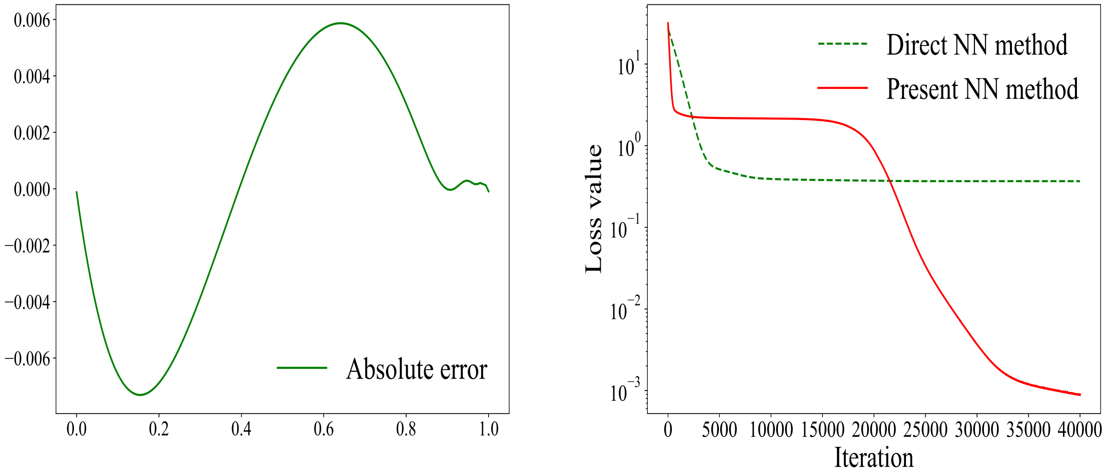

FIGURE 5 Absolute error (a) on test set and the convergence of loss function (b)

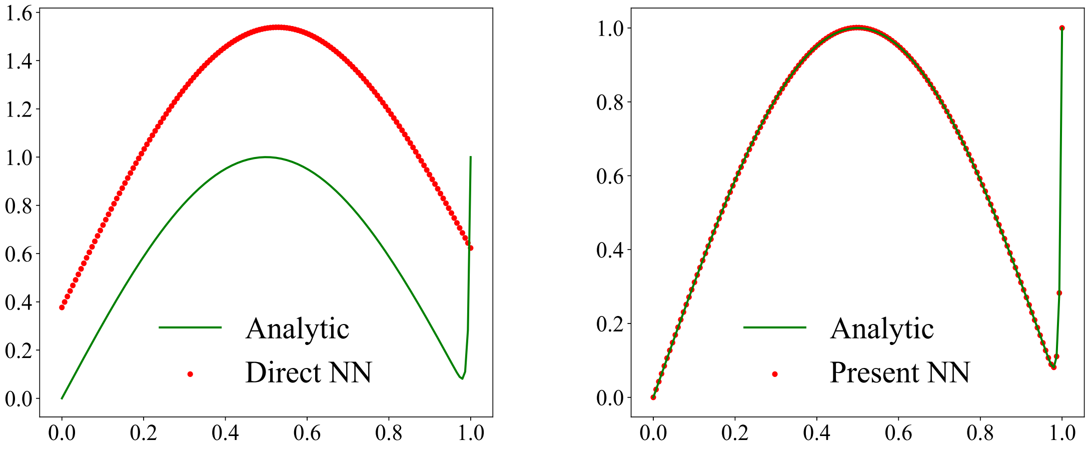

FIGURE 6 Comparison of numerical and analytical solution by direct NN method (a) and present method (b)

Fig. 8 (a) shows the absolute error between the solution by present neural network and the analytic solution. The average error on test set is about 1×10^{-3}. Fig. 8 (b) plots the convergence error of loss function by the direct NN solution and the present method during the training process. It is obvious that the error of the direct NN method cannot be reduced as the number of iteration steps increases. Conversely, the error of the presented method can decrease with the number of iteration step increase.

Table 2 displays the L_2 norm, average and variance of weight on three layers. It is shown that the L_2 norm and variance of the weights of the first hidden layer are very different from those without mapping. Meanwhile, the L_2 norm, average and variances of weight on the second and third hidden layers have little difference between present method and the direct neural network.

Example 3 We consider a two dimensional convection-diffusion equation with boundary layer at $y = 1$,

$$\begin{cases} -(u_{xx} + u_{yy}) + d(x,y)u_x + c(x,y)u_y = f(x,y), (x,y) \in \Omega, \\ u(x,y) = g(x,y), \qquad (x,y) \in \partial\Omega. \end{cases}$$

where

$$c(x,y) = (y-0.5)(y-1)(1-2x), \quad d(x,y) = 2x(x-1)(1-y).$$

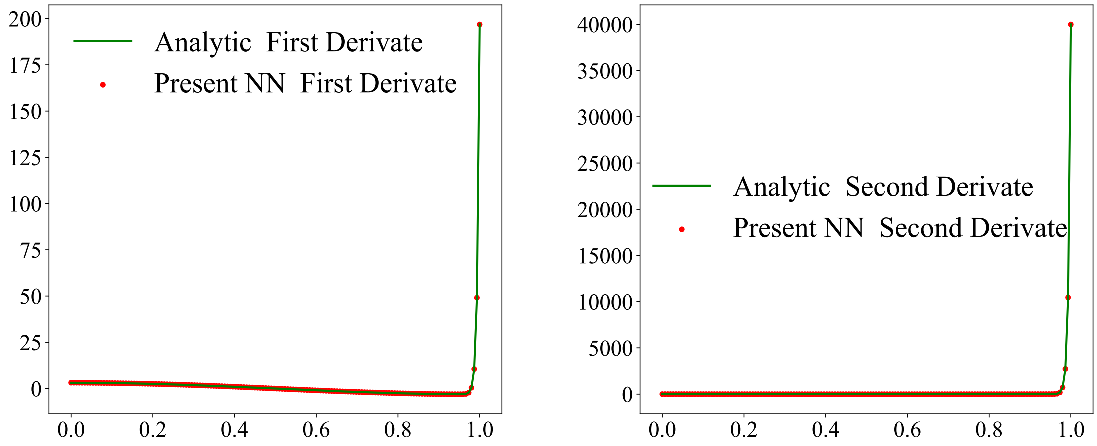

FIGURE 7 Comparison of first (a) and second (a) derivatives between present method and analytic solution

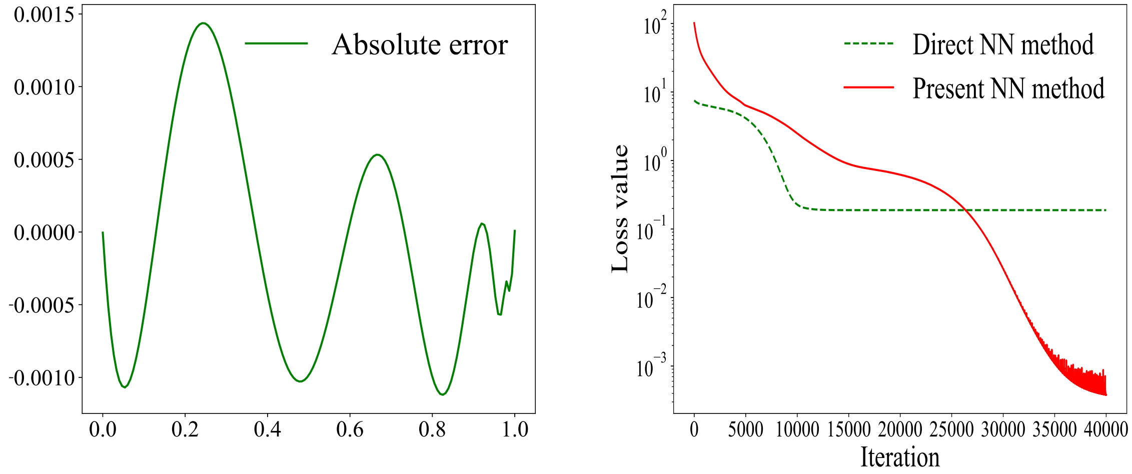

FIGURE 8 Absolute error (a) on test set and the convergence of loss function (b)

The $f(x, y)$ is imposed to satisfy the analytic solution

$$u(x) = e^{-\varepsilon(1-y)^2-x^2}.$$

The loss function of this problem is

$$\begin{aligned}\mathcal{L}(\vec{\theta}) = {} & \omega_1 \sum_{i=1}^{N_{int}} \Big(-\big(\frac{\partial^2 N(x,y,\theta)}{\partial x^2} + \frac{\partial^2 N(x,y,\theta)}{\partial y^2}\big) \\ & + d(x,y)\frac{\partial N(x,y,\theta)}{\partial x} + c(x,y)\frac{\partial N(x,y,\theta)}{\partial x} - f(x,y)\Big)^2 \\ & + \omega_2 \sum_{x=1}^{N_{bnd}} \Big(N(x,y,\theta) - g(x,y)\Big)^2,\end{aligned}$$

where $\omega_1 = 1$,$\omega_2 = 500$ and $\varepsilon = 500$.

TABLE 2 The comparison of L_2 norm, average and variance of weights in three layers for Example 2

Weight	W^1			W^2			W^3		
	L_2	Aver	Var	L_2	Aver	Var	L_2	Aver	Var
Direct NN	5.72	0.02	1.81	10.4	0.03	1.38	4.47	-0.23	1.67
Present NN	332.4	-49.9	105.0	10.3	0.24	1.56	5.33	0.47	1.34

TABLE 3 The comparison of L_2 norm, average and variance of weights in three layers for Example 3

Weight	W^1			W^2			W^3		
	L_2	Aver	Var	L_2	Aver	Var	L_2	Aver	Var
Direct NN	8.81	-0.27	2.21	14.02	-0.02	2.13	8.84	0.7	2.7
Present NN	229.3	-7.5	54.6	7.29	-0.07	1.18	3.85	0.29	1.17

Fig. 9 displays the surface of analytic solution (a), the solution by direct NN method (b), the solution by present method (c) and the error contour (d). It can be seen that there is a big difference between the analytical solution with the numerical solution by direct NN method. Meanwhile, the numerical results of the present method is very close to the analytical solution, which illustrates that the present method can approximate to the solution of the two-dimensional boundary layer problem with large gradient. The average error of the present method is about to 1.0×10^{-2} and the error contour is shown in Fig. 9 (d). Fig. 10 shows the convergence error of loss function by the direct NN solution and the present method during the training process. It can be seen that the latter converges significantly as the number of iteration steps increases.

From Table 3 , it is clear that the L_2 norm and variance of the weights of the first hidden layer by present method are larger than those without mapping. Meanwhile, the L_2 norm, average and variances of weight on the second and third hidden layers have little difference between mapping and non-mapping.

Example 4 We consider another two dimensional convection-diffusion equation with boundary layer at $y = 1$,

$$\begin{cases} -\varepsilon(u_{xx} + u_{yy}) + \frac{1}{1+y}u_y = f(x, y), & (x, y) \in \Omega, \\ u(x, y) = g(x, y), & (x, y) \in \partial\Omega. \end{cases}$$

The analytic solution is

$$u(x) = e^{y-x} + 2^{-1/\varepsilon}(1 + y)^{1+1/\varepsilon}.$$

The loss function of this problem is as follows

$$\mathcal{L}(\vec{\theta}) = \omega_1 \sum_{i=1}^{N_{int}} \Big(- \varepsilon(\frac{\partial^2 N(x, y, \theta)}{\partial x^2} + \frac{\partial^2 N(x, y, \theta)}{\partial y^2}) + \frac{1}{1+y}\frac{\partial N(x, y, \theta)}{\partial y} - f(x, y)\Big)^2 + \omega_2 \sum_{x=1}^{N_{bnd}} \Big(N(x, y, \theta) - g(x, y)\Big)^2,$$

where $\omega_1 = 1$,$\omega_2 = 500$ and $\varepsilon = 0.005$.

Fig. 11 , displays the surface of analytic solution (a), the solution by direct NN method (b), the solution by present method (c) and the error contour (d). It can be seen that the direct NN method can not approximate to the analytical solution and the error is obvious. By contrast, the numerical solution of the present method agree well with the analytical solution, which indicates that the present method is able to approximate to solutions of problems with sharp gradient. The average error of the present method is about to 2.0×10^{-2} and the error contour is shown in Fig. 11 (d). Fig. 12 shows the convergence error of loss function by the direct NN solution and the present method during the training process. It can be seen that the latter converges significantly as the number of iteration steps increases. It is concluded that the present method effectively improves the performance of the neural network, so that it can converge and approximate to the solution of the convection-diffusion problems with boundary layer and sharp gradient. Table 4 displays the L_2 norm, average and variance of weight on three layers. It's not hard to see that

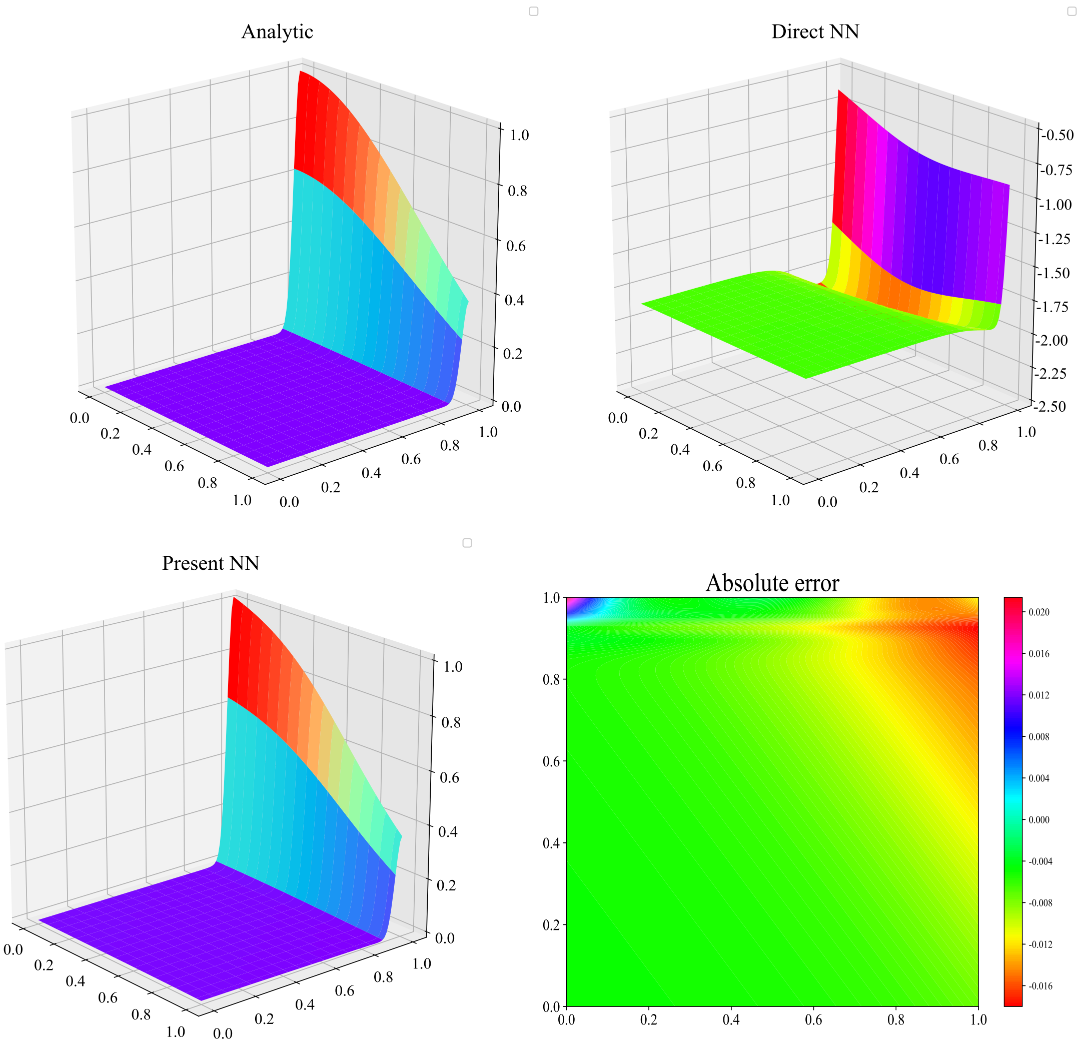

FIGURE 9 The surface of analytic solution (a), direct NN solution (b), present method (c) and error contour (d)

TABLE 4 The comparison of L_2 norm, average and variance of weights in three layers for Example 4

Weight	W^1			W^2			W^3		
	L_2	Aver	Var	L_2	Aver	Var	L_2	Aver	Var
Direct NN	12.9	0.58	3.24	10.4	0.18	3.51	30.1	0.01	3.23
Present NN	239.0	5.86	53.56	9.97	0.67	1.38	6.89	1.23	18.0

the mapping strategy by present method can inevitably affect the distribution of the weight coefficient of the first hidden layer, such that the neural network can better approximate complex functions and improve the convergence.

5 | CONCLUSION

The artificial neural networks for solving the differential equation with boundary layer is considered. This kind of problems is featured by the gradient of the solution changes drastically near the boundary layer, which poses a huge challenge for both

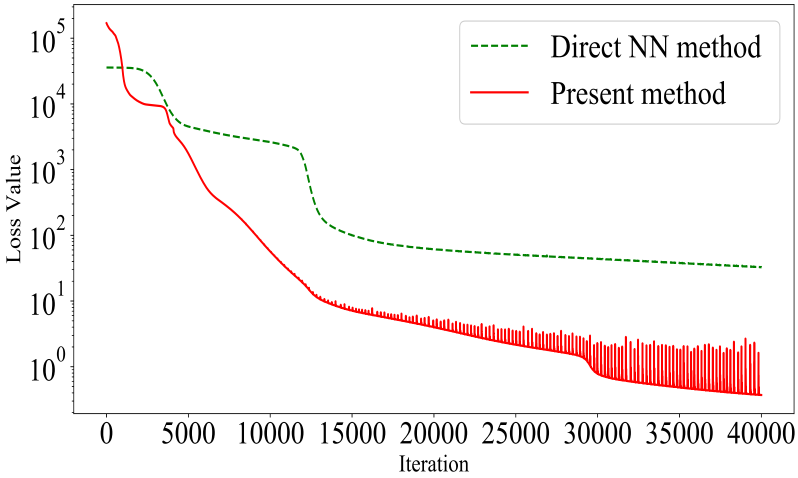

FIGURE 10 The convergence of loss function by direct NN and present method

traditional numerical methods and artificial neural network methods. By theoretically analyzing the changing rate of the weights of first hidden layer near the boundary layer, a mapping strategy and the initialization strategy coupling with the nonuniform sampling points of train set are proposed to improve the convergence of the traditional neural network. Numerical examples are carried out for the 1D and 2D convection-diffusion equations with boundary layers. The results demonstrate that the modified neural network method significantly improve the ability in approximating the solutions with sharp gradient.

6 | ACKNOWLEDGE

This work is partially supported by the National Natural Science Foundation of China (51961031,61763039),NSAF(U1630249),the National Natural Science Youth Fund Project(11801287), inner Mongolia Autonomous Region "Youth Science and technology talents" support program (NJYT20B15), the Inner Mongolia Scientific Fund Project (NJZZ18140, 2018BS01002, 2018MS06002, 2018LH01008, 2019BS01007,2020MS06010), Innovation fund project of Inner Mongolia University of science and technology-Excellent Youth Science Fund Project(2019YQL02).

References

1. Sirignano J, Spiliopoulos K. DGM:a deep learning algorithm for solving partial differential equations. *J Comput Phys*, 2018, 375: 1339-1364.
2. Hornik K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 1991, 4(2): 251-257.
3. Piscopo M L , Spannowsky M , Waite P. Solving differential equations with neural networks: applications to the calculation of cosmological phase transitions. *Physical Review D*, 2019, 100(1): doi:10.1103/physrevd.100.016002
4. Lagaris I E, Likas A, and Fotiadis D I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. On Neural Networks*, 1998, 9(5): 987-1000.
5. He S, Reif K, Unbehauen R. Multilayer neural networks for solving a class of partial differential equations. *Neural Networks*, 2000, 13(3): 385-396.
6. Weinan E, Yu B. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 2018, 6(1): 1-12.

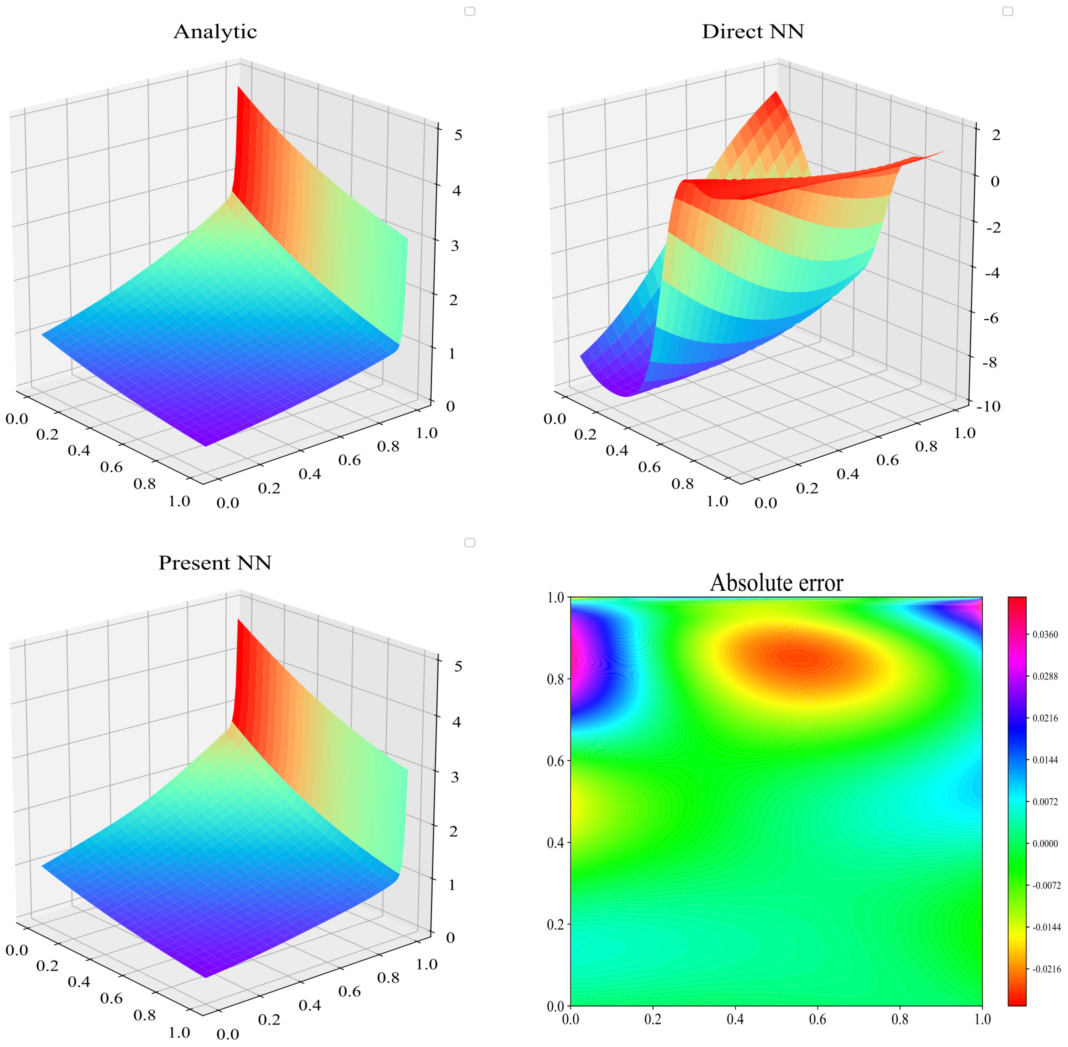

FIGURE 11 The surface of analytic solution (a), direct NN solution (b), present method (c) and error contour (d)

7. Wang Z, Zhang Z. A mesh-free method for interface problems using the deep learning approach. *J Comput Phys*, 2020, 400: doi:10.1016/j.jcp.2019.108963

8. Hayati M,Karami B. Feedforward neural network for solving partial differential equations. *J Appl Sci*, 2007, 7(19): 2812-2817.

9. Berg J, Nystrom K, A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 2018, doi:10.1016/j.neucom.2018.06.056

10. Rizaner F B , Rizaner A. Approximate solutions of initial value problems for ordinary differential equations using radial basis function networks. *Neural Process Lett*, 2018, 48(2): 1063-1071.

11. Pao Y H ,Philips S M. The functional link net and learning optimal control. *Neurocomputing*, 1995, 9: 149-164.

12. Patra J C, Kot A C, Nonlinear dynamic system identification using chebyshev functional link artificial neural network. *IEEE Trans. Syst. Man Cybern. Part B-Cybern*, 2002, 32(4): 505-511.

13. Mall S, Chakraverty S. Single layer Chebyshev neural network model for solving elliptic partial differential equations. *Neural Process Lett*, 2017, 45(3): 825-840.

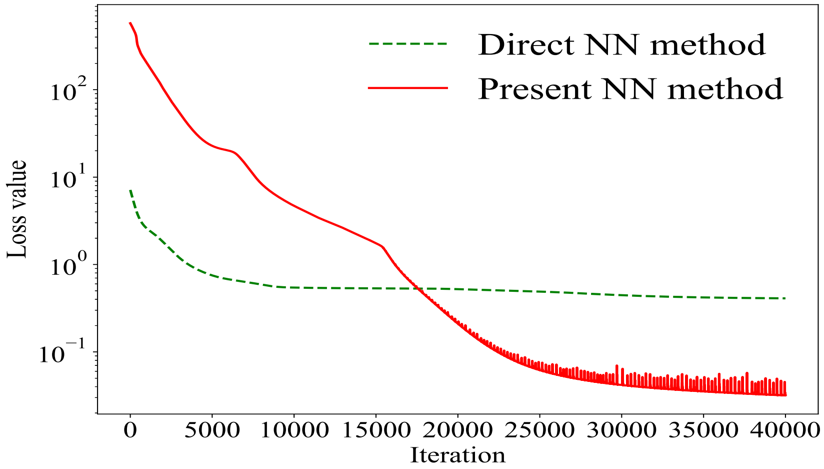

FIGURE 12 The convergence of loss function by direct NN and present method

14. Mall S, Chakraverty S. Application of Legendre neural network for solving ordinary differential equations. *Applied Soft Comput*, 2016, 43: 347-356.

15. Han J, Nica M, Stinchcombe A R. A derivative-free method for solving elliptic partial differential equations with deep neural networks. 2020,https://arxiv.org/abs/2001.06145

16. Karumuri S , Tripathy R, Bilionis I, Panchal J. Simulator-free solution of high dimensional stochastic elliptic partial differential equations using deep neural networks. *J. Comput. Phys.*, 2020, 404: 109-120.

17. Raissi M. Forward-backward stochastic neural networks: deep learning of high-dimensional partial differential equations. 2018,https://arxiv.org/abs/1804.07010

18. Weinan E, Han J, Jentzen A. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations.*Communi Math Stat*, 2017, 5: 349-380.

19. Raissi M,Perdikaris P, Karniadakis G E. Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations. 2017.

20. Raissi M,Perdikaris P, Karniadakis G E. Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations. 2017.

21. Dwivedi V, Srinivasan B, Physics informed extreme learning machine (piwelm)-a rapid method for the numerical solution of partial differential equations. *Neurocomputing*, 2020, 391: https://arxiv.org/abs/1907.03507

22. Sitzmann V, Martel J N P, Bergman A W, Lindell D B, Wetzstein G. Implicit neural representations with periodic activation functions. 2020,https://arxiv.org/abs/2006.09661