Soft Computing
https://doi.org/10.1007/s00500-020-05401-w

METHODOLOGIES AND APPLICATION

Solving ordinary differential equations using an optimization technique based on training improved artificial neural networks

Shangjie Li[1] · Xingang Wang[2]

Abstract
The solution of ordinary differential equations (ODEs) arises in a wide variety of engineering problems. This paper presents a novel method for the numerical solution of ODEs using improved artificial neural networks (IANNs). In the first step, we derive an approximate solution of ODEs by artificial neural networks (ANNs). Then, we construct a joint cost function of network system, it consists of several error functions corresponding to different sample points, and we reformulate Levenberg–Marquardt (RLM) algorithm to adjust the network parameters. The advantages of this method are high calculation accuracy and fast convergence speed compared with other existed methods, also increasing the simulation stability of ANNs method. The performance of the new proposed method in terms of calculation accuracy and convergence speed is analyzed for several different types of nonlinear ODEs.

Keywords Differential equations · Improved artificial neural networks · Joint cost function · Reformulate Levenberg–Marquardt algorithm

1 Introduction

Differential equations are the basic structures in representation of engineering problems; many problems in science and engineering can be reduced to a set of differential equations through a process of mathematical modeling (Sneddon 2006; Ricardo 2009). Problems involving ordinary differential equations (ODEs) can be divided into two main categories, namely initial value problems (IVPs) and boundary value problems (BVPs). Analytic solutions for these problems are not generally available, especially for nonlinear ODEs; hence, the numerical methods must be applied. So various numerical methods such as predictor–corrector (Douglas and Jones 1963) and Runge–Kutta (Wambecq 1978) have been developed to solve these equations. These numerical methods require the discretization of domain into the number of finite domains or points where the function is approximated locally.

Various machine intelligence methods, in particular artificial neural networks (ANNs), have been used to solve differential equations; the ANNs method in comparison with other numerical methods has more advantages. Most other numerical methods are usually iterative in nature, where we fix the step size before initiating the computation; after the solution is obtained, if we want to know the solution in between steps, then again the procedure is to be repeated from initial stage. The ANNs method may be one of the reliefs where we may overcome this repetition of iterations. Moreover, the solution so obtained is analytic in nature, the solution is available in the form of a continuous and differentiable function, and thus, we can better study the properties of the solution if we need to do so by integrating or differentiating the solution analytically. Also the solution obtained is highly generalized and preserves accuracy throughout the domain despite very few points being used for obtaining the solution of differential equations. The methods which are based on the use of ANNs can be realized in hardware and hence offer the opportunity to tackle in real-time difficult differential equation problems arising in many engineering applications.

Communicated by V. Loia.

✉ Xingang Wang
wangxingang1217@126.com

[1] School of Mechanical Engineering and Automation, Northeastern University, Shenyang 110819, China

[2] Research Center of Mechanical Kinetics and Reliability, Northeastern University, Qinhuangdao 066004, China

Published online: 26 October 2020

In 1990, Lee and Kang (1990) used Hopfield neural network model to solve first order ODEs; Lagaris et al. (1998) used multilayer perceptron in their network architecture to solve both ordinary and partial differential equations. Also they solved BVPs with irregular boundaries using multilayer perceptron (Lagaris et al. 2000). Malek and Beidokhti (2006) proposed a method based on optimization technique and ANNs for the solution of ODEs. Mcfall and Mahan (2009) introduced an ANNs method for solution of mixed boundary value problems with irregular domain. Effati and Pakdaman (2010) used the ANNs to solve fuzzy differential equations. Yazdi and Pourreza (2010) applied the unsupervised kernel least mean square (KLMS) algorithm to solve ODEs which is an unsupervised version of KLMS. Another method for solving mixed BVPs on irregular domains has been implemented by Hoda and Nagla (2011). An unsupervised version of KLMS algorithm along with new optimization approach for solving first and second-order ODEs had been developed by Yazdi et al. (2011, 2012). Mehrkanoon et al. (2012) proposed a new approach based on least squares support vector machines (LS-SVMs) for solving linear and nonlinear ODEs; the approximate solution is presented in closed form by means of LS-SVMs. Mosleh (2013) presented a novel approach to solve system of fuzzy differential equations with fuzzy initial values by applying the universal approximation method through an artificial intelligence utility in a simple way. Mall and Chakraverty (2013) and Chakraverty and Mall (2014) proposed regression-based ANNs model for solving lower as well as higher-order ODEs. Also a new method based on single-layer Legendre neural network model had been developed to solve IVPs and BVPs (Mall and Chakraverty 2016). Then, Yang et al. (2018) proposed a novel improved extreme learning machine algorithm in solving ODEs by Legendre neural network method; the improved learning machine algorithm is used for network training when solving algebraic equation systems. Rizaner and Rizaner (2018) presented a numerical approach for the approximate solution of first order IVPs by using unsupervised radial basis function; the proposed unsupervised method is able to solve IVPs with high accuracy. Lu et al. (2019) introduced the improved LS-SVM algorithm for solving two-point and multi-point BVPs of high-order linear and nonlinear ODEs, third-order, fourth-order linear and nonlinear ODEs with two-point and multi-point boundary conditions are performed. Otadi (2019) presented a novel hybrid method based on fuzzy neural network for the solution of second-order differential equations with fuzzy boundary conditions. Hou et al. (2020) investigated the approximate ANNs solution of pantograph type differential equations; it is applied to various problems with a proportional delay term subject to initial or boundary conditions. Yang et al. (2020) used block neural network (BNN) method to solve several kinds of differential equations, the BNN is used to construct approximating function and its derivatives, and the improved extreme learning machine algorithm is designed to training network parameters (weights and biases). Panghal and Kumar (2020) proposed a new technique that eliminates the need of time-consuming optimization procedure for training of ANNs and used the extreme learning machine algorithm for calculating the ANNs parameters (weights and biases) so as to make it satisfy the differential equations and associated boundary conditions.

In this paper, we present a novel method for the numerical solution of ODEs using improved artificial neural networks (IANNs). Firstly, a trail solution is constructed, the trail solution is written as the sum of two parts, the first part is used to satisfy the initial and boundary conditions with no adjustable parameters (weights and biases), and the second part is a novel three-layered complex ANNs structure. Then, a targeted joint cost function of ANNs system is constructed, it consists of several error functions corresponding to different sample points, and we reformulate Levenberg–Marquardt (RLM) algorithm to adjust the parameters (weights and biases) of network system, this process determines the optimal values of the network parameters (weights and biases) for minimizing the corresponding joint cost function, and then, we get the approximate analytical solution expressed by ANNs.

Some of the main advantages of the new proposed method are as follows: Determining the approximate solution of the problem in a closed analytical form, the ability of applying the method to IVPs and BVPs for ODEs, faster convergence speed and higher simulation accuracy are illustrated in Sect. 4, the simulation stability of ANNs method is strengthened, and the method developed in this paper can be used as a paradigm for many numerical applications.

The remaining of this paper is organized as follows: In Sect. 2, some related basic definitions of ANNs and ODEs are briefly presented. Section 3 gives the detailed solution process of the new proposed IANNs method, construction of the joint cost function and corresponding training method. Illustrative numerical examples and results are discussed in Sect. 4, and in this section, we also compare with some existing methods. In Sect. 5, the conclusion of this work is outlined.

2 Basic definitions

Firstly, the three-layered ANNs structure and error back propagation (BP) training algorithm are introduced, and then, we describe the neural network formulations for ODEs briefly.

2.1 A three-layered ANNs model

For many years, ANNs has been successfully applied to a wide variety of real world applications, it can make a nonlinear mapping from the inputs to the outputs of corresponding system, and a three-layered ANNs can approximate any function. Figure 1 shows the typical structure of three-layered ANNs.

The function relationship between the network outputs and the inputs in Fig. 1 can be expressed as:

$$y = \sum_{j=1}^{m} h\left(\sum_{i=1}^{n} w_{ji}^{1} x_i + \vartheta_j\right) W_j^2 \tag{1}$$

This structure contains input, output and hidden layers. w_{ji}^1 is a weight parameter from ith input to jth neuron of hidden layer, W_j^2 is an jth weight parameter from hidden layer to the output layer, and ϑ_j is an jth bias in the hidden layer. The accuracy of the approximation depends on the number of neuron in the hidden layer, each input is multiplied with the corresponding weight value, and the sum of all the calculation results is added to bias value. Finally, activation function h determines the real-valued output of the neuron; in this paper we choose sigmoid function as activation function, and it can be expressed as:

$$h(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{2}$$

2.2 Error back propagation training algorithm

The training of ANNs means updating the parameters (weights and biases) so that the error value converges to zero. The training algorithm of error back propagation is widely used for updating the network parameters in supervised learning method. Here, the LM algorithm (Hagan and Menhaj 1994) is used for updating network parameters; the adjustment method of network parameters can be expressed as:

$$w_{n+1} = w_n - (J^{\mathrm{T}} J + \mu I)^{-1} J^{\mathrm{T}} e \tag{3}$$

where J is jacobian matrix, e represents the training error vector, I is the unit matrix; if one iteration is significant, then reduce adjustment factor μ; otherwise, increase it. When the value of μ is zero, which is same with quasi-Newton algorithm, when the adjustment factor μ goes to a large value, it is similar with gradient descent method.

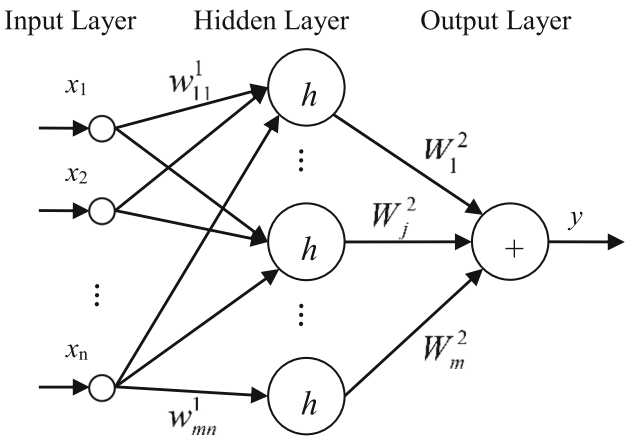

Fig. 1 Structure of the three-layered ANNs

2.3 Neural network formulations for ODEs

The basic idea for solving differential equations by ANNs was proposed by Lagaris et al. (1998). General form of differential equation may be written as:

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

where G is the function that defines the structure of differential equations, $x = (x_1, x_2, \ldots, x_n) \in \overline{D} \in R^n$ denotes the definition domain and the domain $\overline{D}$ is discretized over finite set of points D in R^n, $y(x)$ is the solution to be computed.

If $y_t(x, p)$ denotes the ANNs trail solution with adjustable parameters (weights and biases) p, then the above general differential equation changes to the follow form:

$$G(x, y_t(x,p), \nabla y_t(x,p), \nabla^2 y_t(x,p), \ldots, \nabla^n y_t(x,p)) = 0 \tag{5}$$

The error function used to measure the approximation degree to the original differential Eq. (4) is defined as:

$$E = \sum_{x_i \in D} \left(G((x_i), y_t(x_i,p), \nabla y_t(x_i,p), \nabla^2 y_t(x_i,p), \ldots, \nabla^n y_t(x_i,p))\right)^2 \tag{6}$$

Finding an approximation solution of Eq. (4) is equal to finding a function which minimizes the error E. Since multilayer feed forward ANNs are universal approximators (Hornik 1989), the ANNs trail solution is defined as the following form (Lagaris et al. 1998):

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

where $A(x)$ can satisfy initial or boundary conditions and contains no adjustable parameters (weights and biases); the second term $F(x, N(x,p))$ is a multilayer ANNs structure that consists of the input x and output $N(x,p)$. Then, the problem is transformed into an unconstrained optimization problem that is simple to deal with, in case of a given network architecture the problem is reduced to finding a configuration of parameters (weights and biases) corresponding to the minimum error E. As E is differentiable with respect to the parameters (weights and biases) for

most differential equations, efficient and gradient-based learning algorithm for ANNs can be employed.

3 Description of the proposed method

In this section, we briefly introduce a typical conventional ANNs method in solving second-order ODEs, and then, we elaborate on our new proposed IANNs method; it mainly contains three parts: firstly, the new ANNs trail solution model is derived, then a joint cost function is proposed, and the corresponding RLM training method is constructed. The method in this section is mainly aimed at solving second-order nonlinear ODEs and also can be extended to other complicated ODEs.

3.1 Conventional neural network method for second-order ODEs

For the BVPs involving the second-order ODEs, consider the following general expression:

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

We select sigmoid as activation function; the ANNs solution (Lagaris et al. 1998) can be cast as:

$$y_t(x,p) = \frac{bA - aB}{b-a} + \frac{B-A}{b-a}x + (x-a)(x-b) \cdot w(1 + \mathrm{e}^{-(kx+\theta)})^{-1} \tag{9}$$

If we change the boundary conditions with $y(a) = A$ and $y'(a) = A'$, the network solution (Lagaris et al. 1998) will be changed into:

$$y_t(x,p) = A - A'a + A'x + (x-a)^2 w(1 + \mathrm{e}^{-(kx+\theta)})^{-1} \tag{10}$$

In both cases, the error function has the following form:

$$E(p) = \sum_{i=1}^{s} \left\{ \frac{\mathrm{d}^2 y_t(x_i,p)}{\mathrm{d}x^2} - f\left[x_i, y_t(x_i,p), \frac{\mathrm{d}y_t(x_i,p)}{\mathrm{d}x}\right] \right\}^2 \tag{11}$$

where s is the number of sample points. Through the analysis of some problems, we find the instability of the simulation results, and the simulation accuracy and efficiency often fall short of expectation. Our new proposed ANNs model improves on it, and a new training method is constructed; the simulation results show that the performances are improved.

3.2 The new ANNs model of trail solution

In the new model, we add variable coefficient vector λ to the second item; then, Eq. (7) becomes the following form:

$$y_t(x,p) = A(x) + \lambda F(x, N(x,p)) \tag{12}$$

We find variable coefficient vector λ can satisfy any condition in any form; the conventional methods fix λ as 1, which is not an optimal choice; the simulation results are relatively unstable and not satisfactory. In this paper, we choose λ as an adjustable parameter, in order to relate it to the neural network model; after many attempts we choose it as $1/k$, where k is the weight vector from input layer to hidden layer.

Then, Eqs. (9) and (10) are changed as:

$$y_t(x,p) = \frac{bA - aB}{b-a} + \frac{B-A}{b-a}x + (x-a)(x-b)\frac{1}{k} \cdot w(1 + \mathrm{e}^{-(kx+\theta)})^{-1} \tag{13}$$

$$y_t(x,p) = A - A'a + A'x + (x-a)^2 \frac{1}{k} \cdot w(1 + \mathrm{e}^{-(kx+\theta)})^{-1} \tag{14}$$

Then, the corresponding expression of error function is simplified; also it reduces the singular value occurrence of weight regulating quantity.

3.3 The joint cost function

Different sample points correspond to different errors $E(x_i)$; especially in most cases for domain $[a,b]$, the error function expression in boundary and domain interval is different. Existing methods ignore this and only use one error function expression, which increases the simulation errors and training complexity. We take n sample points in boundary a and b separately, and m sample points are selected in the interval without domain boundary. The corresponding error function expressions are as follows:

$$E1(p) = \sum_{i=1}^{n} \left\{ \frac{\mathrm{d}^2 y_t(a,p)}{\mathrm{d}x^2} - g\left[a, y_t(a,p), \frac{\mathrm{d}y_t(a,p)}{\mathrm{d}x}\right] \right\} \tag{15}$$

$$E2(p) = \sum_{i=n+1}^{n+m} \left\{ \frac{\mathrm{d}^2 y_t(x_i,p)}{\mathrm{d}x^2} - f\left[x_i, y_t(x_i,p), \frac{\mathrm{d}y_t(x_i,p)}{\mathrm{d}x}\right] \right\} \tag{16}$$

$$E3(p) = \sum_{i=n+m+1}^{2n+m} \left\{ \frac{\mathrm{d}^2 y_t(b,p)}{\mathrm{d}x^2} - s\left[b, y_t(b,p), \frac{\mathrm{d}y_t(b,p)}{\mathrm{d}x}\right] \right\} \tag{17}$$

The joint cost function of system is written as:

$$E(p) = \sum_{i=1}^{n} E1_i^2 + \sum_{i=1}^{m} E2_i^2 + \sum_{i=1}^{n} E3_i^2 \tag{18}$$

3.4 The RLM training algorithm

Here, Eq. (18) may be considered as a complex ANNs system, the minimization is a training process for this network, and the corresponding error with every entry x has to be minimized, which is to update the network parameters (weights and biases). This is an unconstrained optimization problem; here we construct RLM optimization algorithm as the training method. Figure 2 illustrates the flowchart of training process.

The detailed training process of RLM algorithm is as follows:

Step 1 We choose sigmoid function as activation function and initialize the network parameters, the number of neuron in hidden layer is set Hidenums = 5, the maximum iterations Maxcount = 20, network training precision $\varepsilon = 1e{-}4$, adjustment factor $u = 1e{-}3$, decrement of u is $u\_dec = 1e{-}2$, growth rate of mu is $u\_inc = 2$, the maximum of u is 1e10, the minimum of u is $1e-20$, initial network weights and bias are random values in interval $[0, 1]$.

Step 2 The range of $[a, b]$ is uniformly divided to form network input sample points. We repeated take n sample points in boundary a and b, and m sample points are

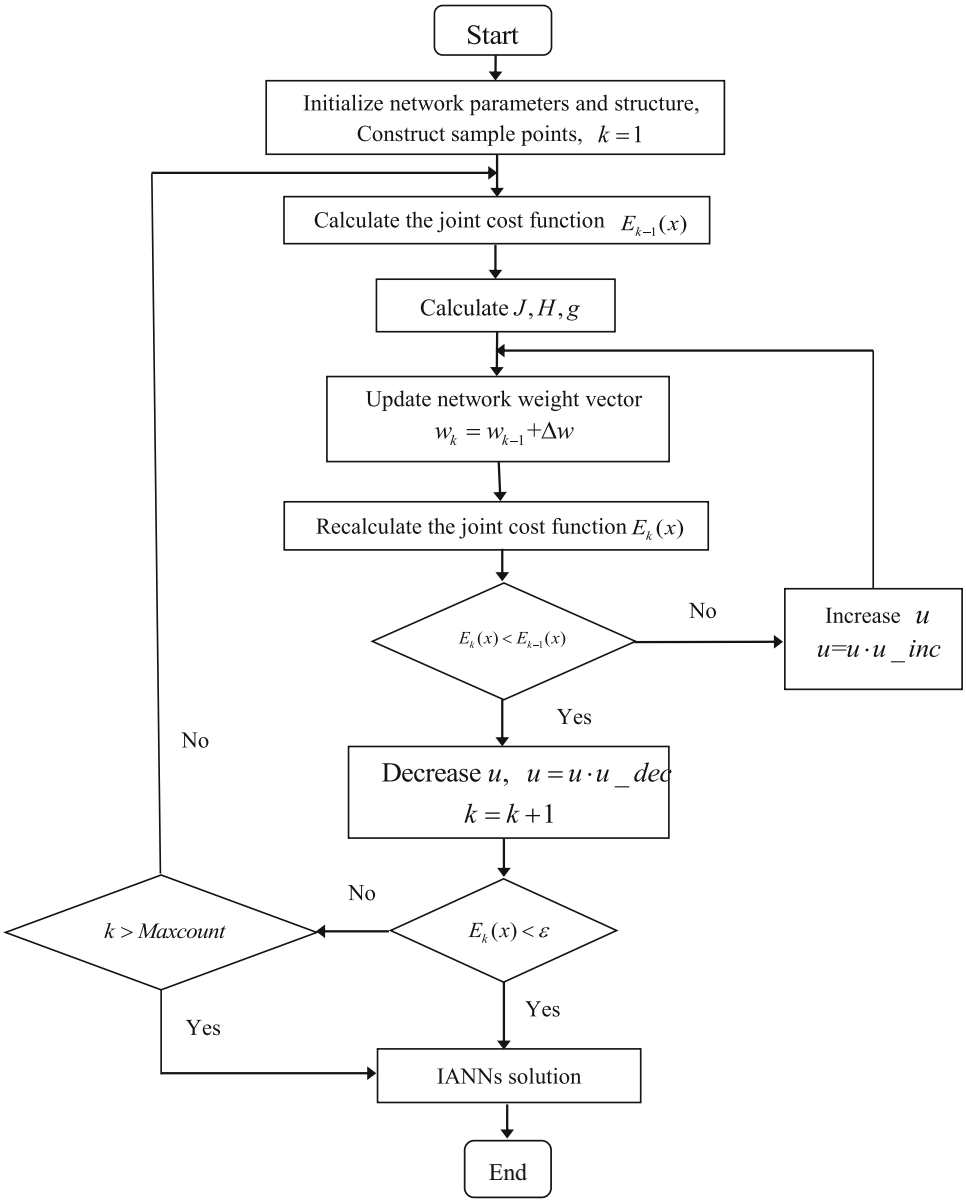

Fig. 2 Flowchart of the RLM training algorithm

selected in the interval (a,b); the total sample points number is $2n+m$.

Step 3 The calculation of network output and error function $\sum_i^n E1(a)$, $\sum_i^m E2(x_i)$, $\sum_i^n E3(b)$. According to Eq. (18), calculate the joint cost function $E_{k-1}(x_i)$.

Step 4 The Jacobian matrix J of joint cost function $E_{k-1}(x_i)$, Hessian matrix H and gradient g are calculated as follows.

The Jacobian matrix J can be expressed as:

$$J=\begin{bmatrix} \frac{\partial E1_1}{\partial k}\frac{\partial E1_2}{\partial k}\cdots\frac{\partial E1_n}{\partial k}\frac{\partial E2_{n+1}}{\partial k}\frac{\partial E2_{n+2}}{\partial k}\cdots\frac{\partial E2_{n+m}}{\partial k}\frac{\partial E3_{n+m+1}}{\partial k}\frac{\partial E3_{n+m+2}}{\partial k}\cdots\frac{\partial E3_{2n+m}}{\partial k} \\ \frac{\partial E1_1}{\partial b}\frac{\partial E1_2}{\partial b}\cdots\frac{\partial E1_n}{\partial b}\frac{\partial E2_{n+1}}{\partial b}\frac{\partial E2_{n+2}}{\partial b}\cdots\frac{\partial E2_{n+m}}{\partial b}\frac{\partial E3_{n+m+1}}{\partial b}\frac{\partial E3_{n+m+2}}{\partial b}\cdots\frac{\partial E3_{2n+m}}{\partial b} \\ \frac{\partial E1_1}{\partial w}\frac{\partial E1_2}{\partial w}\cdots\frac{\partial E1_n}{\partial w}\frac{\partial E2_{n+1}}{\partial w}\frac{\partial E2_{n+2}}{\partial w}\cdots\frac{\partial E2_{n+m}}{\partial w}\frac{\partial E3_{n+m+1}}{\partial w}\frac{\partial E3_{n+m+2}}{\partial w}\cdots\frac{\partial E3_{2n+m}}{\partial w} \end{bmatrix}^{\mathrm{T}} \tag{19}$$

The Hessian matrix H can be expressed as:

$$H=J^{\mathrm{T}}J \tag{20}$$

The gradient g is expressed as:

$$g=J^{\mathrm{T}}[E1_1,E1_2\ldots E1_n,E2_{n+1},E2_{n+2}\ldots E2_{n+m},E2_{n+m+1},E2_{n+m+2}\ldots E_{2n+m}]^{\mathrm{T}} \tag{21}$$

Step 5 Update network parameters (weights and biases) by $w^k=w^{k-1}+\Delta w$ according to Eq. (3), recalculate, and get the new joint cost function $E_k(x_i)$.

Step 6 After the kth iterations, if $E_k(x_i)<\varepsilon$, then break and turn to the step 8.

Step 7 If $E_k(x_i)>E_{k-1}(x_i)$, then $mu=mu\cdot mu\_inc$, and turn to the step 5. Else, $k=k+1$, $\{mu=mu\cdot mu\_dec\}$, if $k>$ Maxcount, then break and turn to the step8, else turn to the step3.

Step 8 Output optimal parameters (weights and biases) and the IANNs solution are obtained.

4 Numerical results

In this section, to demonstrate the applicability and properties of proposed IANNs method, we conduct experiments on the following examples. All of the programs are written with MATLAB R2014a.

Example 1 Consider the following nonlinear second-order ODEs with BVPs:

$$\frac{\mathrm{d}^2y}{\mathrm{d}x^2}=y^3-y\cdot\frac{\mathrm{d}y}{\mathrm{d}x},\quad x\in[1,2]$$
$$y(1)=\frac{1}{2},y(2)=\frac{1}{3}. \tag{22}$$

The corresponding IANNs trail solution is written as:

$$y_t(x,p)=\frac{2}{3}-\frac{1}{6}x+(x^2-3x+2)\cdot\frac{1}{k}w(1+\mathrm{e}^{-(kx+\theta)})^{-1} \tag{23}$$

Then, the first and second derivatives of IANNs trail solution are calculated as:

$$\frac{\mathrm{d}y_t}{\mathrm{d}x}=-\frac{1}{6}+(2x-3)\cdot\frac{1}{k}w(1+\mathrm{e}^{-(kx+\theta)})^{-1}+(x^2-3x+2)\cdot w\mathrm{e}^{-(kx+\theta)}\cdot(1+\mathrm{e}^{-(kx+\theta)})^{-2} \tag{24}$$

$$\frac{\mathrm{d}^2y_t}{\mathrm{d}x^2}=2\frac{1}{k}w\cdot(1+\mathrm{e}^{-(kx+\theta)})^{-1}+2(2x-3)\cdot w\mathrm{e}^{-(kx+\theta)}\cdot(1+\mathrm{e}^{-(kx+\theta)})^{-2}+(x^2-3x+2)\cdot kw\mathrm{e}^{-(kx+\theta)}\cdot(\mathrm{e}^{-(kx+\theta)}-1)\cdot(1+\mathrm{e}^{-(kx+\theta)})^{-3} \tag{25}$$

Different sample points correspond to different error functions; here we repeatedly take 10 sample points at boundary 1 and boundary 2, namely $n=10$, and 99 sample points are selected in domain $(1,2)$, namely $m=99$, and then, it forms the following sample points $[1\ldots1,1.01,1.02\ldots1.98,1.99,2\ldots.2]$; the corresponding error functions $E1(p)$, $E2(p)$, $E3(p)$ are calculated as follows:

$$E1=\sum_{i=1}^{n}\left[2\frac{1}{k}w\cdot(1+\mathrm{e}^{-(k+\theta)})^{-1}-2w\cdot\mathrm{e}^{-(k+\theta)}\cdot(1+\mathrm{e}^{-(k+\theta)})^{-2}-\left(\frac{1}{2}\right)^3-\frac{1}{2}\left(\frac{1}{6}+\frac{1}{k}w\cdot(1+\mathrm{e}^{-(k+\theta)})^{-1}\right)\right] \tag{26}$$

$$
\begin{aligned}
E2 = \sum_{i=n+1}^{n+m} \Bigg[& 2\frac{1}{k} w \cdot (1 + \mathrm{e}^{-(kx_i+\theta)})^{-1} \\
& + 2(2x_i - 3) \cdot w\mathrm{e}^{-(kx_i+\theta)} \cdot (1 + \mathrm{e}^{-(kx_i+\theta)})^{-2} \\
& + (x_i^2 - 3x_i + 2) \cdot kw\mathrm{e}^{-(kx_i+\theta)} \cdot (\mathrm{e}^{-(kx_i+\theta)} - 1) \cdot (1 + \mathrm{e}^{-(kx_i+\theta)})^{-3} \\
& - \left(\frac{2}{3} - \frac{1}{6} x_i + (x_i^2 - 3x_i + 2) \cdot \frac{1}{k} w (1 + \mathrm{e}^{-(kx_i+\theta)})^{-1} \right)^3 \\
& + \left(\frac{2}{3} - \frac{1}{6} x_i + (x_i^2 - 3x_i + 2) \cdot \frac{1}{k} w (1 + \mathrm{e}^{-(kx_i+\theta)})^{-1} \right) \cdot \\
& \left(-\frac{1}{6} + (2x_i - 3) \cdot \frac{1}{k} w (1 + \mathrm{e}^{-(kx_i+\theta)})^{-1} \right. \\
& \left. + (x_i^2 - 3x_i + 2) \cdot w\mathrm{e}^{-(kx_i+\theta)} \cdot (1 + \mathrm{e}^{-(kx_i+\theta)})^{-2} \right) \Bigg]
\end{aligned}
\tag{27}
$$

$$
\begin{aligned}
E3 = \sum_{i=n+m+1}^{2n+m} \Bigg[& 2\frac{1}{k} w \cdot (1 + \mathrm{e}^{-(2k+\theta)})^{-1} \\
& + 2w \cdot \mathrm{e}^{-(2k+\theta)} \cdot (1 + \mathrm{e}^{-(2k+\theta)})^{-2} - \left(\frac{1}{3} \right)^3 \\
& - \frac{1}{3} \left(\frac{1}{6} - \frac{1}{k} w \cdot (1 + \mathrm{e}^{-(2k+\theta)})^{-1} \right) \Bigg]
\end{aligned}
\tag{28}
$$

Then, we get the joint cost function $E=(E1^2 + E2^2 + E3^2)$, and 5 hidden neurons are fixed, and the maximum iterations are set as 20. The absolute errors between the approximate solution obtained by Hilber–Hughes–Taylor-α (HHT-α) method (Attili 2010), Shooting method (Ha 2001) and the new proposed IANNs method are compared in Table 1.

As is shown in Table 1, the proposed method has a better performance than other conventional numerical methods in terms of calculation accuracy, and the calculation error can approximate 1e−5. Only 20 training steps show the high efficiency of the new proposed method. The approximate solution obtained with the proposed method is also compared with the exact solution in Fig. 3, and Fig. 4 shows the calculation error of the proposed IANNs method.

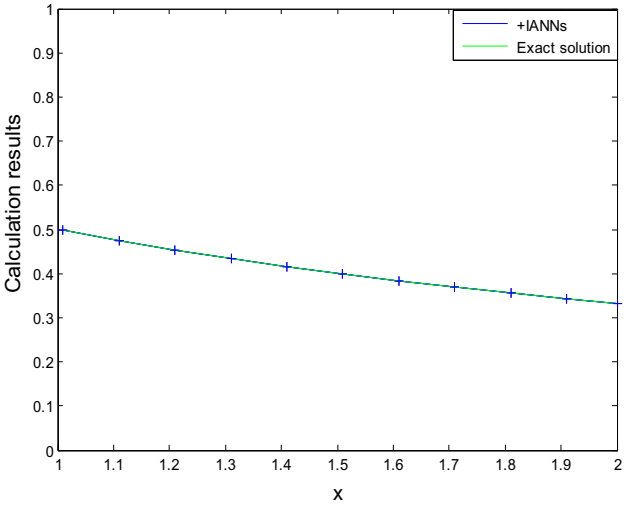

Fig. 3 Comparison of exact solution and IANNs method

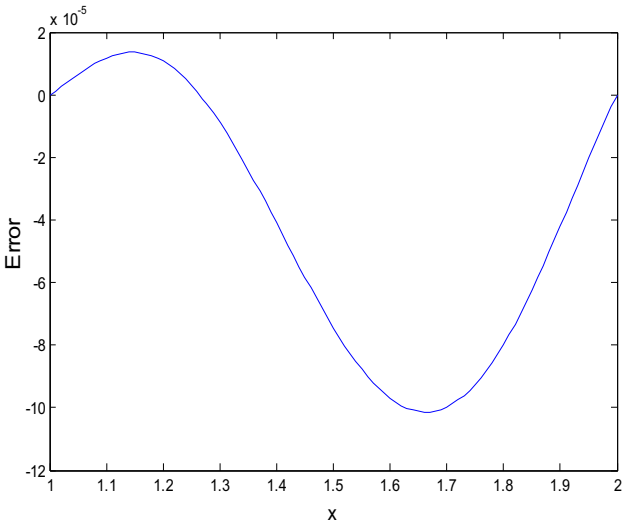

Fig. 4 Error plot between exact solution and IANNs results

Table 1 Comparison of the absolute errors on example 1

x	IANNs	HHT-α	Shooting method	x	IANNs	HHT-α	Shooting method
1.00	0	0	0	1.50	0.0000206	0.00053	0.000109
1.05	0.0000045	0.00008	0.000030	1.55	0.0000261	0.00038	0.000103
1.10	0.0000090	0.00013	0.000054	1.60	0.0000299	0.00036	0.000096
1.15	0.0000119	0.00025	0.000073	1.65	0.0000316	0.00018	0.000088
1.20	0.0000123	0.00028	0.000088	1.70	0.0000312	0.00019	0.000078
1.25	0.0000102	0.00026	0.000099	1.75	0.0000287	0.00024	0.000066
1.30	0.0000060	0.00043	0.000107	1.80	0.0000242	0.00033	0.000053
1.35	0.0000002	0.00046	0.000111	1.85	0.0000183	0.00011	0.000040
1.40	0.0000069	0.00056	0.000113	1.90	0.0000117	0.00009	0.000025
1.45	0.0000140	0.00041	0.000112	1.95	0.0000052	0.00013	0.000009
				2.00	0.0000000	0.00008	0.000007

Table 2 Comparison of the IANNs results we obtained with exact solution

Sample x	1.0050	1.1050	1.2050	1.3050	1.4050	1.5050	1.6050	1.7050	1.8050	1.9050
Exact solution	0.4988	0.4751	0.4535	0.4338	0.4158	0.3992	0.3838	0.3697	0.3565	0.3442
IANN method	0.4988	0.4751	0.4535	0.4338	0.4158	0.3992	0.3837	0.3697	0.3565	0.3442

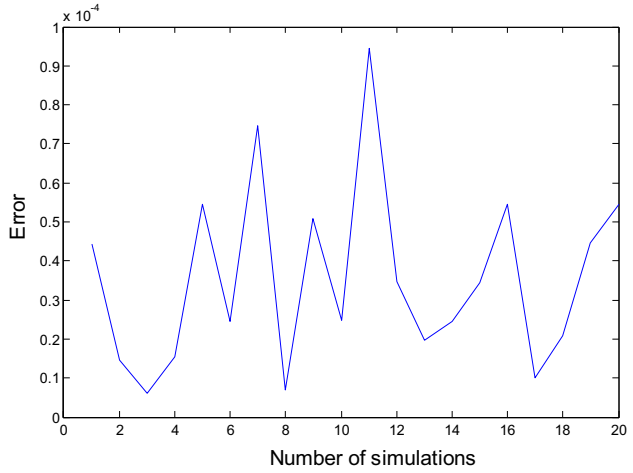

Fig. 5 Corresponding simulation accuracy to each simulation

Figures 3 and 4 show that the calculation results of IANNs method are in good accordance with exact solution.

For testing the simulation accuracy, some sample points not taken during training are randomly selected in domain $[1, 2]$, and the simulation results are listed in Table 2.

In order to illustrate the stability of high simulation accuracy, we did the simulation 20 times with the proposed IANNs method; Fig. 5 shows the accuracy of each simulation results.

Then, a set of weights and biases vectors after the completion of training is listed in Table 3.

Substituting these network parameters (weights and biases) into Eq. (23), we get the approximate analytic expression of the problem.

Example 2 Let us take a nonlinear singular IVPs of Lane–Emden type equation:

$$\begin{aligned} &\frac{\mathrm{d}^2y}{\mathrm{d}x^2}+\frac{2}{x}\frac{\mathrm{d}y}{\mathrm{d}x}+4(2e^y+e^{y/2}), \quad x\in[0,1] \\ &y(0)=0, \left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_{x=0}=0. \end{aligned} \tag{29}$$

The IANNs trail solution is defined as:

$$y_t(x,p)=x^2\cdot\frac{1}{k}w(1+\mathrm{e}^{-(kx+\theta)})^{-1} \tag{30}$$

Then, we calculate the first and second derivatives, and the error function E is written as follows:

$$\begin{aligned} E=\sum_{i=1}^{101}\Big[&2\frac{1}{k}w\cdot(1+\mathrm{e}^{-(kx_i+\theta)})^{-1} \\ &+4x_i\cdot w\mathrm{e}^{-(kx_i+\theta)}\cdot(1+\mathrm{e}^{-(kx_i+\theta)})^{-2} \\ &+\ x_i^2\cdot kw\mathrm{e}^{-(kx_i+\theta)}\cdot(\mathrm{e}^{-(kx_i+\theta)}-1)\cdot(1+\mathrm{e}^{-(kx_i+\theta)})^{-3} \\ &+4\cdot\frac{1}{k}w(1+\mathrm{e}^{-(kx_i+\theta)})^{-1} \\ &+\ 2x_i\cdot w\mathrm{e}^{-(kx_i+\theta)}\cdot(1+\mathrm{e}^{-(kx_i+\theta)})^{-2} \\ &+8\cdot\mathrm{e}^{x_i^2\cdot\frac{1}{k}w(1+\mathrm{e}^{-(kx_i+\theta)})^{-1}}+4\cdot\mathrm{e}^{\frac{1}{2}x_i^2\cdot\frac{1}{k}w(1+e^{-(kx_i+\theta)})^{-1}}\Big] \end{aligned} \tag{31}$$

Following the procedure of the proposed RLM training method, the maximum iterations is set as Maxcount = 50, the network system of error function E has been trained here with 100 equidistant points in domain $[0, 1]$, and then, the simulation error can reach 1e−4.

Also we calculate it with variational iteration (VI) method (Yildirim and Ozis 2009); the variational iteration formula is as follows:

$$\begin{aligned} y_{n+1}(x)=y_n(x)+\int_0^x\left[\frac{s^2}{x}-s\right] \\ \times\left[(y_n)_{ss}+\frac{2}{s}(y_n)_s+4(2e^y+e^{y/2})\right]\mathrm{d}s, \quad n\geq 0. \end{aligned} \tag{32}$$

After 50 iterations of Eq. (32), we get the approximate solution. The comparison of calculation results among exact solution, VI method, Legendre neural networks (LeNN) (Mall and Chakraverty 2016) method and our proposed IANNs method is listed in Table 4.

Table 4 shows the proposed IANNs method has significantly higher calculation accuracy compared with other

Table 3 A set of optimal weights and biases vectors of IANNs method

w	9.35156812	− 2.938697392	− 1.894263053	− 6.437596474	− 0.07821478961
k	− 7.215607923	− 14.18225686	9.382962491	3.168220386	− 0.01903155258
θ	− 0.8673713591	2.653574991	4.690448322	2.407492293	0.2674912657

Table 4 Comparison of calculation results among exact solution, IANNs, VI and LeNN method

x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Exact	0	− 0.0199	− 0.0784	− 0.1724	− 0.2968	− 0.4463	− 0.6150	− 0.7976	− 0.9894	− 1.1867	− 1.3863
IANNs	0	− 0.0199	− 0.0784	− 0.1724	− 0.2966	− 0.4459	− 0.6145	− 0.7971	− 0.9890	− 1.1862	− 1.3859
VI	0	− 0.0199	− 0.0784	− 0.1724	− 0.2968	− 0.4463	− 0.6150	− 0.7976	− 0.9894	− 1.1867	− 1.3665
LeNN	0	− 0.0195	− 0.0785	− 0.1725	− 0.2965	− 0.4468	− 0.6135	− 0.7975	− 0.9896	− 1.1869	− 1.3861

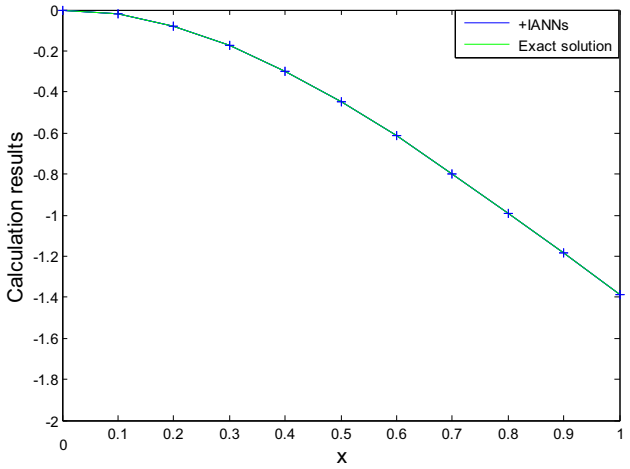

Fig. 6 Comparison of exact solution and IANNs method

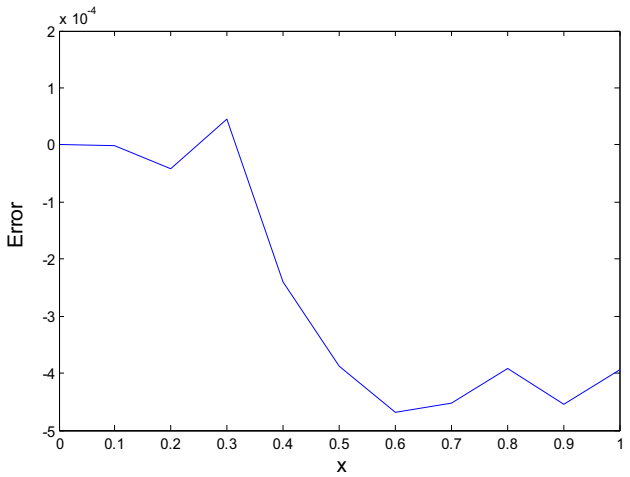

Fig. 7 Error plot between exact solution and IANNs results

two methods. Also the conventional numerical VI method needs more tedious calculation steps and longer calculation time; the calculation results fluctuate greatly in domain. The comparison of the calculation results between exact solution and IANNs method is depicted in Fig. 6, and Fig. 7 shows the calculation error of the proposed IANNs method.

Figures 6 and 7 show that the simulation results of IANNs method are in good accordance with exact solution.

Table 5 lists a set of weights and biases vectors after the completion of training.

Substituting these network parameters (weights and biases) into Eq. (30), we get the approximate analytic expression of the problem.

Example 3 To exhibit the applicability of this method to more difficult problems, consider the following Troesch's problem (Troesch 1976):

Table 6 Numerical results for Troesch's problem ($\lambda = 1$)

x	Exact solution	IANNs	MHPM	DM
0.1	0.0846612565	0.0846607294	0.0843817004	0.084248760
0.2	0.1701713582	0.1701680092	0.1696207644	0.169430700
0.3	0.2573939080	0.2573898029	0.2565929224	0.256414500
0.4	0.3472228551	0.3472227479	0.3462107378	0.346085720
0.5	0.4405998351	0.4406059387	0.4394422743	0.439401985
0.6	0.5385343980	0.5385434519	0.5373300622	0.537365700
0.7	0.6421286091	0.6421340410	0.6410104651	0.641083800
0.8	0.7526080939	0.75260639935	0.7517335467	0.751788000
0.9	0.8713625196	0.8713585345	0.8708835371	0.870908700

Table 5 A set of optimal weight and biases vectors of IANNs method

w	0.3722438642	− 5.0214668020	14.923516070	5.2521280960	− 9.5719673000
k	− 1.7223891710	− 4.5510970400	− 0.6162264999	1.3765213420	− 7.7324878180
θ	− 7.6864525590	− 0.3671821657	− 2.1793657240	− 12.785647110	− 5.4617392440

Table 7 Numerical results for Troesch's problem ($\lambda = 10$)

x	Exact solution	IANNs	VIM	MHPM	DM
0.1	0.0000763	0.1000000001	0.1186109866	17.617500000	667,081.18744
0.2	0.0001299	0.2000000058	0.4461962517	33.693333333	1,333,955.1189
0.3	0.0003589	0.5166517565	3.8003366781	46.785833333	1,999,860.1189
0.4	0.0009779	0.6476034148	79.891472730	55.653333333	2,661,970.7366
0.5	0.0026590	0.7579202238	1880.3539472	59.354166667	3,310,585.4201
0.6	0.0072289	0.8476034148	41642.365193	57.346666667	3,914,127.8659
0.7	0.0196640	0.9166529879	878764.64189	49.589166667	4,374,578.5342
0.8	0.0537303	0.9650689432	18064027.967	36.640000000	4,406,724.4178
0.9	0.1521140	0.9928512806	366613074.02	19.757500000	3,290,268.6374

$$\begin{aligned} &\frac{\mathrm{d}^2y}{\mathrm{d}x^2} - \lambda\sinh(\lambda y) = 0, \quad x \in [0,1] \\ &y(0) = 0, y(1) = 1 \end{aligned} \tag{33}$$

Troesch's problem arises from a system of nonlinear ODEs which occur in an investigation of the confinement of a plasma column by radiation pressure. Also, this problem is recommended by Boyd (2011) as a test problem in order to show efficiency of any numerical method.

The IANNs trail solution is defined as:

$$y_t(x,p) = x + x(x-1)\cdot\frac{1}{k}w(1+\mathrm{e}^{-(kx+\theta)})^{-1} \tag{34}$$

Here we repeatedly take 10 sample points at boundary 0 and boundary 1, namely $n = 10$, and 99 sample points are selected in domain $(1,2)$, namely $m = 99$, and then, it forms the following sample points $[0\ldots.0, 0.01, 0.02\ldots.0.98, 0.99, 1\ldots.1]$; the corresponding error functions $E1(p)$, $E2(p)$,$E3(p)$ are as follows:

$$E1 = \sum_{i=1}^{n}\left[2\frac{1}{k}w\cdot(1+\mathrm{e}^{-\theta})^{-1} - 2w\mathrm{e}^{-\theta}\cdot(1+\mathrm{e}^{-\theta})^{-2} - \lambda\sinh(0)\right] \tag{35}$$

$$\begin{aligned} E2 = \sum_{i=n+1}^{n+m}\Bigg[&2\frac{1}{k}w\cdot(1+\mathrm{e}^{-(kx_i+\theta)})^{-1} \\ &+2(2x_i-1)\cdot w\mathrm{e}^{-(kx_i+\theta)}\cdot(1+\mathrm{e}^{-(kx_i+\theta)})^{-2} \\ &+(x_i^2-x_i)\cdot kw\mathrm{e}^{-(kx_i+\theta)}\cdot(\mathrm{e}^{-(kx_i+\theta)}-1)\cdot(1+\mathrm{e}^{-(kx_i+\theta)})^{-3} \\ &-\lambda\sinh\left(\lambda\cdot(x_i+x_i(x_i-1)\cdot\frac{1}{k}w(1+\mathrm{e}^{-(kx_i+\theta)})^{-1}\right)\Bigg] \end{aligned} \tag{36}$$

$$\begin{aligned} E3 = \sum_{i=n+m+1}^{2n+m}\Big[&2\frac{1}{k}w\cdot(1+\mathrm{e}^{-(k+\theta)})^{-1} \\ &+2\cdot w\mathrm{e}^{-(k+\theta)}\cdot(1+\mathrm{e}^{-(k+\theta)})^{-2} - \lambda\sinh(\lambda)\Big] \end{aligned} \tag{37}$$

Then, we get the joint cost function $E = E1^2 + E2^2 + E3^2$; the maximum iterations is set as Maxcount = 50. For $\lambda = 1$ and $\lambda = 10$, following the procedure of the proposed RLM training method, we get the optimal network parameters (weights and biases). The comparison of calculation results among exact solution (Mickens 1994), decomposition method (DM) (Deeba et al. 2000), variational iteration method (VIM) (Momani et al. 2006), modified homotopy perturbed method (MHPM) (Feng et al. 2007) and our proposed IANNs method is listed in Tables 6 and 7.

Tables 6 and 7 show that the proposed IANNs method has higher calculation accuracy in solving Troesch's problem compared with other methods. In view of computational complexity, for different λ, we can directly use the same joint cost function model, and the process of the new proposed method is more simple than other methods.

5 Conclusion

In this paper, the IANNs method and RLM training algorithm have been developed for solving ODEs with initial values and boundary values. The calculation accuracy and efficiency of this method have been examined by representative examples and compared with some previously proposed approaches. It has been shown that the proposed approximation method has higher calculation accuracy and efficiency, and the simulation results are always stable at a high precision value. This article from the model construction to the training algorithm reconstruction has carried on the detailed elaboration; the method developed in this paper can be used as a paradigm for many other numerical applications.

Acknowledgements The work was supported by the National Natural Science Foundation of China (Grant No. 51475086), CASTBISEE2019-019, and the Fundamental Research Funds for the Central Universities (Grant No. N162312001).

Compliance with ethical standards

Conflict of interest Author Shangjie Li declares that he has no conflict of interest. Author Xingang Wang declares that he has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

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

References

Attili BS (2010) The Hilber–Hughes–Taylor-α (HHT-α) method compared with an implicit Runge–Kutta for second-order systems. Int J Comput Math 87(8):1755–1767

Boyd JP (2011) One-point pseudospectral collocation for the one-dimensional Bratu equation. Appl Math Comput 217(12):5553–5565

Chakraverty S, Mall S (2014) Regression-based weight generation algorithm in neural network for solution of initial and boundary value problems. Neural Comput Appl 25(3):585–594

Deeba E, Khuri SA, Xie S (2000) An algorithm for solving boundary value problems. J Comput Phys 159(2):125–138

Douglas J, Jones BF (1963) On predictor–corrector methods for nonlinear parabolic differential equations. J Soc Ind Appl Math 11(1):195–204

Effati S, Pakdaman M (2010) Artificial neural network approach for solving fuzzy differential equations. Inf Sci 180(8):1434–1457

Feng X, Mei L, He G (2007) An efficient algorithm for solving Troesch's problem. Appl Math Comput 189(1):500–507

Ha SN (2001) A nonlinear shooting method for two-point boundary value problems. Comput Math Appl 42(11–12):1411–1420

Hagan MT, Menhaj MB (1994) Training feed-forward networks with the Marquardt algorithm. IEEE Trans Neural Netw 5(6):989–993

Hoda SA, Nagla HA (2011) Neural network methods for mixed boundary value problems. Int J Nonlinear Sci 11(3):312–316

Hornik K (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366

Hou C, Simos TE, Famelis IT (2020) Neural network solution of pantograph type differential equations. Math Methods Appl Sci 43(6):3369–3374

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

Lagaris IE, Likas A, Papageorgiou DG (2000) Neural-network methods for boundary value problems with irregular boundaries. IEEE Trans Neural Netw 11(5):1041–1049

Lee H, Kang IS (1990) Neural algorithm for solving differential equations. J Comput Phys 91(1):110–131

Lu Y, Yin Q, Li H, Sun H, Yang Y, Hou M (2019) The LS-SVM algorithms for boundary value problems of high-order ordinary differential equations. Adv Differ Equ 1:1–22

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 (2013) Regression-based neural network training for the solution of ordinary differential equations. Int J Math Model Numer Optim 4(2):136–149

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

Mcfall K, Mahan JR (2009) Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. IEEE Trans Neural Netw 20(8):1221–1233

Mehrkanoon S, Falck T, Suykens JA (2012) Approximate solutions to ordinary differential equations using least squares support vector machines. IEEE Trans Neural Netw 23(9):1356–1367

Mickens RE (1994) Nonstandard finite difference models of differential equations. World Scientific, Singapore

Momani S, Abuasad S, Odibat Z (2006) Variational iteration method for solving nonlinear boundary value problems. Appl Math Comput 183(2):1351–1358

Mosleh M (2013) Fuzzy neural network for solving a system of fuzzy differential equations. Appl Soft Comput 13(8):3597–3607

Otadi M (2019) Simulation and evaluation of second-order fuzzy boundary value problems. Soft Comput 23(20):10463–10475

Panghal S, Kumar M (2020) Optimization free neural network approach for solving ordinary and partial differential equations. Eng Comput. https://doi.org/10.1007/s00366-020-00985-1

Ricardo H (2009) A modern introduction to differential equations. Academic Press, Elsevier

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

Sneddon IN (2006) Elements of partial differential equations. Courier Corporation, Chelmsford

Troesch BA (1976) A simple approach to a sensitive two-point boundary value problem. J Comput Phys 21(3):279–290

Wambecq A (1978) Rational Runge–Kutta methods for solving systems of ordinary differential equations. Computing 20(4):333–342

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 Diff Equ 1:1–24

Yang Y, Hou M, Luo J, Tian Z (2020) Numerical solution of several kinds of differential equations using block neural network method with improved extreme learning machine algorithm. J Intell Fuzzy Syst 38(3):3445–3461

Yazdi HS, Pourreza R (2010) Unsupervised adaptive neural-fuzzy inference system for solving differential equations. Appl Soft Comput 10(1):267–275

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

Yazdi HS, Modaghegh H, Pakdaman M (2012) Ordinary differential equations solution in kernel space. Neural Comput Appl 21(1):79–85

Yildirim A, Ozis T (2009) Solutions of singular IVPs of Lane–Emden type by the variational iteration method. Nonlinear Anal theory Methods Appl 70(6):2480–2484