ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/327451159

Solving Partial Differential Equation Based on Bernstein Neural Network and Extreme Learning Machine Algorithm

Article *in* Neural Processing Letters · October 2019

DOI: 10.1007/s11063-018-9911-8

CITATIONS
51

READS
1,609

6 authors, including:

Muzhou Hou
Central South University
49 PUBLICATIONS **542** CITATIONS

SEE PROFILE

Tianle Zhang
University of Exeter
8 PUBLICATIONS **130** CITATIONS

SEE PROFILE

Futian Weng
Central South University
12 PUBLICATIONS **158** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

National Natural Science Foundation of China View project

All content following this page was uploaded by Futian Weng on 29 March 2020.

The user has requested enhancement of the downloaded file.

Neural Processing Letters
https://doi.org/10.1007/s11063-018-9911-8

Solving Partial Differential Equation Based on Bernstein Neural Network and Extreme Learning Machine Algorithm

Hongli Sun[1] · Muzhou Hou[1] · Yunlei Yang[1] · Tianle Zhang[1] · Futian Weng[1] · Feng Han[2]

© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract
In this paper, we introduce a new method based on Bernstein Neural Network model (BeNN) and extreme learning machine algorithm to solve the differential equation. In the proposed method, we develop a single-layer functional link BeNN, the hidden layer is eliminated by expanding the input pattern by Bernstein polynomials. The network parameters are obtained by solving a system of linear equations using the extreme learning machine algorithm. Finally, the numerical experiment is carried out by MATLAB, results obtained are compared with the existing method, which proves the feasibility and superiority of the proposed method.

Keywords Bernstein neural network · Extreme learning machine · Differential equation

1 Introduction

Many problems in the field of science, such as physics, chemistry, biology, economics and so on need to be modeled by differential equations. However, it is difficult to find the exact solutions of differential equations in practical applications, so numerical methods for solving differential equations have important theoretical significance and practical application value. Many numerical solutions are proposed in the literature, such as the Runge–Kutta method [1, 2], Predictor correction method [3], Finite difference method [4, 5], Finite Element Method [6, 7], B-spline [8, 9] and other methods [10–21]. Although these methods are widely applied and powerful, they usually need to be discretized through a grid and then iterated, which is difficult in high-dimensional problems and computationally expensive.

Neural networks have become an alternative to the numerical solution of differential equations, because they avoid some of the drawbacks of traditional numerical techniques. For example, domain discretization usually involves a simple square grid, and does not require special processing for nonlinear differential equations. In 1998, Issac Ellias Lagaris and Aristidis Likas proposed a method that using the artificial neural network to solve the initial

✉ Muzhou Hou
houmuzhou@sina.com

1 School of Mathematics and Statistics, Central South University, Changsha 410083, China

2 College of Science, Hunan University of Technology, Hunan, China

Published online: 05 September 2018

boundary value problem. They chose the form of the test function to satisfy the given boundary condition by constructing trial solution [22]. In 2000, Lagaris and Likas [23] proposed a method to deal with irregular borders (Dirichlet or Neumann boundary conditions), based on multi-layer perceptron (MLP) and radial basis function (RBF) network. In 2001, Mai-Duy and Tran-Cong [24] proposed the use of radial basis function neural network to obtain the numerical solution of differential equations, Aarts and Veer [25] use the evolutionary algorithm to train the neural network, and then the numerical solution of the partial differential equation is obtained. In 2002, Mai-Duy and Tran-Cong [26] proposed direct radial basis network (DRBFN) and indirect neural network (IRBFN) for solving differential equation. In 2003, Li et al. [27] proposed a two-stage gradient-down learning strategy to train radial-based neural networks. In 2006, Malek and Beidokhti [28] proposed a hybrid neural network based on the optimization method to solve the high order ordinary differential equation. In 2007, Xu et al. [29] used the trigonometric function as the basic function of the neural network to solve the ordinary differential equation. In 2009, Aein and Talebi [30] obtained the numerical solution by training the convolution neural network based on the online learning strategy. Beidokhti and Malek [31] proposed a hybrid method based on artificial neural networks, minimization techniques and collocation methods, and calculated the optimal value of the corresponding adjustable parameter to determine the approximate solution in a closed analytical form. In 2012, Xu and Wang [32] solved the differential equation based on the finite element neural network, Mcfall [33] solved coupled partial differential equations with discontinuities using automatic design parameters selection of neural networks. In 2017, Zuniga-Aguilar et al. [34] proposed using artificial neural networks to solve fractional differential equations of variable-order involving operators with Mittag–Leffler kernel.

Pao and Philips [35] introduced a single-layer functional link artificial neural network (FLANN) model in 1995, functional neural networks based on orthogonal polynomials have been widely used, such as function approximation [36, 37], digital communication [38], channel equalization [39], nonlinear dynamic system identification [40, 41], etc. Mall and Chakraverty [42] applied the Legendre neural network to the ordinary differential equation in 2016, the hidden layer in the neural network was replaced by a functional extension block, using the Legendre orthogonal polynomial to extend the input pattern, they also solved elliptic neural networks using a single Chebyshev neural network in 2017 [43].

In this paper, we propose a new method to solve the partial differential equation, based on single-layer Bernstein neural network, and use the extreme learning machine algorithm to determine the network parameters [44], which has the advantages of less parameters and simple calculation. Finally, we verified the effectiveness of the method by numerical experiments.

The remaining of this paper is organized as follows: In Sect. 2, the Bernstein neural network model is briefly described. In Sect. 3, the general formulation for differential equations using the extreme learning machine algorithm are shown. Section 4 describes the results of numerical experiments and comparison with existing methods. Finally, Sect. 5 presents a conclusion.

2 Bernstein Neural Network Model

A single-layer Bernstein neural network (BeNN), which consists of input node x, an output layer, and a functional extension block based on the Bernstein basis function. In fact, we eliminated the hidden layer by using the Bernstein basis function, which transforms the input

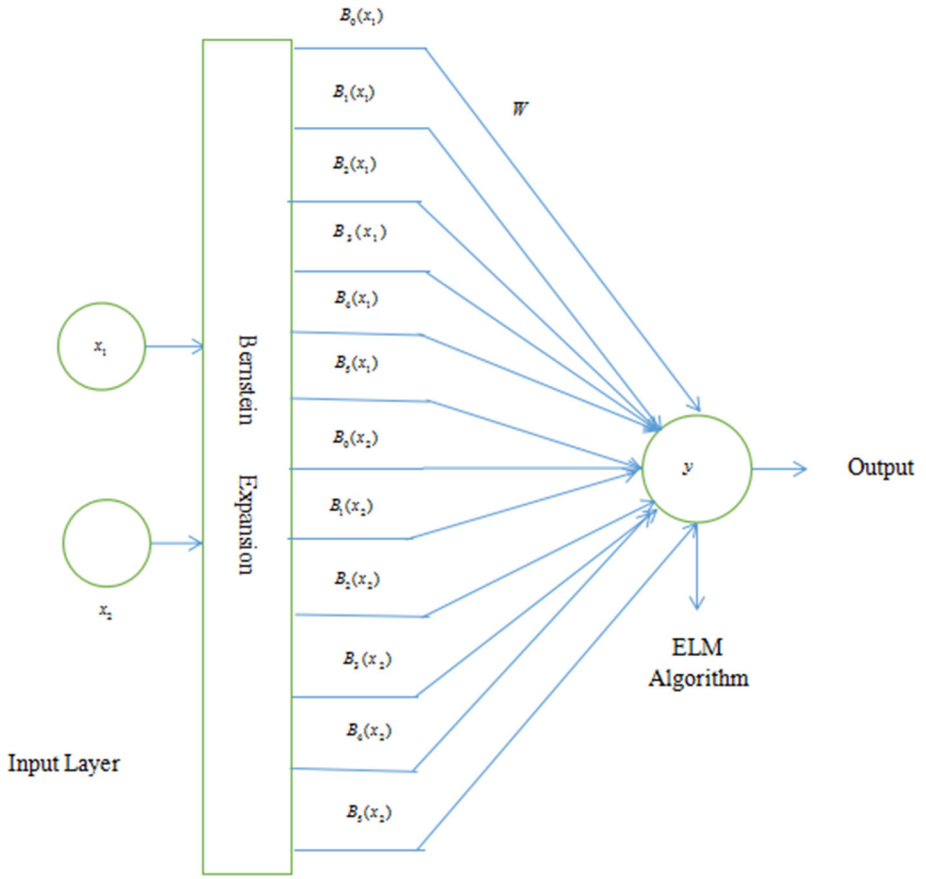


Fig. 1 Structure of a Bernstein neural network, m = 2, n = 12

pattern to a higher dimension space, where the Bernstein basis function is denoted by $B_n(x)$, x is the input data, $n = 0, 1, 2 \ldots$.

The general form of the Bernstein basis function is as follows:

$$B_{j,n}(x) = C_n^j x^j (1-x)^{n-j}, \quad j = 0, 1, \ldots n \tag{1}$$

The Bernstein basis function has the symmetric and recursive formulas as follows:

$$\begin{aligned} B_{j,n}(x) &= B_{n-j,n}(1-x) \\ B_{j,n}(x) &= (1-x)B_{j,n-1}(x) + x B_{j-1,n-1}(x) \end{aligned} \tag{2}$$

Consider an m-dimensional input vector $x = (x_1, x_2, x_3, \ldots x_m)$, the enhanced pattern is obtained by using Bernstein polynomials as follows:

$$[B_0(x_1),\ B_1(x_1), \ldots B_n(x_1);\ B_0(x_2),\ B_1(x_2), \ldots B_n(x_2); \ldots B_0(x_m),\ B_1(x_m), \ldots B_n(x_m)] \tag{3}$$

Here we extend the m-dimensional input vector to n-dimensional and apply it to a single-layer neural network ($n > m$). The BeNN structure is shown in Fig. 1, where m = 2 and n =

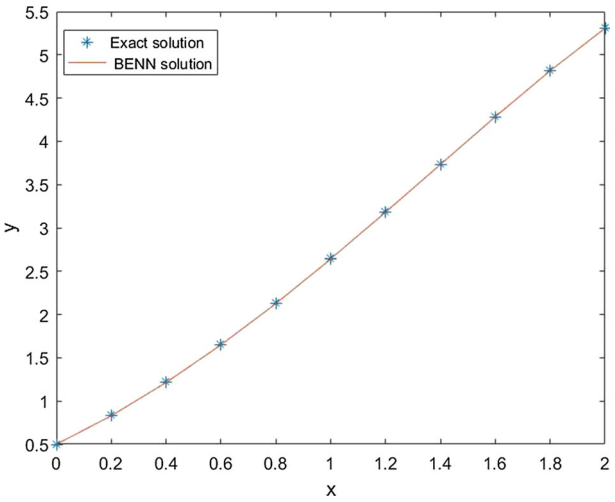


Fig. 2 Figure of Exact and BeNN solution

12 are selected. In this paper, we apply this BeNN model to obtain the solution of differential equations.

3 Neural Network Model for Differential Equation Solution with Extreme Learning Machine Algorithm

3.1 Neural Network Model for Differential Equation Solution

In this section, we introduce the form and construction of the solution of the differential equation using the neural network. Consider the general form of the ordinary differential and partial differential equations as follows:

$$G(x, y(x), \nabla y(x), \nabla^2 y(x), \ldots, \nabla^k y(x)) = 0, \quad x \in D \subseteq R^n \tag{4}$$

This function is established under certain initial and boundary conditions, where $y(x)$ is the solution of the differential equation, ∇ is the differential operator, G is a function that defines the structure of the differential equation, and D is the discrete domain on the finite set of points.

Let $y_t(x,\ p)$ denote the trial solution with parameter p, which can be transformed into solving the following minimization problem:

$$\underset{p}{Min} \sum_{x_n \in D} \left(G(x_n,\ y_t(x_n,\ p),\ \nabla y_t(x_n,\ p),\ \nabla^2 y_t(x_n,\ p) \ldots \nabla^k y_t(x_n,\ p)) \right)^2$$

where the trial solution satisfies the initial condition or boundary condition, which can be written as the addition of two terms:

$$y_t(x, p) = A(x) + F(x, N(x, p)) \tag{5}$$

where $A(x)$ satisfies the initial condition or boundary condition and does not contain the parameter p, $N(x,\ p)$ is the output of the neural network with parameters p and inputs x, and the second term has no contribution to the initial or boundary condition.

3.1.1 First Order Ordinary Differential Equation

Consider the following first-order ordinary differential equation initial value problem:

$$\begin{cases} \frac{dy}{dx} = f(x, y), x \in [a, b] \\ y(a) = A \end{cases} \tag{6}$$

According to Sect. 3.1, we can construct a trial solution that satisfies the condition:

$$y_t(x, p) = A + (x - a)N(x, p) \tag{7}$$

3.1.2 Second Order Ordinary Differential Equation

Consider the following boundary value problem for the second order differential equation:

$$\begin{cases} \frac{d^2y(x)}{dx^2} = f\left(x, y, \frac{dy}{dx}\right),\ x \in [a, b], \\ y(a) = A, y(b) = B \end{cases} \tag{8}$$

The corresponding trial solution for the above boundary value problem is written as follows:

$$y_t(x, p) = \frac{bA - aB}{b - a} + \frac{B - A}{b - a}x + (x - a)(x - b)N(x, p) \tag{9}$$

Likewise, if we change the boundary condition in (8) to $y(a) = A,\ \ y'(a) = A'$, then the trial solution can be expressed as:

$$y_t(x, p) = A - A'a + A'x + (x - a)^2N(x, p) \tag{10}$$

3.1.3 Initial Value Problems for Systems of Ordinary Differential Equations

Here, we consider the initial value problems of the following systems of differential equations:

$$\begin{cases} \frac{dy_i(x)}{dt} = f_i[x, y_1(x), \ldots y_m(x)],\ t \in [t_0, t\max],\ i = 1, \ldots m \\ y_i(x_0) = A_i,\ i = 1, \ldots m \end{cases} \tag{11}$$

The corresponding trial solution can be expressed as:

$$y_t(x, p_i) = A_i + (x - x_0)N_i(x, p_i), \quad i = 1, \ldots, m \tag{12}$$

3.1.4 Two Dimensional Problems with Dirichlet Boundary Conditions

Here, we consider the following Poisson equation:

$$\frac{\partial^2 y(x_1, x_2)}{\partial x_1^2} + \frac{\partial^2 y(x_1, x_2)}{\partial x_2^2} = f(x_1, x_2) \quad x_1 \in [0, 1],\ x_2 \in [0, 1] \tag{13}$$

Subject to Dirichlet boundary conditions

$$\begin{cases} y(0,\, x_2) = f_0(x_2),\; y(1,\, x_2) = f_1(x_2) \\ y(x_1,\, 0) = g_0(x_1),\; y(x_1,\, 1) = g_1(x_1) \end{cases}$$

The trial solution is written as

$$y_t(x_1, x_2, p) = A(x_1, x_2) + x_1(1 - x_1)x_2(1 - x_2)N(x_1, x_2, p) \tag{14}$$

where $A(x_1,\, x_2)$ satisfies the boundary conditions.
Here $A(x_1,\, x_2)$ is expressed as

$$\begin{aligned} A(x_1, x_2) = {} & (1 - x_1) f_0(x_2) + x_1 f_1(x_2) + (1 - x_2)[g_0(x_1) - \{(1 - x_1)g_0(0) + x_1 g_0(1)\}] \\ & + x_2[g_1(x_1) - \{(1 - x_1)g_1(0) + x_1 g_1(1)\}] \end{aligned} \tag{15}$$

3.1.5 Two Dimensional Problems with Mixed Boundary Conditions

For this problem we consider the following equation:

$$\begin{cases} \nabla^4 y(x_1, x_2) = \left(\frac{\partial^4}{\partial x_1^4} + 2\frac{\partial^4}{\partial x_1^2 \partial x_2^2} + \frac{\partial^4}{\partial x_2^4} \right) y(x_1, x_2) = f(x_1, x_2), (x_1, x_2) \in \Omega \\ y(x_1, x_2) = B_1(x_1, x_2), (x_1, x_2) \in \partial\Omega, \\ \frac{\partial y}{\partial n}(x_1, x_2) = B_2(x_1, x_2), (x_1, x_2) \in \partial\Omega, \\ \Omega = [a_1, b_1] \times [a_2, b_2] \end{cases} \tag{16}$$

The trial solution we constructed is:

$$y_t(x_1, x_2, p) = \varphi(x_1, x_2) + \psi[x_1, x_2, N(x_1, x_2, p)] \tag{17}$$

where

$$\varphi(x_1,\, x_2) = \left(a x_1^4 + b x_1^3 + c x_1^2 + d x_1\right) + \left(a' x_2^4 + b' x_2^3 + c' x_2^2 + d' x_2\right),$$

$$\psi[x_1,\, x_2,\, N(x_1,\, x_2,\, p)] = (x_1 - a_1)^2 (x_1 - b_1)^2 (x_2 - a_2)^2 (x_2 - b_2)^2 N(x_1,\, x_2,\, p).$$

3.2 Extreme Learning Machine algorithm

Given a different set of samples $(X_i,\, t_i)$, $X_i \in R^n$, $t_i \in R$, the neural network with N hidden neurons is modeled as follows:

$$\sum_{i=1}^{N} \beta_i f(W_i X_j + b_i), \quad j \in [1, M] \tag{18}$$

where f is the activation function, W_i is the input weight of the i-th neuron in the hidden layer, b_i is the threshold, and β_i is the output weight.

When the error between the output and the actual data is zero, that is the neural network completely approximates the data, and the following relationship holds:

$$\sum_{i=1}^{N} \beta_i f(W_i X_j + b_i) = t_i, \quad j \in [1, M] \tag{19}$$

Written in matrix form:

$$H\beta = T \tag{20}$$

where H is the hidden layer output matrix, defined as follows:

$$H = \begin{bmatrix} f(W_1 X_1 + b_1) & \cdots & f(W_N X_1 + b_N) \\ \vdots & \ddots & \vdots \\ f(W_1 X_M + b_1) & \cdots & f(W_N X_M + b_N) \end{bmatrix} \tag{21}$$

and $\beta = \beta_1, \ldots, \beta_N^T$, $Y = (t_1, \ldots, t_M)^T$.

Given the W_i and the training input $X_i \in R^n$, the hidden layer output matrix H can be calculated. If the target output $t_i \in R$ is given, the output weight can be solved by the defined $H\beta = T$ linear system. This can be obtained from $\beta = H^{\dagger}T$, where $H^{\dagger}$ is the Moore–Penrose generalized inverse of matrix H. In fact, this β solution is the only least-squares solution of the equation.

3.3 BeNN Model with Extreme Learning Machine Algorithm

Following are the steps to the BeNN network:

Step 1 Initialize the input vector $x = [x_1, x_2, \ldots, x_m]$

Step 2 Make BeNN functional block of input data as
$[B_0(x_1), B_1(x_1), \ldots B_n(x_1); \quad B_0(x_2), B_1(x_2), \ldots B_n(x_2); \ldots B_0(x_m), B_1(x_m), \ldots B_n(x_m)]$

Step 3 Calculate the output y_t of the BeNN model as
$N(x, p) = \sum_{j=1}^{n} W_{ij} B_{j-1}(x), \; i = 1, \ldots m, \; y_t = A(x) + F(x, N(x, p))$
and compute $\nabla y_t(x), \nabla^2 y_t(x), \ldots, \nabla^k y_t(x), \quad k = 1, 2, \ldots h.$

Step 4 Discrete the interval $x_i \in [a, b]$, $a = x_{i1} < x_{i2} < \cdots < x_{iM} = b$, compute
$y_t(x_{ik}), \nabla y_t(x_{ik}), \nabla^2 y_t(x_{ik}), \ldots, \nabla^k y_t(x_{ik}), \; i = 1, \ldots m, \; k = 1, \ldots M$

Step 5 Substitute $y_t, \nabla y_t(x), \nabla^2 y_t(x), \ldots, \nabla^k y_t(x)$ from step 4 into $G(x, y(x), \nabla y(x), \nabla^2 y(x), \ldots, \nabla^k y(x)) = 0$, then
write it in a matrix $H\beta = T$, where $\beta = [W_{1_j}, \; W_{2_j}, \ldots, W_{i_j}, \ldots, W_{m_j}]^T$, $W_{ij} = [W_{i_1}, W_{i_2}, \ldots, W_{i_n}]^T, \; i = 1, 2, \ldots m.$

Step 6 Apply Extreme Learning Machine to train BeNN, compute $\beta = H^{\dagger}\mathrm{T}$.

Step 7 Print the output of the BeNN model

3.4 Convergence Discussion

Theorem 1 *Given a single layer Bernstein Neural Network, assume that the one-dimensional input vector x is extended to N-dimensional by Bernstein polynomial, for N arbitrary distinct* (x_i, t_i) $i = 1, 2 \ldots N$, *where* $x_i \in R$, $t_i \in R$, *for any* b_i *randomly chosen from any intervals of R, the Bernstein expansion layer output matrix H is invertible and* $||H\beta - T|| = 0$.

Proof Let us consider a vector $c(b_i) = [B_i(x_1 + b_i), B_i(x_2 + b_i), \ldots, B_i(x_{\mathrm{N}} + b_i)]^{\mathrm{T}}$, where $b_i \in (a, b)$ an (a, b) is any interval of R, the ith column of H belongs to R^N. □

Following the proof in [44], it can be easily proved by contradiction that vector c does not belong to any subspace whose dimension is less than N.

Since $x_i \neq x_j$, $i \neq j$, we assume that c belongs to a subspace of dimension N − 1, then there exists a vector α which is orthogonal to this subspace

$$(\alpha,\, c(b_i) - c(a)) = \alpha_1 . B_i(b_i + x_1) + \alpha_2 . B_i(b_i + x_2) + \cdots + \alpha_N . B_i(b_i + x_N) - \alpha . c(a) = 0$$

We assume $\alpha_N \neq 0$, then

$$B_i(b_i + x_N) = -\sum_{p=1}^{N-1} \gamma_p B_i(b_i + x_p) + z/\alpha_N,$$

where $z = \alpha . c(a)$, $\gamma_p = \alpha_p/\alpha_N$..

Since $B_i(x)$ is infinitely differentiable in any interval, we have

$$B_i^{(l)}(b_i + x_N) = -\sum_{p=1}^{N-1} \gamma_p B_i^{(l)}(b_i + x_p), \quad l = 1, 2, \ldots, N, \ N+1, \ldots$$

where $B_i^{(l)}$ is the lth derivative of function B_i of b_i. But there are only N − 1 free coefficients:

$\gamma_1, \gamma_2, \ldots, \gamma_N$ for the derived more than N − 1 linear equations, which is contradictory. Therefore, the vector c does not belong to any subspace whose dimension is less than N. In other words, for any b_i chosen from any intervals of R, according to any continuous probability distribution, the column vectors of H can be made full-rank.

Since $b_i \in R$ is arbitrary, we can make $b_i = 0$, $i = 1, 2, \ldots N$, the matrix H is still full-rank.

Obviously, given any small positive value $\varepsilon > 0$, when the one-dimensional input vector x is extended to $\tilde{\mathrm{N}} < N$ by Bernstein polynomial,for any arbitrary N samples (x_i, t_i), $i = 1, 2, \ldots N$,the matrix H satisfies $||\mathrm{H}_{N\times\tilde{N}}\beta_{\tilde{N}\times m} - \mathrm{T}_{N\times m}|| < \varepsilon$.

Similarly, when the input vector is M dimension, the theorem 1 is still valid.

4 Numerical Results

In this section, we consider various types of differential equations to show the powerfulness of the proposed method. The error appearing in this paper is defined as follows:

$$\text{BeNN Error} = |\text{Exact solution} - \text{BeNN solution}|.$$

Example 1 Given the following ordinary differential equation [29]:

$$\begin{cases} \frac{dy}{dx} = y - x^2 + 1, x \in [0, 2] \\ y(0) = 0.5 \end{cases} \tag{22}$$

According to (6, 7), the trial solution of the equation is:

$$y_t(x) = 0.5 + xN(x, p) \tag{23}$$

Ten equidistant points in [0, 2] and six weights with respect to first six Bernstein polynomials are considered. Comparison between exact solution and BeNN solution has been shown in Table 1. These comparisons are also depicted in Fig. 2. The error of the BeNN method and the MLP (multi-layer perceptron) method are shown in Fig. 3. The maximum absolute error of MLP is 1.9×10^{-2} in [29], and the maximum absolute error of BeNN is 2.7×10^{-3}, which fully demonstrates the superiority of the proposed method.

Table 1 Comparison between exact solution and BeNN solution (Example 1)

x	Exact solution	BeNN solution	BeNN error	Error in [29]
0	0.500000000000000	0.500000000000000	0	0
0.2	0.829298620919915	0.829298545243513	7.7e−8	7e−4
0.4	1.214087651179365	1.214087632466867	1.7e−8	4e−4
0.6	1.648940599804746	1.648940492519046	1.7e−7	2.3e−3
0.8	2.127229535753767	2.127229479459782	5.3e−8	1.5e−3
1.0	2.640859085770477	2.640858940680662	1.5e−7	1.8e−3
1.2	3.179941538631726	3.179944740068502	3.0e−6	4.4e−3
1.4	3.732400016577662	3.732440434210966	4.4e−5	3.5e−3
1.6	4.283483787802443	4.283710101644466	2.6e−4	5.5e−3
1.8	4.815176267793527	4.816047825144281	8.2e−4	4.3e−3
2.0	5.305471950534675	5.308143827056824	2.7e−3	1.9e−2

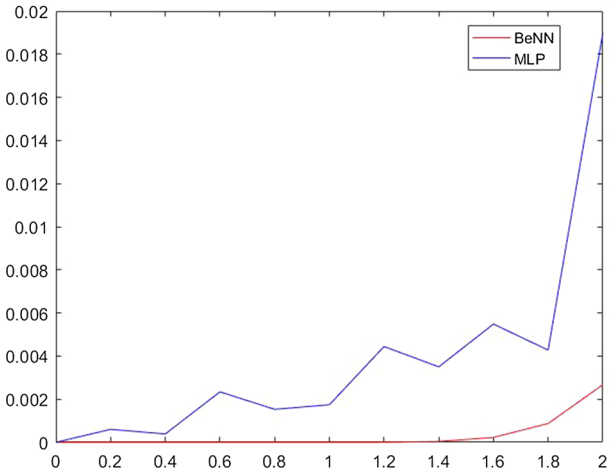


Fig. 3 Error comparison between MLP in [29] and BENN

Example 2 Given the following 2-order ordinary differential equation [21]:

$$\begin{cases} \frac{d^2}{dx^2}y(x) + y(x) = 2 \\ y(0) = 1, y(1) = 0 \end{cases} \quad x \in [0, 1] \tag{24}$$

According to (9, 10), the trial solution of the equation is:

$$y_t(x) = 1 - x + x(1 - x)N(x, p) \tag{25}$$

Here we consider ten equidistant points in [0, 1] and the results are compared between exact solution and BeNN solution. The exact solution and the obtained solution via our proposed method are displayed in Table 2 and Fig. 4. The obtained absolute error for points in the domain [0, 1] are recorded in Table 2, which shows the superiority of the proposed method over the described method in [21]. Note that in [21], 50 equidistant points are used for training and the maximum absolute error in [21] is approximately 5×10^{-1}, but the maximum absolute error via our proposed method is 7.3×10^{-9}. It is clear that the solution is of higher accuracy compared in [21], despite the fact that fewer training points are used.

Table 2 Comparison between exact and BeNN solution (Example 2)

x	Exact solution	BeNN solution	BeNN error
0.0	1.000000000000000	1.000000000000000	0
0.1	0.831815046902001	0.831815050021417	3.1e−9
0.2	0.675302211704751	0.675302208312442	3.4-9
0.3	0.532025318921278	0.532025314579426	4.3e−9
0.4	0.403415943903228	0.403415946636496	2.7e−9
0.5	0.290759109013176	0.290759116358856	7.3e−9
0.6	0.195180444105945	0.195180447030709	2.9e−9
0.7	0.117634939607180	0.117634935314112	4.3e−9
0.8	0.058897404564699	0.058897401065029	3.5e−9
0.9	0.019554725012598	0.019554728222903	3.2e−9
1.0	0	0	0

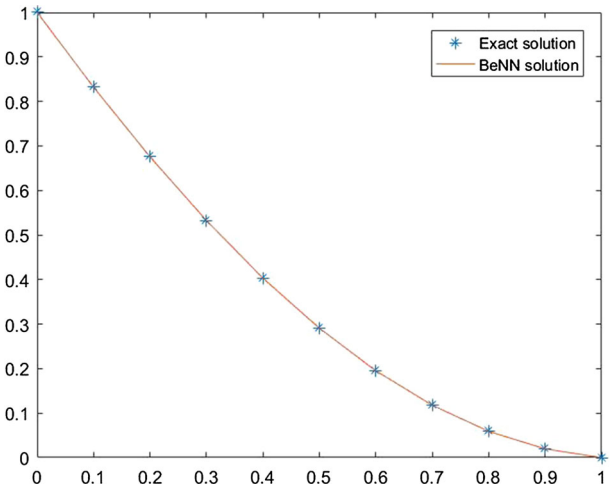


Fig. 4 Figure of Exact and BeNN solution

Example 3 We consider the following systems of differential equations [10]:

$$\begin{cases} \frac{dy_1}{dx} = \cos(x), y_1(0) = 0 \\ \frac{dy_2}{dx} = -y_1, y_2(0) = 1 \\ \frac{dy_3}{dx} = y_2, y_3(0) = 0 \qquad x \in [0,1] \\ \frac{dy_4}{dx} = -y_3, y_4(0) = 1 \\ \frac{dy_5}{dx} = y_4, y_5(0) = 0 \end{cases} \tag{26}$$

The related BeNN trial solution is written as:

$$\begin{aligned} y_i(x,p) &= xN(x,p), \quad i = 1,3,5 \\ y_j(x,p) &= 1 + xN(x,p), \quad j = 2,4 \end{aligned} \tag{27}$$

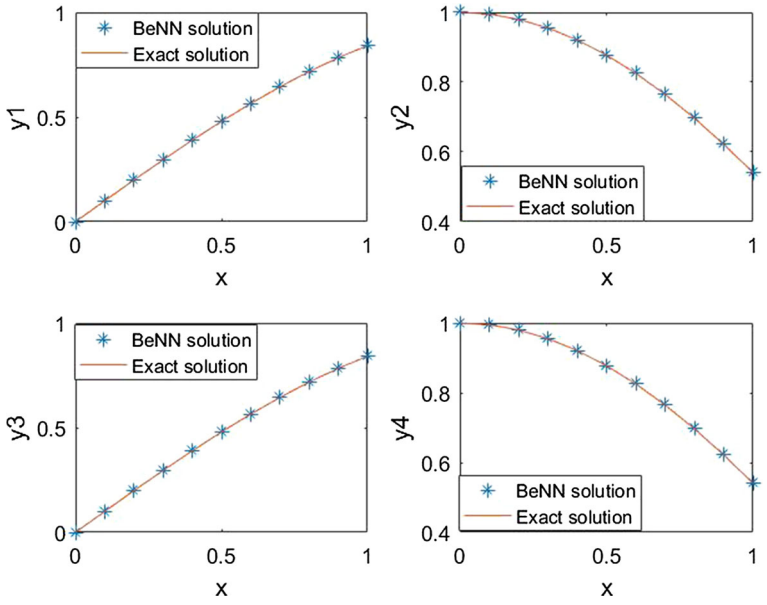


Fig. 5 Figure of Exact and BeNN solution

Ten equidistant points in the given interval are used for the training process, and the obtained results are shown in Fig. 5. The proposed method shows a better performance in comparison with the described method in [10] in terms of accuracy, the obtained maximum absolute error is 6.8×10^{-8}, which is smaller than 2.1×10^{-5} shown in [10].

Example 4 We consider the following two dimensional elliptic partial differential equation [43]:

$$\begin{cases} \frac{\partial^2 y}{\partial x_1^2} + \frac{\partial^2 y}{\partial x_2^2} = \sin(\pi x_1)\sin(\pi x_2) \\ y(0, x_2) = 0, y(1, x_2) = 0, \qquad x_1, x_2 \in [0, 1] \\ y(x_1, 0) = 0, y(x_1, 1) = 0 \end{cases} \tag{28}$$

Following the procedure of the present method, we write the BeNN trial solution:

$$y_t(x_1, x_2, p) = x_1(1 - x_1)x_2(1 - x_2)N(x, p) \tag{29}$$

In order to have a fair comparison with the results reported in [43], we train the network for 100 equidistant points in the given domain [0, 1] × [0, 1] with the first six Bernstein polynomials. The exact solution and BeNN solution are cited in Figs. 6 and 7. Plot of the error function between exact solution and BeNN solution is cited in Fig. 8. Table 3 incorporates corresponding results for some test points, this test verifies whether the converged BeNN can give a direct result by using points that are not trained in the training process. We selected the same training point as in [43], the maximum absolute error in [43] is 7×10^{-4}, in our proposed method is 7.6×10^{-5}, which proves the effectiveness of our proposed method. The proposed method is also superior to the method in [43] in calculating speed, the time cost in [43] is 3.49s, and the time in our method is only 0.38s, which again shows the superiority of the proposed method.

Fig. 6 Figure of exact solution

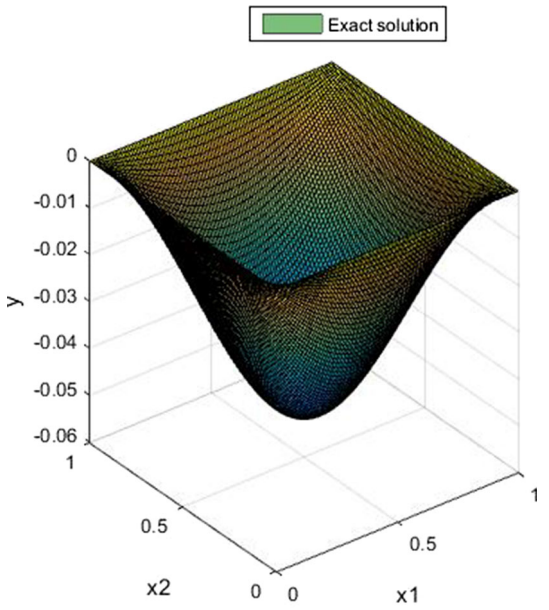
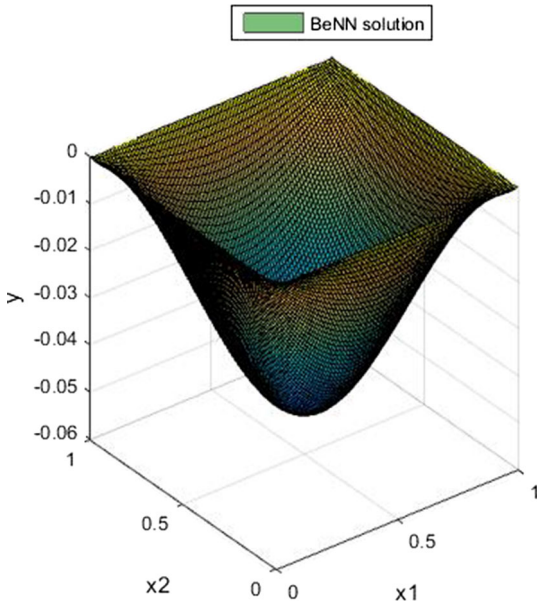


Fig. 7 Figure of BeNN solution



Example 5 We consider the following partial differential equation with Dirichlet boundary conditions [43]:

$$\begin{cases} \frac{\partial^2 y}{\partial x_1^2} + \frac{\partial^2 y}{\partial x_2^2} = e^{-x_1}\left(x_1 - 2 + x_2^3 + 6x_2\right) \\ y(0, x_2) = x_2^3, y(1, x_2) = (1 + x_2^3)e^{-1}, \\ y(x_1, 0) = x_1 e^{-x_1}, y(x_1, 1) = (x_1 + 1)e^{-x_1} \end{cases} \quad x_1, x_2 \in [0, 1] \tag{30}$$

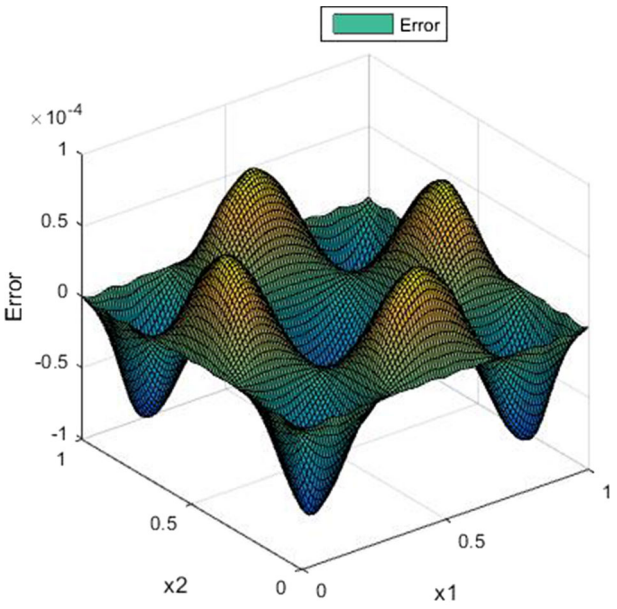


Fig. 8 Plot of error between exact solution and BeNN solution

Table 3 Comparison between exact and BeNN results for test points (Example 4)

x_1	x_2	Exact solution	BeNN solution	BeNN error	Chebyshev error [43]
0	0.8910	0	0	0	0
0.1710	0.0770	−0.006210390112558	−0.006147679107526	6.3e−5	1.0e−4
0.3900	0.1940	−0.027285307324177	−0.027315231616783	3.0e−5	0
0.4700	0.2840	−0.039262336504184	−0.039259973225224	2.4e−6	2.0e−4
0.8250	0.5390	−0.026271654853940	−0.026323476610311	5.2e−5	7.0e−4
0.3400	0.6820	−0.037333074613064	−0.037321217215945	1.2e−5	1.0e−4
0.7410	0.5680	−0.035983843931486	−0.035994576102912	1.1e−5	1.0e−4
0.9500	0.3940	−0.007489684013004	−0.007518922022177	2.9e−5	4.0e−4
0.1530	0.8820	−0.008485874395444	−0.008410353206707	7.6e−5	1.0e−4
0.9370	0.4720	−0.004275437645072	−0.004302227392013	2.7e−5	1.0e−4

Following (16, 17) the trial neural form must be written:

$$y_t(x_1, x_2, p) = A(x_1, x_2) + x_1(1 - x_1)x_2(1 - x_2)N(x_1, x_2, p) \tag{31}$$

where $A(x_1, x_2) = (1 - x_1)x_2^3 + x_1(1 + x_2^3)e^{-1} + (1 - x_2)x_1(e^{-x_1} - e^{-1}) + x_2[(1 + x_1)e^{-x_1} - (1 - x_1 - 2x_1e^{-1})]$

We train the network for 100 equidistant points in the given domain $[0, 1] \times [0, 1]$. The exact solution and BeNN solution are shown in Figs. 9 and 10. Figure 11 presents the error between the BeNN solution and exact solution at 100 points that were selected for training. From the obtained results, it is clear that our method outperforms the method in [43] in terms of accuracy (see [43, Fig. 10]), the accuracy of the error is $O(10^{-3})$ in [43], the accuracy

Fig. 9 Figure of exact solution

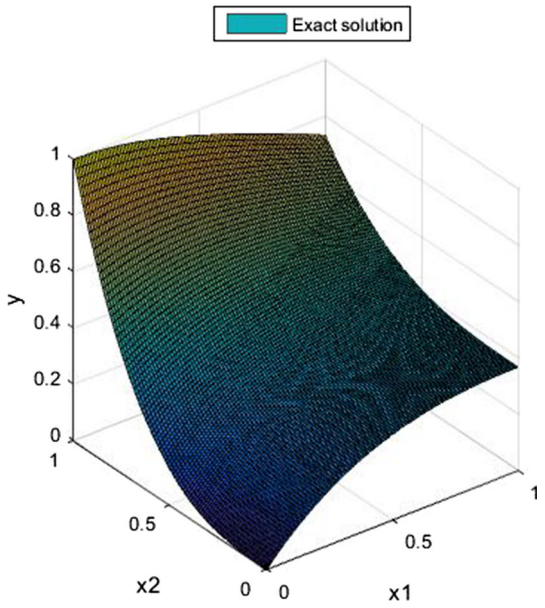
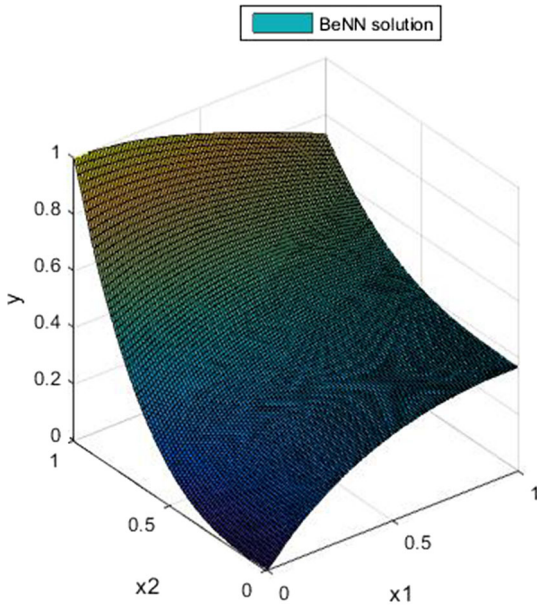


Fig. 10 Figure of BENN solution



of the error obtained by our method is $O(10^{-4})$. Table 4 shows the BeNN solution of the corresponding test points directly by using the convergent weight, and the obtained maximum absolute error in testing points is 2.4×10^{-4}, which shows that the error remains low at the test point. In terms of calculation speed, the computational time consumption in [43] is 2.10s, and the time of our proposed method is only 0.12s. So, no matter in calculating accuracy or calculating speed, our proposed method is more powerful than the method in [43].

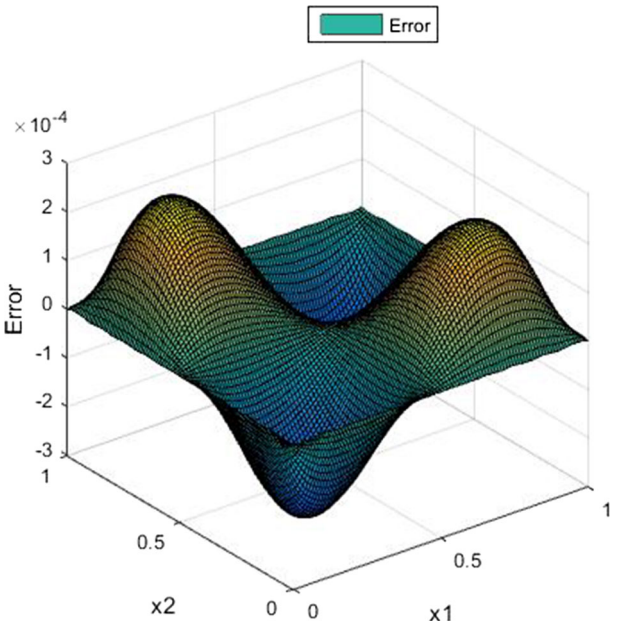


Fig. 11 Plot of error between exact solution and BeNN solution

Table 4 Comparison between exact and BeNN solution at test points (Example 5)

x_1	x_2	Exact solution	BeNN solution	BeNN error	Chebyshev error [43]
0	0.2318	0.012454901432000	0.012454901432000	0	0
0.2174	0.7490	0.513009852104651	0.512773580821592	2.4e−4	1.3e−2
0.5870	0.3285	0.346077236496052	0.345983957233659	9.3e−5	3.2e−2
0.3971	0.6481	0.449964154569349	0.449888592973102	7.6e−5	2.6e−3
0.7193	0.2871	0.361892942424939	0.361693139135802	1.1e−4	2.6e−2
0.8752	0.5380	0.429665815073745	0.429703532019806	3.8e−5	1.6e−3
0.9471	0.4691	0.407384517762609	0.407368099922590	1.6e−5	7.6e−3
0.4521	0.8241	0.643786002811775	0.643744179557902	4.2e−5	1.5e−2
0.2980	0.9153	0.790413322406531	0.790275502139587	1.4e−4	5.5e−3
0.6320	0.1834	0.339204362555939	0.339044986521569	1.6e−4	2.3e−2

Example 6 We consider the following partial differential equations with mixed boundary conditions [31]:

$$\begin{cases} \nabla^4 y(x_1,x_2) = 24(x_1^4+x_2^4) - 144(x_1^2+x_2^2) + 288x_1^2x_2^2 + 80 \\ y(x_1,x_2) = 0, (x_1,x_2) \in \partial([-1,1]^2) \\ \frac{\partial y}{\partial n}(x_1,x_2) = 0, (x_1,x_2) \in \partial([-1,1]^2) \end{cases} \quad (x_1,x_2) \in [-1,1]^2 \qquad (32)$$

First, we construct the experimental solution of the equation according to (13, 14):

$$y_t(x_1,x_2,p) = \left(x_1^2-1\right)^2\left(x_2^2-1\right)^2 N(x_1,x_2,p) \qquad (33)$$

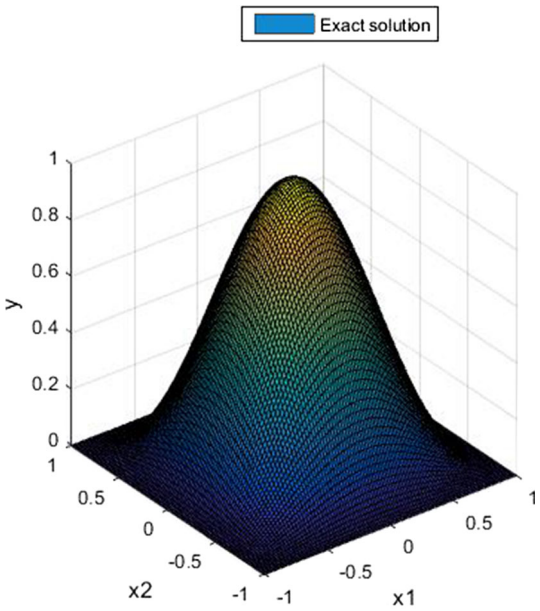


Fig. 12 Figure of exact solution

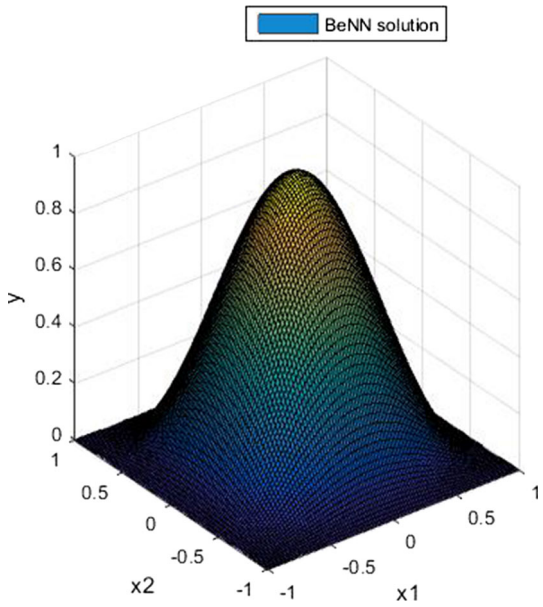
Table 5 Comparison between exact and BeNN solution at test points (Example 6)

x_1	x_2	Exact solution	BeNN solution	BeNN error	MLP error [31]
−0.74	−1.2	0.039623291136000	0.039623291135997	3e−15	2.0e−6
−0.46	0.0	0.621574560000000	0.621574560000025	2.5−14	1.6e−6
−0.3	1.2	0.160320160000000	0.160320160000004	4e−15	3.0e−6
0.4	0.0	0.705600000000000	0.705600000000000	0	7.0e−7
0.78	0.98	0.000240478214170	0.000240478214170	0	0
0.92	−0.35	0.018166726656000	0.018166726656000	0	0
1.2	−1.2	0.037480960000000	0.037480959999993	7e−15	5.7e−6
1.2	0.0	0.193600000000000	0.193600000000000	0	1.1e−7

In order to have a fair comparison with the results reported in [31], 121 equidistant points in the given domain $[-1, 1]\times[-1, 1]$ are used for training. As in previous cases the exact solution and BeNN solution are shown in Figs. 12 and 13, respectively. Plot of error function is citied in Fig. 14. From the obtained results, it is apparent that our method outperforms the method in [31] in terms of accuracy, the accuracy of the error in [31] is $O(10^{-5})$, the accuracy of the error obtained by our method is $O(10^{-14})$. Finally, Table 5 shows the BeNN solution at the test points through the direct use of converged weights. The maximum absolute error in the test points is 5.7×10^{-6} in [31], and the obtained maximum absolute error by BeNN is 2.5×10^{-14}. It is clear that the error between BeNN solution and exact solution remains low in the test points.

Remark 1 The popular learning algorithm used in feedforward neural networks is the BP learning algorithm, its calculation speed is very slow, mainly for the following two reasons: (1) gradient descent algorithm is very time-consuming, when the learning rate is too small, the learning algorithm converges very slowly. However, when the learning rate is too large,

Fig. 13 Figure of BeNN solution



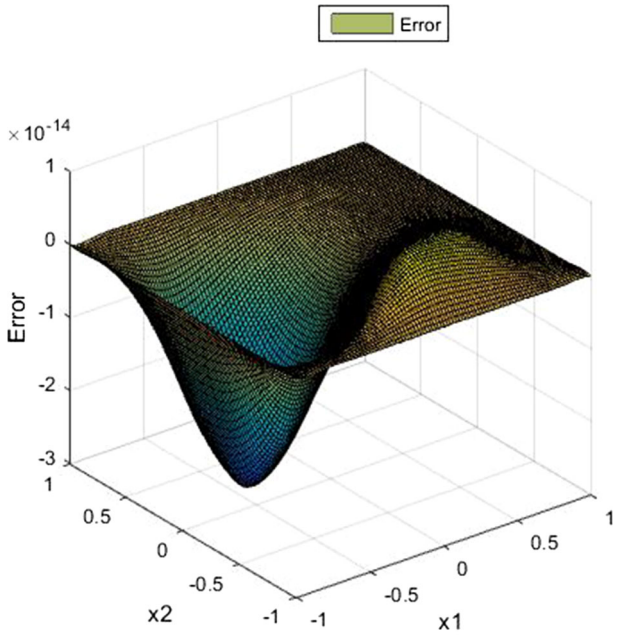


Fig. 14 Plot of error between exact solution and BeNN solution

the algorithm becomes unstable and diverges. (2) All the parameters of the networks are tuned iteratively by using such learning algorithms,and many iterative learning steps may be required in order to obtain better learning performance [44]. In our proposed method, only output layer weight parameters need to be determined. The BENN model is finally simplified

as a linear system, and the weight parameters can be obtained by a simple generalized inverse matrix, which greatly improves the calculation speed.

5 Conclusion

In this paper, we propose a single-layer Bernstein neural network model to solve the partial differential equation. Here we use a single-layer functional link artificial neural network architecture, the hidden layer is replaced by a functional extension block for extended input mode, and uses the Bernstein basis function to extend the dimension of the input data. We combine the Extreme Learning Machine algorithm to determine the network parameters, which makes the computational complexity of the Bernstein neural network model less than other traditional numerical method. The numerical experiments show the good consistency of the results, which proves the reliability and superiority of the proposed method.

Acknowledgements This study was funded by the National Natural Science Foundation of China under Grants 61375063, 61271355, 11301549 and 11271378. This study was also funded by the Central South University Caitian Xuanzhu student innovation and business Projects 201710533542.

Compliance with Ethical Standards

Conflict of interest All authors have no conflict of interest.

References

1. Butcher JC (1987) The numerical analysis of ordinary differential equations: Runge Kutta and general linear methods. Math Comput 51(183):693
2. Verwer JG (1996) Explicit Runge–Kutta methods for parabolic partial differential equations. Appl Numer Math 22(1–3):359–379
3. Hamming RW (1959) Stable predictor corrector methods for ordinary differential equations. J ACM 6(1):37–47
4. Tseng AA, Gu SX (1989) A finite difference scheme with arbitrary mesh system for solving high order partial differential equations. Comput Struct 31(3):319–328
5. Wu B, White RE (2004) One implementation variant of finite difference method for solving ODEs/DAEs. Comput Chem Eng 28(3):303–309
6. Kumar M, Kumar P (2009) Computational method for finding various solutions for a quasilinear elliptic equations with periodic solutions. Adv Eng Softw 40(11):1104–1111
7. Thomee V (2001) From finite difference to finite elements:a short history of numerical analysis of partial differential equations. J Comput Appl Math 128(1–2):1–54
8. Sallam S, Ameen W (1990) Numerical solution of general nth order differential equations via splines. Appl Numer Math 6(3):225–238
9. Ei-Hawary HM, Mahmoud SM (2003) Spline collocation methods for solving delay differential equations. Appl Math Comput 146(2–3):359–372
10. Tsoulos IG, Gavrilis D, Glavas E (2009) Solving differential equations with constructed neural networks. Neurocomputing 72(10–12):2385–2391
11. Liu J, Hou G (2011) Numerical solutions of the space-and time-fractional coupled Burgers equations by generalized differential transform method. Appl Math Comput 217(16):7001–7008
12. Darnia P, Ebadian A (2007) A method for the numerical solution of the integro-differential equations. Appl Math Comput 188(1):657–668
13. Kumar M, Singh N (2009) A collection of computational techniques for solving singular boundary value problems. Adv Eng Softw 40(4):288–297
14. Wang Q, Cheng D (2005) Numerical solution of damped nonlinear Klein–Gordon equations using variational method and finite element approach. Appl Math Comput 162(1):381–401

15. Kumar M, Singh N (2010) Modified Adomian decomposition method and computer implementation for solving singular boundary value problems arising in various physical problems. Comput Chem Eng 34(11):1750–1760
16. Yang X, Liu Y, Bai S (2006) A numerical solution of second order linear partial differential equation by differential transform. Appl Math Comput 173(2):792–802
17. Erturk VS, Momani S (2008) Solving system of fractional differential equations using differential transform method. J Comput Appl Math 215(1):142–151
18. Coronel-Escamilla A, Gomez-Aguilar JF, Torres L, Escobar-Jimenez RF, Valtierra-Rodriguez M (2017) Synchronization of chaotic systems involving fractional operators of Liouville–Caputo type with variable-order. Phys A 487:1–21
19. Coronel-Escamilla A, Gomez-Aguilar JF, Torres L, Escobar-Jimenez RF (2018) A numerical solution for a variable-order reaction-diffusion model by using fractional derivatives with non-local and non-singular kernel. Phys A 491:406–424
20. Zuniga-Aguilar CJ, Gomez-Aguilar JF, Escobar-Jimenez RF, Romero-Ugalde HM (2018) Robust control for fractional variable-order chaotic systems with non-singular kernel. Eur Phys J Plus 133(1):1–13
21. Yazdi HS, Pakdaman M, Modaghegh H (2011) Unsupervised kernel least mean square algorithm for solving ordinary differential equations. Neurocomputing 74(12–13):2062–2071
22. Lagaris IE, Likas A (1998) Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans Neural Netw 9(5):987–1000
23. Lagaris IE, Likas AC (2000) Neural-network methods for boundary value problems with irregular boundaries. IEEE Trans Neural Netw 11(5):1041
24. Mai-Duy N, Tran-Cong T (2001) Numerical solution of differential equations using multiquadric radial basis function networks. Neural Netw 14(2):185–199
25. Aarts LP, Veer PVD (2001) Neural network method for solving partial differential equations. Neural Process Lett 14(3):261–271
26. Mai-Duy N, Tran-Cong T (2003) Approximation of function and its derivatives using radial basis function networks. Appl Math Model 27(3):197–220
27. Li J, Luo S, Qi Y, Huang Y (2003) Numerical solution of elliptic partial differential equation using radial basis function neural networks. IJCNN 16(5–6):85–90
28. Malek A, Beidokhti RS (2006) Numerical solution for high order differential equations using a hybrid neural network—optimization method. Appl Math Comput 183(1):260–271. https://doi.org/10.1016/j.amc.2006.05.068
29. Xu LY, Wen H, Zeng ZZ (2007) The algorithm of neural networks on the initial value problems in ordinary differential equations. In: IEEE conference on industrial electronics & applications, pp 813–816
30. Aein MJ, Talebi HA (2009) Introducing a training methodology for cellular neural networks solving partial differential equations. IJCNN, Atlanta, Georgia, USA, June 14–19, pp 71–75
31. Beidokhti RS, Malek A (2009) Solving initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques. J Franklin Inst 346(9):898–913
32. Xu C, Wang C, Ji F, Yuan X (2012) Finite-element neural network-based solving 3-D differential equations in MFL. IEEE Trans Magn 48(12):4747–4756
33. Mcfall KS (2013) Automated design parameter selection for neural networks solving coupled partial differential equations with discontinuities. J Franklin Inst 350(2):300–317
34. Zuniga-Aguilar CJ, Romero-Ugalde HM, Gomez-Aguilar JF, Escobar-Jimenez RF, Valtierra-Rodriguez M (2017) Solving fractional differential equations of variable-order involving operators with Mittag–Leffler kernel using artificial neural networks. Chaos Soliton Fract 103:382–403
35. Pao YH, Phillips SM (1995) The functional link net and learning optimal control. Neurocomputing 9(2):149–164
36. Lee TT, Jeng JT (1998) The Chebyshev-polynomials based unified model neural networks for function approximation. IEEE Trans Syst Man Cybern 28(6):925–935
37. Yang SS, Tseng CS (1996) An orthogonal neural network for function approximation. IEEE Trans Syst Man Cybern 26(5):779
38. Patra JC, Juhola M, Meher PK (2008) Intelligent sensors using computationally efficient Chebyshev neural networks. IET Sci Meas Technol 2(2):68–75
39. Weng WD, Yang CS, Lin RC (2007) A channel equalizer using reduced decision feedback Chebyshev functional link artificial neural networks. Inf Sci 177(13):2642–2654
40. Patra JC, Kot AC (2002) Nonlinear dynamic system identification using Chebyshev functional link artificial neural network. IEEE Trans Syst Man Cybern 32(4):505–511
41. Purwar S, Kar IN, Jha AN (2007) Online system identification of complex systems using Chebyshev neural network. Appl Soft Comput 7(1):364–372

42. Mall S, Chakraverty S (2016) Application of Legendre neural network for solving ordinary differential equations. Appl Soft Comput 43(C):347–356. https://doi.org/10.1016/j.asoc.2015.10.069
43. Mall S, Chakraverty S (2017) Single layer Chebyshev neural network model for solving elliptic partial differential equations. Neural Process Lett 45(3):825–840
44. Huang GB, Zhu QY, Siew CK (2006) Extreme learning machine: theory and applications. Neurocomputing 70(1–3):489–501

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

View publication stats