Soft Computing (2023) 27:3955–3964
https://doi.org/10.1007/s00500-022-07529-3

APPLICATION OF SOFT COMPUTING

Radial basis function neural network with extreme learning machine algorithm for solving ordinary differential equations

Min Liu[1] · Wenping Peng[2] · Muzhou Hou[1] · Zhongchu Tian[2]

Accepted: 24 September 2022 / Published online: 28 October 2022

Abstract
We present a novel numerical method for solving ordinary differential equations using radial basis function (RBF) network with extreme learning machine algorithm. A single-layer RBF link neural network model has been developed for the proposed method. The weight from the hidden layer to the output layer can be calculated efficiently by extreme learning machine algorithm. The experimental comparison of various methods proves that the proposed method shows better performance than the existing methods.

Keywords Radial basis function network · Extreme learning machine · Ordinary differential equation

1 Introduction

Differential equations (DEs) play an important role in various fields of science and engineering. Many problems encountered in many fields of physics, economics, biology, chemistry, population, resources, etc., can be solved by DEs models. Therefore, when the ODEs were put forward, they became a useful tool for human beings to understand nature and explore the laws of motion of the material world. However, in many cases, the analytical solution of the DEs does not exist or is difficult to obtain. Therefore, the numerical solution of DEs becomes an important research direction. At present, there are many numerical methods for solving DEs, such as finite difference method, finite element method, finite volume method, Runge–Kutta method, and other methods. The computational complexity of traditional methods increases rapidly with the increase in sampling points, while the method based on artificial neural network (ANN) can effectively avoid this problem. In addition, the traditional methods can only obtain numerical solutions at finite points, and it needs repeated calculation to obtain numerical solutions at other points, while the result obtained by ANN-based model is a closed analytic form. We can use this form to get the numerical solution of any point.

In the past few decades, researchers have been devoting themselves to the study of various machine intelligence methods, especially ANN-based model for solving Des. In 1990, Lee and kang presented a Hopfield neural network model for the solutions of DEs. In 2006, Malek and Beidokhti presented a hybrid neural network for solving higher-order differential equations. In 2016, Mall and Chakraverty introduced.

Legendre functional link neural network for the solution of DEs. In 2017, Mall and Chakraverty used Chebyshev polynomial as an activation function to construct the approximate solution of DEs.

RBFNN has the advantages of simple training, fast convergence and can overcome the local minimum problem. It is widely used in function approximation, speech recognition, pattern recognition, image processing, and other fields. In 1991, Park and Sandberg proved that RBFNN with a hidden layer can be effectively used for universal approximation. In 2012, Lin, Chen and Sze proposed a radial basis function method for solving the Helmholtz problem. In 2017, Qu obtained the numerical solution of the fractional Riccati equation and the fractional

✉ Muzhou Hou
houmuzhou@sina.com

Min Liu
liuhellomin@163.com

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

[2] School of Civil Engineering, Changsha University of Science and Technology, Changsha 410114, China

Langevin equation by applying the cosine radial basis function network.

The rest of this paper is organized as follows: Section 2 describes the problem to be solved. Section 3 introduces the proposed RBFNN. Section 4 introduces the process of solving parameters of RBFNN using extreme learning machine algorithm. Section 5 shows some numerical results obtained with the RBFNN model. Finally, the last section is some conclusions.

2 Radial basis functional link neural network model

Single-layer radial basis functional link neural network model has been considered for the present problem. Figure 1 depicts the three-layer structure of RBFNN. The first layer of the network has only one node; the input data is the independent variable of the ordinary differential equation. The general form of the nonlinear Gaussian basis function is as follows:

$$\varphi_i(x) = e^{-\frac{(x-c_i)^2}{\sigma^2}} \tag{1}$$

There are two parameters in the Gaussian basis function, and the choice of these two parameters has a great influence on the construction of the model, which will eventually affect the accuracy of the approximate solution. There are usually two ways to select the center. The first way is to select from sample points, and the second way is to self-organize selection method, such as clustering samples, and gradient training method. When solving the problem of solving differential equations, the sample point is the independent variable x. The following method of selecting sample points is to sample uniformly in the solution area. Therefore, whether the first method is used to select the center point or the second method, the final center point will be uniformly sampled from the solution area, where σ is the width parameter, which determines the shape and scope of the Gaussian basis function. This value is chosen empirically. c_i is the center of the i-th Gaussian basis function, x is the input of the network. More closer the input x is to the center c_i, more larger the output of the corresponding hidden layer node is. Finally, the output of the network can be calculated by the following formula:

$$z(x,\beta) = \sum_{i=1}^{n} \beta_i \varphi_i(x) \tag{2}$$

where $\beta_i, i = 1, 2, \ldots n$ is the weight from the hidden layer to the output layer.

RBF is used as the base of hidden layer neurons to form the hidden layer space, so that input vectors can be mapped directly to the hidden space (i.e., without weight connection). When the center of RBF is determined, the mapping relationship is determined. The mapping from the hidden layer space to the output layer space is linear, that is, the output of the network is the linear weighted sum of the output of the hidden layer neurons, where the weight is the adjustable parameter of the network. Thus, in general, the mapping of the network from input to output is nonlinear, while the network is linear for adjustable parameters.

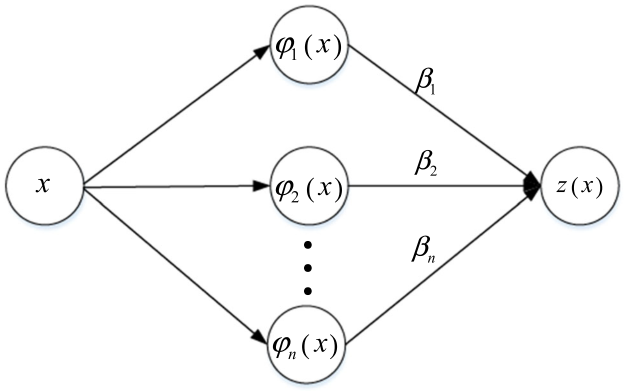

Fig. 1 The structure of RBFNN

3 Description of the problem

The general form of ordinary differential equations is as follows:

$$G\left(x, y(x), \nabla y(x), \nabla^2 y(x), \ldots, \nabla^k y(x)\right) = 0, x \in \overline{D} \subseteq R^n \tag{3}$$

where $\nabla^i y(x), i = 1, 2 \ldots k$.. The trial solution can be written as follows:

$$y_T(x,\beta) = A(x) + F(x, z(x,\beta)) \tag{4}$$

The first part of the trial solution $A(x)$ ensures the trial solution satisfies the initial or boundary conditions of the ordinary differential equation and contains no adjustable parameters. The second part of the trial solution contains adjustable parameters. The closer the error function is to zero, the better the parameters are.

In Sect. 5, Examples 1, 2, and 3 show the results of RBFNN method for solving first-order ordinary differential equations. The general form of first-order ordinary differential equation is as follows:

$$\frac{\partial y(x)}{\partial x} = f(x,y),\ x \in [a,b] \tag{5}$$

When the initial value condition satisfies $y(a) = A$, the trial solution can be written as follows:

$$y_T(x,\beta) = A(x) + (x-a) z(x,\beta) \tag{6}$$

In Sect. 5, Examples 4 and 5 are two examples of solving a system of first-order ordinary differential equations. The general form of first-order system of ordinary differential equations is as follows:

$$\frac{\partial y_i(x)}{\partial x} = f(x, y_1(x), y_2(x), \ldots y_m(x)),\ x \in [a,b], i = 1,2,\ldots m \tag{7}$$

When the initial value condition satisfies $y_i(a) = A_i$, the trial solution can be written as follows:

$$y_{iT}(x,\beta) = A_i + (x-a)z_i(x,\beta_i), i = 1,2,\ldots m \tag{8}$$

Examples 6 and 7 are two second-order ordinary differential equations; the general form of second-order ordinary differential equations is as follows:

$$\frac{\partial^2 y(x)}{\partial x} = f\left(x, y(x), \frac{\partial y(x)}{\partial x}\right),\ x \in [a,b] \tag{9}$$

Second-order ordinary differential equations can be divided into initial value problems and boundary value problems. When the initial value condition satisfies $y(a) = A, y(b) = B$, the trial solution can be written as follows:

$$y_T(x,\beta) = \frac{bA - aB}{b-a} + \frac{B-A}{b-a}x + (x-a)(x-b)z(x,\beta) \tag{10}$$

When the boundary value condition satisfies $y(a) = A, y'(a) = A'$, the trial solution can be written as follows:

$$y_T(x,\beta) = A - A'a + A'x + (x-a)^2 z(x,\beta) \tag{11}$$

4 Extreme learning machine algorithm with proposed RBFNN model

We minimize the error function by adjusting the parameters β. For a point x in the solution region, when the numerical solution is equal to the exact solution, the error at this point is zero. Then, the Eq. (12) can be obtained by substituting the trial solution $y_T(x,\beta)$ and its derivatives of this point into the equation to be solved.

$$G\big(x, y_T(x,\beta), \nabla y_T(x,\beta), \nabla^2 y_T(x,\beta), \ldots, \nabla^k y_T(x,\beta)\big) = 0 \tag{12}$$

In Eq. (12), only β is unknown. Put the part containing β on the left side of the equation and the other parts on the right side of the equation to get a system of linear equations $H\beta = T$ about β.

Let us take the following system of ordinary differential equations as an example:

$$\begin{cases} \dfrac{dy_1}{\mathrm{d}x} + g_1(x)y_1(x) + g_2(x)y_2(x) = f_1(x), y_1(x_0) = A_1 \\ \dfrac{dy_2}{\mathrm{d}x} + h_1(x)y_1(x) + h_2(x)y_2(x) = f_2(x), y_2(x_0) = A_2 \end{cases}, x \in [a,b] \tag{13}$$

Selecting m sampling points and n hidden layer neurons, according to (8), the trial solutions can be written as follows:

$$y_{jt}(x) = A_j + (x - x_0)\sum_{i=1}^{n} \beta_{ji}\varphi_i(x), j = 1,2. \tag{14}$$

By substituting (14) and m sampling points into (13), the following linear equations with respect to β are obtained.

$$\begin{cases} \sum_{i=1}^{n} \beta_{1i}\varphi_i(x_j)\left\{(x_j - x_0)\left[\left(-\frac{\sigma^2}{2}\right)(x_j - c_i) + g_1(x_j)\right] + 1\right\} + \\ \sum_{i=1}^{n} \beta_{2i}\varphi_i(x_j)\left[(x_j - x_0)g_2(x_j)\right] = f_1(x_j) - g_1(x_j)A_1 - g_2(x_j)A_2 \\ \sum_{i=1}^{n} \beta_{2i}\varphi_i(x_j)\left\{(x_j - x_0)\left[\left(-\frac{\sigma^2}{2}\right)(x_j - c_i) + h_2(x_j)\right] + 1\right\} + \\ \sum_{i=1}^{n} \beta_{1i}\varphi_i(x_j)\left[(x_j - x_0)h_1(x_j)\right] = f_2(x) - h_1(x_j)A_1 - h_2(x_j)A_2 \end{cases}, j = 1,2\ldots m. \tag{15}$$

Set $H_{11}, H_{12}, H_{21}, H_{22}$ be matrices of size $m \times n$, β_1, β_2 be matrices of size $n \times 1$, T_1, T_2 be matrices of size $m \times 1$, and

$$H_{11}(j,i) = \varphi_i(x_j)\left\{(x_j - x_0)\left[\left(-\frac{\sigma^2}{2}\right)(x_j - c_i) + g_1(x_j)\right] + 1\right\},$$
$$H_{22}(j,i) = \varphi_i(x_j)\left[(x_j - x_0)g_2(x_j)\right],$$
$$H_{21}(j,i) = \varphi_i(x_j)\left[(x_j - x_0)h_1(x_j)\right],$$
$$H_{22}(j,i) = \varphi_i(x_j)\left\{(x_j - x_0)\left[\left(-\frac{\sigma^2}{2}\right)(x_j - c_i) + h_2(x_j)\right] + 1\right\},$$

$$\beta_1 = (\beta_{11}, \beta_{12}, \ldots \beta_{1n})^T, \beta_2 = (\beta_{21}, \beta_{22}, \ldots \beta_{2n})^T,$$

$$T_1 = \begin{pmatrix} f_1(x_1) - g_1(x_1)A_1 - g_2(x_1)A_2 \\ f_1(x_2) - g_1(x_2)A_1 - g_2(x_2)A_2 \\ \vdots \\ f_1(x_m) - g_1(x_m)A_1 - g_2(x_m)A_2 \end{pmatrix},$$

$$T_2 = \begin{pmatrix} f_2(x_1) - h_1(x_1)A_1 - h_2(x_1)A_2 \\ f_2(x_2) - h_1(x_2)A_1 - h_2(x_2)A_2 \\ \vdots \\ f_2(x_m) - h_1(x_m)A_1 - h_2(x_m)A_2 \end{pmatrix},$$

$$H = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}_{2m\times 2n}, \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix}_{2n\times 1}, T = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}_{2m\times 1}$$

Then, (15) can be written in the following matrix form:

$$H\beta = T.$$

Solutions of the system of linear equation $H\beta = T$ are the most appropriate parameter value. When H is a reversible matrix,$\beta = H^{-1}T$. In many cases, if H not a reversible matrix, then $\beta = H^{\dagger}T$, where $H^{\dagger}$ is the Moore–Penrose generalized inverse of matrix. Moreover, the solution of $H\beta = T$ is unique.

The steps of using extreme learning machine algorithm with the proposed RBFNN model to solve ODEs are as follows:

- *Step 1* Determinate the sampling points $x_1, x_2, \ldots x_m$;
- *Step 2* Calculate the network output based on sampling points $z(x_i, \beta), i = 1, 2, \ldots m$;
- *Step 3* Write the expression of the trial solution $y_T(x, \beta)$ according to the initial value condition or the boundary value condition;
- *Step 4* substituting the trial solution $y_T(x, \beta)$ and its derivatives of this point into the equation to be solved. Then, we get this equation:

$$G\big(x, y_T(x, \beta), \nabla y_T(x, \beta), \nabla^2 y_T(x, \beta), \ldots, \nabla^k y_T(x, \beta)\big) = 0$$

- *Step 5* Transform this formula into a matrix form $H\beta = T$;
- *Step 6* Solving equations in Step 5 by $\beta = H^{\dagger}T$;
- *Step 7* Substitute the parameter values obtained in step 6 into the trial function. Thus, the numerical solution of any point in the domain can be obtained by substituting the independent variable into the trial function.

5 Numerical examples

In this section, some examples are given to verify the effectiveness of the proposed method. To compare the proposed method with other methods conveniently, two error functions are used in this paper. Suppose the sampling point is $x_1, x_2, \ldots x_m$ and the exact solution of the ordinary differential equation at x_i is $y(x_i)$.The means of average mean-squared error are as follows:

$$E_1(x) = \frac{1}{m}\sum_{j=1}^{m}\big(y_t(x_j, \beta) - y(x_j)\big)^2$$

and the maximum absolute error is as follows:

$$E_2(x) = \max_i \max_j \left| y_{it}(x_j, \beta) - y(x_j) \right|$$

We illustrate the effectiveness of the proposed method from three perspectives. The first method is to compare the approximate solutions of sampling points (training points) and test points with their exact solutions, namely $\left|y_t(x_j, \beta) - Y(x_j)\right|$. The second is to compare the approximate solution obtained by our method with that obtained by other classical methods. And the third is to compare the solutions obtained using different numbers of hidden layer neurons and different numbers of sampling points. In this paper, the center of the activation function RBF is obtained by uniform sampling in the solution region, and we take the width parameter $\sigma = 0.8$.

Example 1 Consider the following first-order ordinary differential equations:

$$\begin{cases} \dfrac{\mathrm{dy}}{\mathrm{dx}} + \left(x + \dfrac{1 + 3x^2}{1 + x + x^3}\right)y = 2x + x^3 + x^2\left(\dfrac{1 + 3x^2}{1 + x + x^3}\right), x \\ y(0) = 1 \end{cases} \in [0, 1]$$

with the analytical solution $y(x) = \frac{e^{-\frac{x^2}{2}}}{1+x+x^3} + x^2, x \in [0, 1]$.

The RBFNN trail solution in this case is:

$$y_t(x) = 1 + xz(x).$$

Table 1 compares the proposed algorithm with the RBF Net (Rizaner et al. 2018) that minimizes error with the gradient descent method. When the number of hidden neurons and activation function is the same, the solution accuracy of the proposed algorithm is higher. Table 2 compares the approximate solutions obtained by the proposed method with other methods. It is clearly seen from the data in Table 2 that the proposed method outperforms all the other methods. Figure 2 shows the exact solution

Table 1 Error comparison of Example 1

Error1	$n = 3$	$n = 5$	$n = 7$	$n = 9$
RBF Net in Rizaner et al. 2018	3.48×10^{-05}	3.54×10^{-06}	7.75×10^{-10}	6.80×10^{-10}
RBFNN	6.35×10^{-07}	2.54×10^{-09}	1.76×10^{-11}	7.56×10^{-14}

Table 2 The exact solution and approximations for example 1

x	Exact solution	Euler	Runge Kutta	RBF Net in Rizaner et al. 2018(n = 9)	RBFNN solution(n = 9)
0.00	1.0000	1.000	1.0000	1.0000	1.0000
0.05	0.9536	0.9500	0.9536	0.9536	0.9536
0.01	0.9137	0.9072	0.9138	0.9137	0.9137
0.15	0.8798	0.8707	0.8799	0.8798	0.8798
0.20	0.8514	0.8401	0.8515	0.8514	0.8514
0.25	0.8283	0.8150	0.8283	0.8283	0.8283
0.30	0.8104	0.7953	0.8105	0.8104	0.8104
0.35	0.7978	0.7810	0.7979	0.7978	0.7978
0.40	0.7905	0.7721	0.7907	0.7905	0.7905
0.45	0.7889	0.7689	0.7890	0.7889	0.7889
0.50	0.7931	0.7717	0.7932	0.7930	0.7931
0.55	0.8033	0.7805	0.8035	0.8033	0.8033
0.60	0.8200	0.7958	0.8201	0.8199	0.8200
0.65	0.8431	0.8178	0.8433	0.8431	0.8431
0.70	0.8731	0.8467	0.8733	0.8731	0.8731
0.75	0.9101	0.8826	0.9102	0.9100	0.9101
0.80	0.9541	0.9258	0.9542	0.9540	0.9541
0.85	1.0053	0.9763	1.0054	1.0052	1.0053
0.90	1.0637	1.0342	1.0638	1.0637	1.0637
0.95	1.1293	1.0995	1.1294	1.1293	1.1293
1.00	1.2022	1.1721	1.2022	1.2021	1.2022
Error1	0	4.60×10^{-04}	1.24×10^{-08}	6.80×10^{-10}	7.56×10^{-14}

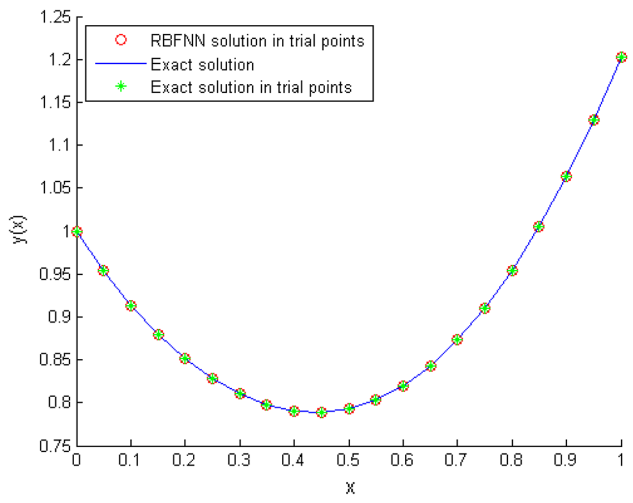

Fig. 2 Exact solution and approximations of example 1

Table 3 Error comparison of example 2

Error1	$n = 9$	$n = 15$	$n = 21$
RBF net in [26]	6.67×10^{-02}	1.96×10^{-05}	1.44×10^{-06}
RBFNN	2.73×10^{-05}	2.87×10^{-09}	5.85×10^{-12}

and approximations of Example 1. The basis function of the network in RBF net (Rizaner et al. 2018) is also the radial basis function. When the number of hidden layer neurons is 9, the accuracy of RBF Net is the highest with an error of 6.80×10^{-10}. When the number of hidden layer neurons in our network is 9, the accuracy is higher than that in RBF Net, and the error function is 7.56×10^{-14}.

Table 4 The exact solution and approximations for example 2

x	Exact Solution	RBF Net in Rizaner et al. 2018 (n = 21)	RBFNN (n = 21)	RBFNN (n = 90)
0.00	3.0000	3.0000	3.0000	3.0000
0.15	2.3438	2.2435	2.3438	2.3438
0.30	1.8142	1.8138	1.8142	1.8142
0.45	1.3511	1.3510	1.3511	1.3511
0.60	0.9348	0.9344	0.9348	0.9348
0.75	0.5763	0.5752	0.5763	0.5763
0.90	0.3012	0.2998	0.3012	0.3012
1.05	0.1318	0.1308	0.1318	0.1318
1.20	0.0726	0.0720	0.0726	0.0726
1.35	0.1038	0.1030	0.1038	0.1038
1.50	0.1845	0.1834	0.1845	0.1845
1.65	0.2643	0.2632	0.2643	0.2643
1.80	0.2988	0.2982	0.2988	0.2988
1.95	0.2638	0.2637	0.2638	0.2638
2.10	0.1625	0.1622	0.1625	0.1625
2.25	0.0235	0.02227	0.0235	0.0235
2.40	− 0.1095	− 0.1107	− 0.1095	− 0.1095
2.55	− 0.1937	− 0.1952	− 0.1937	− 0.1937
2.70	− 0.2025	− 0.2043	− 0.2025	− 0.2025
2.85	− 0.1348	− 0.1373	− 0.1348	− 0.1348
3.00	− 0.0157	− 0.0186	− 0.0157	− 0.0157
Error1	0	1.44×10^{-06}	5.85×10^{-12}	9.95×10^{-13}

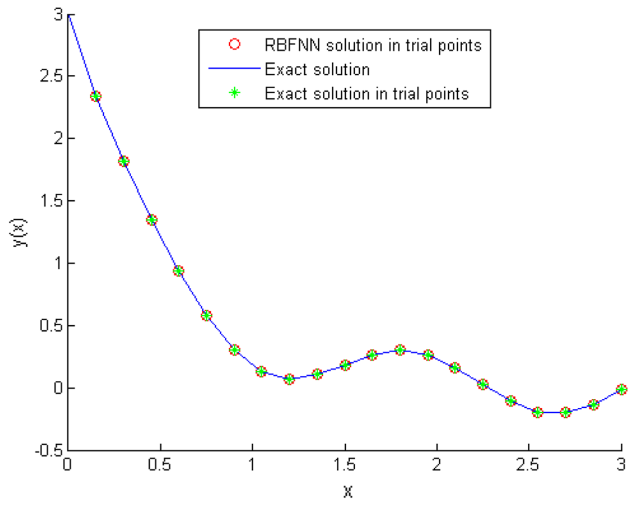

Fig. 3 Exact solution and approximations of example 2

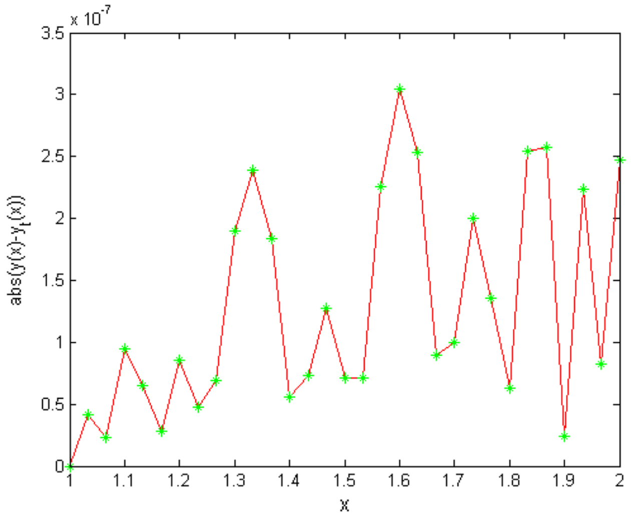

Fig. 4 The maximum absolute error of example 3

Example 2 Next, we look at the following first-order ordinary differential equations:

$$\begin{cases} \frac{dy}{dx} + 2y = \cos(4x) \\ y(0) = 3 \end{cases}, x \in [0, 3]$$

with the analytical solution:$y(x) = \frac{\cos(4x)+29e^{-2x}}{10} + \frac{\sin(4x)}{5}, x \in [0, 3]$.

The RBFNN trail solution in this case is:

$y_t(x) = 3 + xz(x)$.

From Table 3, we can see clearly that our algorithm is more accurate than the algorithm in RBF Net (Rizaner et al. 2018) when the network structure is the same. As can be seen from Table 4, with 21 RBF basis functions, the average mean-squared error of the solution obtained by extreme learning machine method is 5.85×10^{-12}. In RBF Net (Rizaner et al. 2018), the average mean squared is 1.44×10^{-06}.When the number of hidden layer basis functions is 90, the average mean-squared error can be reduced to 9.95×10^{-13} using the proposed method. Increasing the number of neurons in the hidden layer to more than 90 does not improve the accuracy effectively. Figure 3 shows the exact solution and approximations of example 2.

Example 3 Consider the following first-order ordinary differential equations:

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

Its exact solution is $y(x) = (x+1)^2 - 0.5e^x$.

The number of hidden layer neurons is 20, and 30 points are sampled equidistantly at interval [0, 2]. The maximum absolute error of the numerical solution is 2.20×10^{-6}. Figure 4 shows the maximum absolute error. By comparison, the maximum absolute error of the BeNN method (Sun et al. 2018) is 2.7×10^{-3}, and the maximum absolute error in [30] is 1.9×10^{-2}.

Example 4 Next, consider the following system of ordinary differential equations:

$$\begin{cases} \dfrac{dy_1}{dx} = \cos(x), y_1(0) = 0 \\ \dfrac{dy_2}{dx} = -y_1, y_2(0) = 1 \end{cases}, x \in [0, 1]$$

The exact solution of is $\begin{cases} y_1(x) = \sin(x) \\ y_2(x) = \cos(x) \end{cases}, x \in [0, 1]$.

The RBFNN trail solution in this case can be written as:

$$\begin{cases} y_1(x) = xz(x) \\ y_2(x) = 1 + xz(x) \end{cases}.$$

Table 5 The maximum absolute error for example 4

Error2	*m* = 5	*m* = 10	*m* = 15	*m* = 20
n = 10	4.26×10^{-04}	9.85×10^{-07}	5.42×10^{-07}	7.08×10^{-07}
n = 20	4.28×10^{-04}	5.30×10^{-07}	3.40×10^{-07}	1.77×10^{-07}
n = 30	4.30×10^{-04}	4.25×10^{-07}	8.67×10^{-08}	1.30×10^{-07}
n = 40	4.31×10^{-04}	1.06×10^{-06}	1.71×10^{-07}	1.93×10^{-07}
n = 50	4.32×10^{-04}	3.52×10^{-07}	2.91×10^{-07}	2.35×10^{-07}
n = 60	4.33×10^{-04}	2.66×10^{-07}	9.23×10^{-08}	5.84×10^{-08}

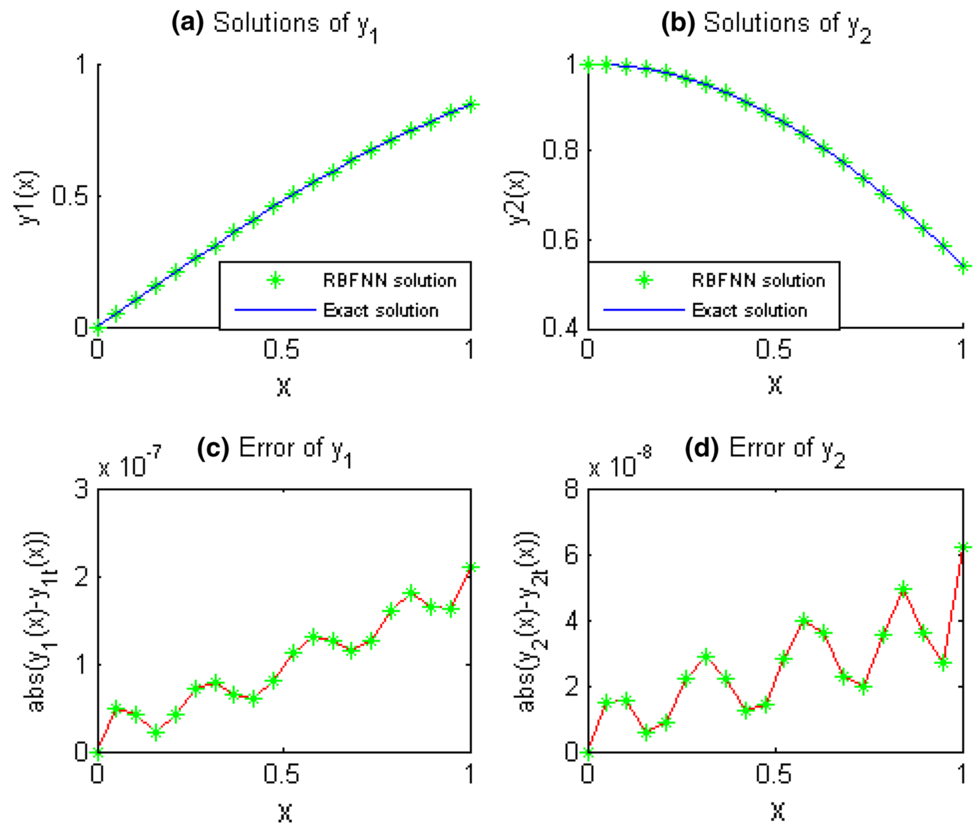

Fig. 5 Solutions and maximum absolute error of example 4

Figure 5a, b compares the exact solution with the approximate solution obtained by the proposed method. Figure 5c, d shows the maximum absolute error. Table 5 shows the effect of the number of sampling points and hidden layer neurons on the approximate solution. When there are more than 10 sampling points, the maximum absolute error function is less than 9.85×10^{-07}. When 20 sampling points and 60 hidden layer neurons are selected, the maximum absolute error is 5.84×10^{-08}. Continuing to increase the number of sampling points or hidden layer neurons can only slightly improve the accuracy of the solution.

Example 5 One more problem is given by

$$\begin{cases} \frac{dy_1}{\mathrm{d}x} + 2y_1 - y_2 = 2\sin(x), y_1(0) = 2 \\ \frac{dy_2}{\mathrm{d}x} - y_1 + 2y_2 = 2(cos(x) - \sin(x)), y_2(0) = 3 \end{cases}, x \in [0, 2]$$

The corresponding exact solution is $\begin{cases} y_1 = 2e^{-x} + \sin(x) \\ y_2 = 2e^{-x} + \cos(x) \end{cases}, x \in [0, 2].$

In this case, the RBFNN trail solution can be written as:

$$\begin{cases} y_1(x) = 2 + xz(x) \\ y_2(x) = 3 + xz(x) \end{cases}.$$

Using 40 hidden neurons and 40 points sampled equidistantly from interval [0, 2], the maximum absolute error of the numerical solution is 2.15×10^{-6}. Figure 6a, b compares the exact solution with the approximate solution obtained by the proposed method. Figure 6c, d shows the maximum absolute error.

Example 6 Next, consider a second-order ordinary differential boundary value problem.

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

the analytical solution is:$y(x) = \frac{\cos(1)-2}{\sin(1)}\sin(x) - \cos(x) + 2, x \in [0, 1]$.

In this case, the RBFNN trail can be written as:

$$y_t(x) = 1 - x + (x - x^2)z(x).$$

Take 50 equidistant points from interval $[0, 1]$ as sampling points. The maximum absolute error of the solution obtained by the proposed method is 8.55×10^{-08}. In unsupervised version of kernel least mean square (KLMS) algorithm (Yazdi et al. 2011), the maximum absolute error with 50 points from interval $[0, 1]$ is 3.5×10^{-02}. The numerical solution obtained by the constructed neural networks (CNN) (Tsoulos et al. 2009)

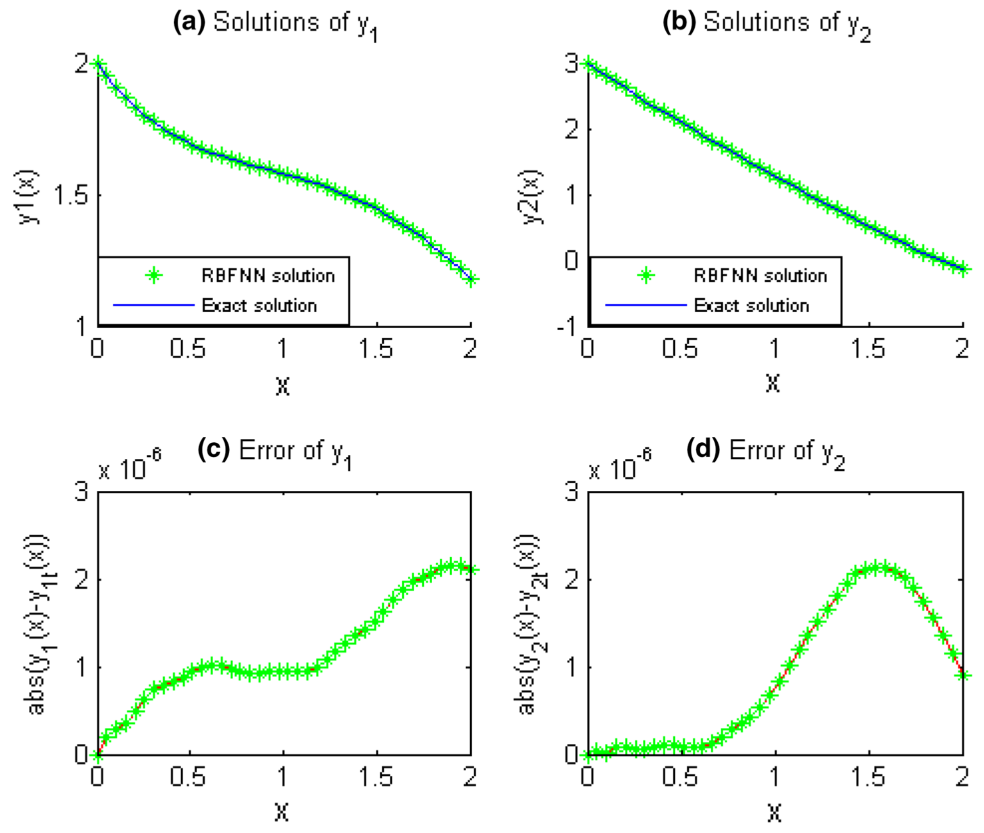

Fig. 6 Solutions and maximum absolute error of example 5

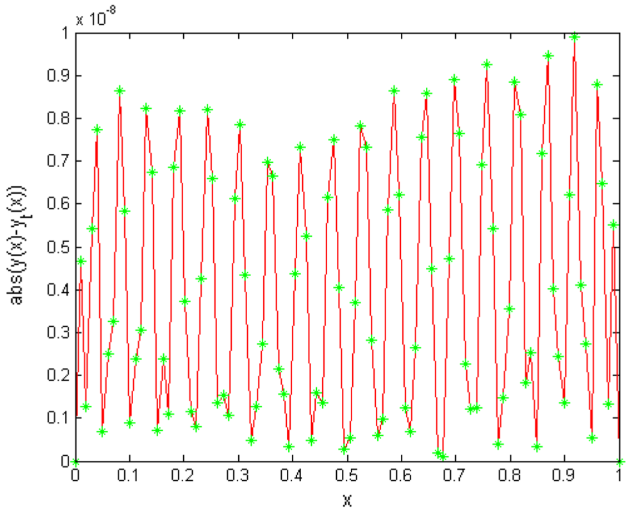

Fig. 7 The maximum absolute error of example 6

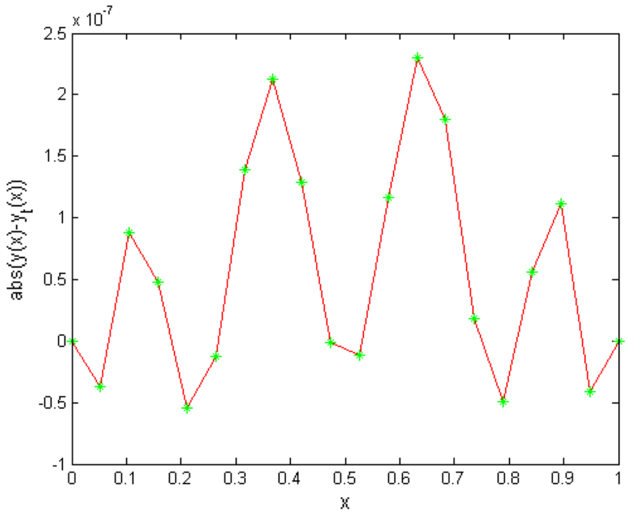

Fig. 8 The maximum absolute error of example 7

has an error of 5×10^{-01}. It can be seen that the proposed method can obtain more accurate numerical results when the sampling points are the same. The maximum absolute error of the proposed method can be reduced to 4.36×10^{-09}, by increasing the number of sampling points to 100. Figure 7 shows the maximum absolute error of example 6.

Example 7 Last, consider the following second-order ordinary differential boundary value problem.

$$\begin{cases} \frac{d^2y}{dx} + x\frac{dy}{dx} - 4y = 12x^2 - 3x \\ y(0) = 0, y(1) = 2 \end{cases}, x \in [0, 1],$$

The corresponding exact solution is $y(x) = x^4 + x, x \in [0, 1]$.

In this case, the RBFNN trail solution can be written as:

$$y_t(x) = 2x + (x^2 - x)z(x).$$

Using 10 hidden neurons and 20 points sampled equidistantly from interval [0, 1], the maximum absolute error of the numerical solution is 2.30×10^{-7}. Figure 8 shows the maximum absolute error of example 7.

6 Conclusion

Traditional methods of solving numerical solutions of differential equations can only find numerical solutions of certain discrete points, while the solving methods based on neural networks often require a large number of repeated iterations when solving network parameters, resulting in low solving efficiency. In this paper, radial basis function (RBF) network with extreme learning machine algorithm is proposed for solving ordinary differential equations (ODEs). A single hidden layer neural network has been used for the solution of ordinary differential equations, and the activation function of the hidden layer is the radial basis function. By substituting the trail solutions of training points into differential equations, a set of equations about network parameters can be obtained. Extreme learning machine algorithm can be used to solve this system of equations. Experiments show that the RBFNN model can be used to solve ordinary differential equations with good results. The advantages of this method are mainly the following two: 1. The numerical solution obtained by this method is an expression of the numerical solution, and this expression can be used to obtain the value of any point in the solution area; 2. The method of calculating neuron weights by this method required simple, efficient, and no iteration. Solution experiments show that the RBFNN model can be used to solve ordinary differential equations with good results.

Funding This study was funded by the Natural Science Foundation of Hunan Province, China under Grants 2022JJ30673.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval All procedures performed in studies involving human participants were in accordance with the ethical standards of the institutional and/or national research committee and with the 1964 Helsinki Declaration and its later amendments or comparable ethical standards.

Human and animal rights This article does not contain any studies with animals performed by any of the authors.

Informed consent Informed consent was obtained from all individual participants included in the study.

References

Boyce WE, DiPrima RC (2001) Elementary differential equations and boundary value problems. Wiley, New York

Butcher JC (1987) The numerical analysis of ordinary differential equations: Runge Kutta and general linear methods. Math Comput 51(183):693

Huang GB, Zhu QY, Siew CK (2006) Extreme learning machine: theory and applications. Neuro Comput 70(1–3):489–501

Jianyu L, Siwei L, Yingjian Q, Yaping H (2003) Numerical solution of elliptic partial differential equation using radial basis function neural networks. Neural Netw 16:729–734

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

Lagaris IE, Likas A, Fotiadis DI (1998) Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans Neural Netw 9(5):987–1000

Lee H, Kang I (1990) Neural algorithms for solving differential equations. J Comput Phys 91:110–117

Lin J, Chen W, Sze KY (2012) A new radial basis function for Helmholtz problems. Eng Anal Boundary Elem 36:1923–1930

Mai-Duy N, Tran-Cong T (2001) Numerical solution of differential equations using multiquadric radial basis function networks. Neural Netw 14(2):185–199

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

Mall S, Chakraverty S (2016) Application of legendre neural network for solving ordinary differential equations. Appl Soft Comput 43:347–356

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

Mayne AJ (1972) Generalized inverse of matrices and its applications. J Oper Res Soc 23(4):598

Nazemi A, Karami R (2017) A neural network approach for solving optimal control problems with inequality constraints and some applications. Neural Process Lett 45(3):995–1023

Park J, Sandberg IW (1991) Universal Approximation using radial-basis-function networks. Neuralcomput 3(2):246–257

Pinchover Y, Rubinsteinan J (2005) Introduction to partial differential equations. Cambridge University Press, Cambridge

Qu H (2017) Cosine radial basis function neural networks for solving fractional differential equations. Adv Appl Math Mech 9(3):667–679

Rao CR, Mitra SK (1971) Generalized inverse of matrices and its applications. Wiley, New York

Ricardo HJ (2009) A modern introduction to differential equations, 2nd edn.

Rizaner FB, Rizaner A (2018) Approximate solutions of initial value problems for ordinary differential equations using radial basis function networks. Neural Process Lett 48:1063–1071

Serre D (2002) Matrices: Theory and Applications. Springer, New York

Sun H, Hou M, Yang Y (2018) Solving partial differential equation based on Bernstein neural network and extreme learning machine algorithm. Neural Process Lett. https://doi.org/10.1007/s11063-018-9911-8

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

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

Tsoulos IG, Gavrilis D, Glavas E (2009) Solving differential equations with constructed neural networks. Neuro Comput 72(10–12):2385–2391

Verwer JG (1996) Explicit Runge-Kutta methods for parabolic partial differential equations. Appl Numer Math 22(1–3):359–379

WuB WhiteRE (2004) One implementation variant of finite difference method for solving ODEs/DAEs. Comput ChemEng 28(3):303–309

Xu LY, Hui W, Zeng ZZ (2007) The algorithm of neural networks on the initial value problems in ordinary differential equations. In: IEEE conference on industrial electronics and applications 2007 (Iciea 2007), pp 813–816 IEEE New York

Yang Y, Hou M, Luo, J, (2018). A novel improved extreme learning machine algorithm in solving ordinary differential equations by Legendre neural network methods. Adv Differ Equ (1)

Yazdi HS, Pakdaman M, Modaghegh H (2011) Unsupervised kernel least mean square algorithm for solving ordinary differential equations. Neurocomputing 74(12–13):2062–2071