Hindawi Publishing Corporation
Advances in Artificial Neural Systems
Volume 2013, Article ID 181895, 12 pages
http://dx.doi.org/10.1155/2013/181895

Research Article

Comparison of Artificial Neural Network Architecture in Solving Ordinary Differential Equations

Susmita Mall and S. Chakraverty

Department of Mathematics, National Institute of Technology, Rourkela, Odisha-769008, India

Correspondence should be addressed to S. Chakraverty; sne_chak@yahoo.com

Received 8 August 2013; Revised 31 October 2013; Accepted 31 October 2013

Academic Editor: Ping Feng Pai

This paper investigates the solution of Ordinary Differential Equations (ODEs) with initial conditions using Regression Based Algorithm (RBA) and compares the results with arbitrary- and regression-based initial weights for different numbers of nodes in hidden layer. Here, we have used feed forward neural network and error back propagation method for minimizing the error function and for the modification of the parameters (weights and biases). Initial weights are taken as combination of random as well as by the proposed regression based model. We present the method for solving a variety of problems and the results are compared. Here, the number of nodes in hidden layer has been fixed according to the degree of polynomial in the regression fitting. For this, the input and output data are fitted first with various degree polynomials using regression analysis and the coefficients involved are taken as initial weights to start with the neural training. Fixing of the hidden nodes depends upon the degree of the polynomial. For the example problems, the analytical results have been compared with neural results with arbitrary and regression based weights with four, five, and six nodes in hidden layer and are found to be in good agreement.

1. Introduction

Differential equations play vital role in various fields of engineering and science. The exact solution of differential equations may not be always possible [1]. So various types of well known numerical methods such as Euler, Runge-kutta, Predictor-Corrector, finite element, and finite difference methods, are used for solving these equations. Although these numerical methods provide good approximations to the solution, but these may be challenging for higher dimension problems. In recent years, many researchers tried to find new methods for solving differential equations. As such here Artificial Neural Network (ANN) based models are used to solve ordinary differential equations with initial conditions.

Lee and Kang [2] first introduced a method to solve first order differential equation using Hopfield neural network models. Then, another approach by Meade and Fernandez [3, 4] has been proposed for both linear and nonlinear differential equations using B_1-splines and feed forward neural network. Artificial neural networks based on Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization technique for solving ordinary and partial differential equations have been excellently presented by Lagaris et al. [5]. Also Lagaris et al. [6] investigated neural network methods for boundary value problems with irregular boundaries. Parisi et al. [7] presented unsupervised feed forward neural network for the solution of differential equations. The potential of the hybrid and optimization technique to deal with differential equation of lower order as well as higher order has been presented by Malek and Shekari Beidokhti [8]. Choi and Lee [9] discussed comparison of generalizing ability on solving differential equation using back propagation and reformulated radial basis function network. Yazdi et al. [10] used unsupervised kernel least mean square algorithm for solving ordinary differential equations. A new algorithm for solving matrix Riccati differential equations has been developed by Selvaraju and Abdul Samant [11]. He et al. [12] investigated a class of partial differential equations using multilayer neural network. Kumar and Yadav [13] surveyed multilayer perceptrons and radial basis function neural network methods for the solution of differential equations. Tsoulos et al. [14] solved differential equations with neural networks using

a scheme based on grammatical evolution. Numerical solution of elliptic partial differential equation using radial basis function neural networks has been presented by Jianyu et al. [15]. Shirvany et al. [16] proposed multilayer perceptron and radial basis function (RBF) neural networks with a new unsupervised training method for numerical solution of partial differential equations. Mai-Duy and Tran-Cong [17] discussed numerical solution of differential equations using multiquadric radial basis function networks. Fuzzy linguistic model in neural network to solve differential equations is presented by Leephakpreeda [18]. Franke and Schaback [19] solved partial differential equations by collocation using radial basis functions. Smaoui and Al-Enezi [20] presented the dynamics of two nonlinear partial differential equations using artificial neural networks. Differential equations with genetic programming have been analyzed by Tsoulos and Lagaris [21]. McFall and Mahan [22] used artificial neural network for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. Hoda and Nagla [23] solved mixed boundary value problems using multilayer perceptron neural network method.

As per the review of the literatures, it reveals that authors have taken the parameters (weights/biases) as arbitrary (random) and the numbers of nodes in hidden layer are considered by trial and error method. In this paper, we propose a method for solving ordinary differential equations using feed forward neural network as a basic approximation element and error back propagation algorithm [24, 25] by fixing hidden nodes as per the required accuracy. The trial solution of the model is generated by training the algorithm. The approximate solution by ANN has many benefits compared with traditional numerical methods. The ANN trial solution is written as sum of two terms, first one satisfies initial/boundary conditions and the second part involves regression based neural network with adjustable parameters. The computational complexity does not increase considerably with the number of sampling points. The method is general so it can be applied to solve linear and nonlinear ordinary and partial differential equations. The modification of parameters has been done without direct use of optimization technique. For which computation of the gradient of error with respect to the network parameters is required. A regression based artificial neural network with combinations of initial weights (arbitrary and regression based) in the connections is first proposed by Chakraverty et al. [26] and then by Singh et al. [27]. Here, number of nodes in hidden layer may be fixed according to the degree of polynomial required for the accuracy. We have considered a first order and an application problem such as damped free vibration problem to show the comparison of different ANN models. Mall and Chakraverty [28] proposed regression-based neural network model for solving ordinary differential equations.

Rest of the paper is organized as follows. In Section 2, we describe the general formulation of the proposed approach and computation of gradient of the error function. Section 3 gives details of problem formulation and construction of the appropriate form of trial solution. The proposed regression based artificial neural network method has been presented in Section 4. Numerical examples and its results are presented in Section 5. In this section, we compare arbitrary and regression based weight results and those are shown graphically. Section 6 incorporates the discussion and analysis part. Lastly conclusion is outlined in Section 7.

2. General Formulation for Differential Equations

Let us consider the following general differential equations which represent both ordinary and partial differential equations [4]:

$$G\left(x, \psi(x), \nabla\psi(x), \nabla^2\psi(x) \cdots\right) = 0, \quad x \in D, \tag{1}$$

subject to some initial or boundary conditions, where $x = (x_1, x_2, \ldots, x_n) \in R^n$, $D \subset R^n$ denotes the domain, and $\psi(x)$ is the solution to be computed. Here, G is the function which defines the structure of the differential equation and ∇ is a differential operator. For the solution of the differential equation, a discretized domain $\overline{D}$ over finite set of points in D is considered. Thus, the problem transformed into the system of equations as follows:

$$G\left(x_i, \psi(x_i), \nabla\psi(x_i), \nabla^2\psi(x_i) \cdots\right) = 0, \quad x \in \overline{D}. \tag{2}$$

Let $\psi_t(x, p)$ denote the trail solution with adjustable parameters (weights, biases) p, and then the problem may be formulated as

$$G\left(x_i, \psi_t(x_i, p), \nabla\psi_t(x_i, p), \ldots, \nabla^m\psi_t(x_i, p) \cdots\right) = 0. \tag{3}$$

Corresponding error function with respect to every input data is written as

$$\min_p \sum_{x_i \in \overline{D}} \left(G\left(x_i, \psi_t(x_i, p), \nabla\psi_t(x_i, p), \ldots, \nabla^m\psi_t(x_i, p)\right)\right)^2. \tag{4}$$

Now, $\psi_t(x, p)$ may be written as the sum of two terms

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

where $A(x)$ satisfies initial or boundary condition and contains no adjustable parameters, whereas $N(x, p)$ is the output of feed forward neural network with the parameters p and input data x. The second term $F(x, N(x, p))$ makes no contribution to initial or boundary but this is used to a neural network model whose weights and biases are adjusted to minimize the error function.

2.1. Computation of the Gradient. The error computation not only involves the outputs but also the derivatives of the network output with respect to its inputs. So, it requires finding out the gradient of the network derivatives with respect to its inputs. Let us now consider a multilayered perceptron with one input node, a hidden layer with m nodes (fixed number of nods as proposed), and one output unit. For the given inputs $x = (x_1, x_2, \ldots x_n)$, the output is given by

$$N(x, p) = \sum_{j=1}^{m} v_j \sigma\left(z_j\right), \tag{6}$$

where $z_j = \sum_{i=1}^{n} w_{ji}x_i + u_j$, w_{ji} denotes the weight from input unit i to the hidden unit j, v_j denotes weight from the hidden unit j to the output unit, u_j denotes the biases, and $\sigma(z_j)$ is the sigmoid activation function.

The derivatives of $N(x, p)$ with respect to input x_i is

$$\frac{\partial^k N}{\partial x_i^k} = \sum_{j=1}^{m} v_j w_{ji}^k \sigma_j^{(k)}, \tag{7}$$

where $\sigma = \sigma(z_j)$ and $\sigma^{(k)}$ denotes the kth order derivative of sigmoid function.

Let N_ϑ denote the derivative of the network with respect to its inputs and then we have the following relation [4]:

$$N_\vartheta = D^n N = \sum_{i=1}^{n} v_i P_i \sigma_i^{(n)}, \tag{8}$$

where

$$P_j = \prod_{k=1}^{n} w_{jk}^{\lambda_k}, \qquad \kappa = \sum_{i=1}^{n} \lambda_i. \tag{9}$$

The derivative of N_ϑ with respect to other parameters may be obtained as

$$\frac{\partial N_\vartheta}{\partial v_j} = P_j \sigma_j^{(\kappa)}, \tag{10}$$

$$\frac{\partial N_\vartheta}{\partial u_j} = v_j P_j \sigma_j^{(\kappa+1)}, \tag{11}$$

$$\frac{\partial N_\vartheta}{\partial w_{ji}} = x_i v_j P_j \sigma_j^{(\kappa+1)} + v_j \lambda_i w_{ji}^{\lambda_i - 1} \left(\prod_{k=1, k\neq i} w_{ji}^{\lambda_k} \right) \sigma_j^{(\kappa)}. \tag{12}$$

3. Formulation of First Order Ordinary Differential Equation

Let us consider first order ordinary differential equation as below

$$\frac{d\psi}{dx} = f(x, \psi), \quad x \in [a, b], \tag{13}$$

with initial condition $\psi(a) = A$.

In this case, the ANN trail solution may be written as

$$\psi_t(x, p) = A + (x - a) N(x, p), \tag{14}$$

where $N(x, p)$ is the neural output of the feed forward network with one input data x with parameters p. The trial solution $\psi_t(x, p)$ satisfies the initial condition. We differentiate the trial solution $\Psi_t(x, p)$ to get

$$\frac{d\psi_t(x, p)}{dx} = N(x, p) + (x - a) \frac{dN(x, p)}{dx}. \tag{15}$$

For evaluating the derivative term in the right hand side of (15), we use (5)–(11).

The error function for this case may be formulated as

$$E(p) = \sum_{i=1}^{n} \left(\frac{d\psi_t(x_i, p)}{dx} - f(x_i, \psi_t(x_i, p)) \right)^2. \tag{16}$$

The weights from input to hidden are modified according to the following rule

$$w_{ji}^{r+1} = w_{ji}^r - \eta \left(\frac{\partial E}{\partial w_{ji}^r} \right), \tag{17}$$

where

$$\frac{\partial E}{\partial w_{ji}^r} = \frac{\partial}{\partial w_{ji}^r} \left(\sum_{i=1}^{n} \left(\frac{d\psi_t(x_i, p)}{dx} - f(x_i, \psi_t(x_i, p)) \right)^2 \right). \tag{18}$$

Here, η is the learning rate and r is the iteration step. The weights from hidden to output layer may be updated in a similar formulation as done for input to hidden.

3.1. Formulation of Second Order Ordinary Differential Equation. In this case, the second order ordinary differential equation may be written in general as

$$\frac{d^2\psi}{dx^2} = f\left(x, \psi, \frac{d\psi}{dx}\right), \quad x \in [a, b], \tag{19}$$

with initial conditions $\psi(a) = A$, $\psi'(a) = A'$.

The ANN trail solution may be discussed as

$$\psi_t(x, p) = A + A'(x - a) + (x - a)^2 N(x, p), \tag{20}$$

where $N(x, p)$ is the neural output of the feed forward network with one input data x with parameters p and the trial solution $\psi_t(x, p)$ satisfies the initial conditions.

The error function to be minimized for second order ordinary differential equation will be

$$E(p) = \sum_{i=1}^{n} \left(\frac{d^2\psi_t(x_i, p)}{dx^2} - f\left(x_i, \psi_t(x_i, p), \frac{d\psi}{dx}\right) \right)^2. \tag{21}$$

Next, the following weight updating rule is applied for weights from input to hidden connections:

$$w_{ji}^{r+1} = w_{ji}^r - \eta \left(\frac{\partial E}{\partial w_{ji}^r} \right), \tag{22}$$

where

$$\frac{\partial E}{\partial w_{ji}^r} = \frac{\partial}{\partial w_{ji}^r} \left(\sum_{i=1}^{n} \left(\frac{d^2\psi_t(x_i, p)}{dx^2} - f\left[x_i, \psi_t(x_i, p), \frac{d\psi}{dx}\right] \right)^2 \right). \tag{23}$$

Again, we update the weights from hidden to output layer, as discussed for input to hidden.

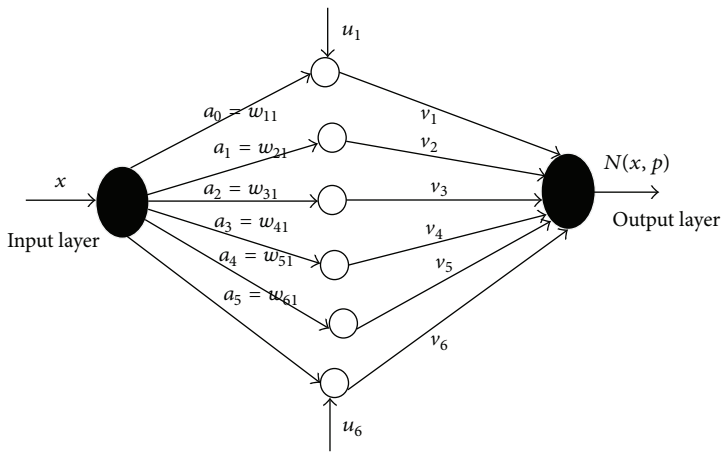

Figure 1: Three-layered neural network architecture with single input and single output node.

4. Proposed Regression-Based Algorithm

Three layer architecture of ANN for the present problem is considered. Usually numbers of nodes in the hidden layer are taken by trial and error method. Here, we fix the number of nodes in hidden layer by using regression-based weight generation [24, 25]. Figure 1 shows the proposed model, in which the input layer consist of single input unit and the output layer consist of one output unit. Numbers of nodes in the hidden layer are fixed according to degree of polynomial to be considered. If nth degree polynomial is considered, then the number of nodes in hidden layer will be $n + 1$ and coefficients (constants) of the polynomial may be considered as initial weights from input to hidden as well as hidden to output layers or any combination of random and regression based weight. Network architecture with five degree polynomial has been shown in Figure 1, the six coefficients (constants) are taken as initial weights in two stages from input to hidden and hidden to output layer. The constants of the polynomial, that is, a_i are taken as initial weights and six nodes for the six constants in the hidden layer are considered.

5. Numerical Examples

In this section, we present solution of two example problems as mentioned earlier. In all cases, we have used error back propagation algorithm and one hidden layer. The weights are taken as arbitrary and regression based for comparison of the training method. Sigmoid function $\sigma(x) = 1/(1 + e^{-x})$ is considered as an activation function for hidden unit.

Example 1. Let us consider the first order ordinary differential equation as follows:

$$\frac{d\psi}{dx} + \left(x + \frac{1 + 3x^2}{1 + x + x^3}\right)\psi = x^3 + 2x + x^2\left(\frac{1 + 3x^2}{1 + x + x^3}\right), \quad x \in [0, 1], \tag{24}$$

with initial condition $\psi(0) = 1$.

The trial solution is written as

$$\psi_t(x, p) = 1 + xN(x, p). \tag{25}$$

We have trained the network for 20 equidistant points in [0, 1] and compared results between analytical and neural with arbitrary and regression based weights with four, five, and six nodes fixed in hidden layer. Comparison between analytical and neural results with arbitrary and regression based weights is given in Table 1. Analytic results are incorporated in second column. Neural results for arbitrary weights $w(A)$ (from input to hidden layer) and $v(A)$ (from hidden to output layer) with four, five, and six nodes are cited in third, fifth, and seventh column, respectively. Similarly neural results with regression weights $w(R)$ (from input to hidden layer) and $v(R)$ (from hidden to output layer) with four, five, and six nodes are given in fourth, sixth, and ninth column, respectively.

Analytical and neural results with arbitrary and regression based weights for six nodes in hidden layer are compared in Figures 2 and 3. The error plot is shown in Figure 4. Absolute deviations in % values have been calculated in Table 1 and the maximum deviation for arbitrary weights neural results (six hidden nodes) is 3.67 (eighth column) and for regression based weights it is 1.47 (tenth column). From Figures 2 and 3, one may see that results from the regression-based weights agree exactly at all points with analytical results but for results with arbitrary weights they are not so. Thus, one may see that the neural results with regression based weights are more accurate.

It may be seen that by increasing the number of nodes in hidden layer from four to six, the results are found to be better. Although the authors increased the number of nodes in hidden layer beyond six, but the results were not improving.

The first problem has also been solved by a well-known numerical method, namely, using Euler and Runge-kutta method. Table 2 shows comparison between the neural results (with six hidden nodes) and other numerical results (Euler and Runge-Kutta results).

Table 1: Analytical and neural solutions with arbitrary and regression based weights (Example 1).

Input data	Analytical	Neural results							
		$w(A), v(A)$ (four nodes)	$w(R), v(R)$ (four nodes)	$w(A), v(A)$ (five nodes)	$w(R), v(R)$ (five nodes)	$w(A), v(A)$ (six nodes)	Deviation%	$w(R), v(R)$ (six nodes)	Deviation%
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.00	1.0000	0.00
0.05	0.9536	1.0015	0.9998	1.0002	0.9768	0.9886	3.67	0.9677	1.47
0.10	0.9137	0.9867	0.9593	0.9498	0.9203	0.9084	0.58	0.9159	0.24
0.15	0.8798	0.9248	0.8986	0.8906	0.8802	0.8906	1.22	0.8815	0.19
0.20	0.8514	0.9088	0.8869	0.8564	0.8666	0.8587	0.85	0.8531	0.19
0.25	0.8283	0.8749	0.8630	0.8509	0.8494	0.8309	0.31	0.8264	0.22
0.30	0.8104	0.8516	0.8481	0.8213	0.9289	0.8013	1.12	0.8114	0.12
0.35	0.7978	0.8264	0.8030	0.8186	0.8051	0.7999	0.26	0.7953	0.31
0.40	0.7905	0.8137	0.7910	0.8108	0.8083	0.7918	0.16	0.7894	0.13
0.45	0.7889	0.7951	0.7908	0.8028	0.7948	0.7828	0.77	0.7845	0.55
0.50	0.7931	0.8074	0.8063	0.8007	0.7960	0.8047	1.46	0.7957	0.32
0.55	0.8033	0.8177	0.8137	0.8276	0.8102	0.8076	0.53	0.8041	0.09
0.60	0.8200	0. 8211	0.8190	0.8362	0.8246	0.8152	0.58	0.8204	0.04
0.65	0.8431	0.8617	0.8578	0.8519	0.8501	0.8319	1.32	0.8399	0.37
0.70	0.8731	0.8896	0.8755	0.8685	0.8794	0.8592	1.59	0.8711	0.22
0.75	0.9101	0.9281	0.9231	0.9229	0.9139	0.9129	0.31	0.9151	0.54
0.80	0.9541	0.9777	0.9613	0.9897	0.9603	0.9755	2.24	0.9555	0.14
0.85	1.0053	1.0819	0.9930	0.9956	1. 0058	1.0056	0.03	0.9948	1.04
0.90	1.0637	1.0849	1.1020	1.0714	1.0663	1.0714	0.72	1.0662	0.23
0.95	1.1293	1.2011	1.1300	1.1588	1.1307	1.1281	0.11	1.1306	0.11
1.00	1.2022	1.2690	1.2195	1.2806	1.2139	1.2108	0.71	1.2058	0.29

Example 2. Let us consider the following second order damped free vibration equation:

$$\frac{d^2\psi}{dx^2} + 4\frac{d\psi}{dx} + 4\psi = 0, \quad x \in [0, 4]. \tag{26}$$

With initial conditions $\psi(0) = 1, \psi'(0) = 1$.

As discussed above, we can write the trail solution as

$$\psi_t(x, p) = 1 + x + x^2 N(x, p). \tag{27}$$

Then, the network is trained for 40 equidistant points in $[0, 4]$ and with four, five, and six hidden nodes according to arbitrary and regression-based algorithm. In Table 3, we compare the analytical solutions with neural solutions taking arbitrary- and regression-based weights for four, five, and six nodes in hidden layer. Here, analytic results are cited in second column of Table 3. Neural results for arbitrary weights $w(A)$ (from input to hidden layer) and $v(A)$ (from hidden to output layer) with four, five, and six nodes are shown in third, fifth, and seventh column, respectively. Neural results with regression-based weights $w(R)$ (from input to hidden layer) and $v(R)$ (from hidden to output layer) with four, five and six nodes are cited in fourth, sixth, and eighth column, respectively.

Analytical and neural results which are obtained for random initial weights are depicted in Figure 5. Figure 6 shows comparison between analytical and neural results for regression-based initial weights for six hidden nodes. Finally, the error plot between analytical and RBNN results are shown in Figure 7.

Example 3. Now we consider an initial value problem as follows:

$$\frac{dy}{dx} + 5y = e^{-3x}, \tag{28}$$

subject to $\psi(0) = 0$.

The ANN trial solution is written as

$$\psi_t(x, p) = xN(x, p). \tag{29}$$

Ten equidistant points in the given domain which are taken with four, five, and six hidden nodes according to arbitrary and regression-based algorithms have been considered. Comparison of analytical and neural results with arbitrary- and regression-based weights have been shown in Table 4. Also, other numerical results, namely, Euler and Runge-Kutta results are compared with neural results in this table.

Analytical and traditional neural results obtained using random initial weights with six nodes are depicted in Figure 8. Similarly, Figure 9 shows comparison between analytical and neural results with regression-based initial weights for six hidden nodes. Finally, the error plot between analytical and RBNN results are cited in Figure 10.

Table 2: Comparison of the results (Example 1).

Input data	Analytical	Euler	Runge-Kutta	$w(R), v(R)$ (Six nodes)
0	1.0000	1.0000	1.0000	1.0000
0.0500	0.9536	0.9500	0.9536	0.9677
0.1000	0.9137	0.9072	0.9138	0.9159
0.1500	0.8798	0.8707	0.8799	0.8815
0.2000	0.8514	0.8401	0.8515	0.8531
0.2500	0.8283	0.8150	0.8283	0.8264
0.3000	0.8104	0.7953	0.8105	0.8114
0.3500	0.7978	0.7810	0.7979	0.7953
0.4000	0.7905	0.7721	0.7907	0.7894
0.4500	0.7889	0.7689	0.7890	0.7845
0.5000	0.7931	0.7717	0.7932	0.7957
0.5500	0.8033	0.7805	0.8035	0.8041
0.6000	0.8200	0.7958	0.8201	0.8204
0.6500	0.8431	0.8178	0.8433	0.8399
0.7000	0.8731	0.8467	0.8733	0.8711
0.7500	0.9101	0.8826	0.9102	0.9151
0.8000	0.9541	0.9258	0.9542	0.9555
0.8500	1.0053	0.9763	1.0054	0.9948
0.9000	1.0637	1.0342	1.0638	1.0662
0.9500	1.1293	1.0995	1.1294	1.1306
1.000	1.2022	1.1721	1.2022	1.2058

Example 4. Here, we consider a standard differential equation which represents exponential growth as follows:

$$\frac{dy}{dx} = \alpha y, \tag{30}$$

with initial condition $y(0) = 1$.

Here $1/\alpha$ represents time constant or characteristic time. Analytic result may be found as

$$y = e^{\alpha x}. \tag{31}$$

Considering $\alpha = 1$, we have the analytical solution as $y = e^{x}$.

The ANN trial solution in this case is

$$\psi_t(x, p) = 1 + xN(x, p). \tag{32}$$

Now, the network is trained for ten equidistant points in the domain $[0, 1]$ with four, five, and six hidden nodes according to arbitrary- and regression-based algorithm. Comparison of analytical and neural results with arbitrary- $(w(A), v(A))$ and regression-based weights $(w(R), v(R))$ has been given in Table 5. Analytical and traditional neural results obtained using random initial weights with six nodes are shown in Figure 11. Figure 12 depicts comparison between analytical and neural results with regression-based initial weights for six hidden nodes. Error plot between analytical and RBNN results is cited in Figure 13.

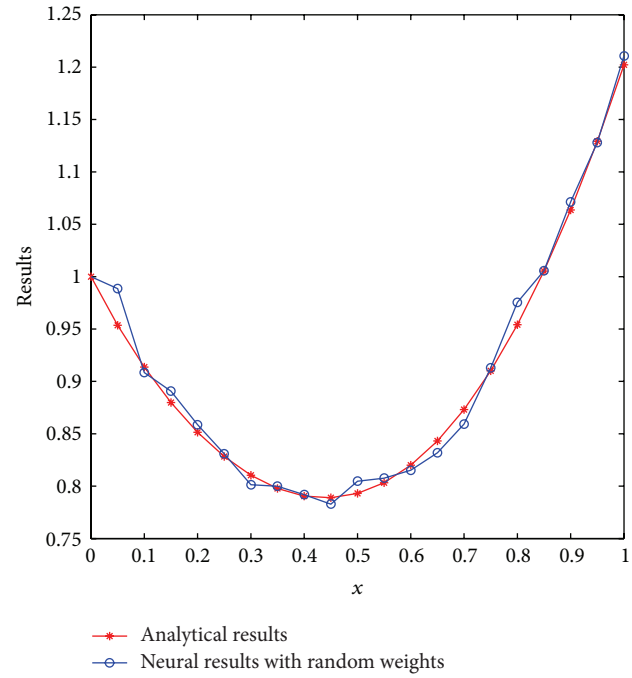

Figure 2: Plot of comparison between (analytical results) and (neural results) with arbitrary weights (Example 1).

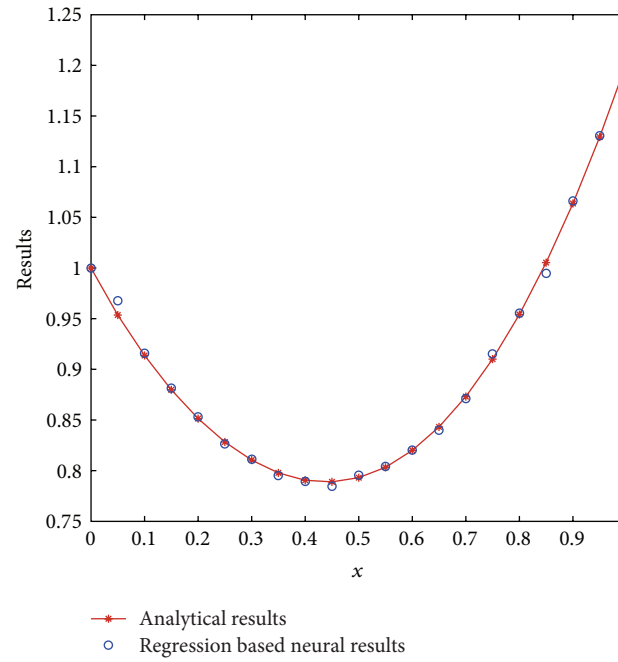

Figure 3: Plot of comparison between (analytical results) and (neural results) with regression-based weights for six nodes (Example 1).

6. Discussion and Analysis

In traditional artificial neural network, the parameters (weights/biases) are usually taken as arbitrary (random) and the number of nodes in hidden layer is considered by trial and error method. Also, few authors have used optimization

Table 3: Analytical and neural solutions with arbitrary- and regression-based weights (Example 2).

Input data	Analytical	Neural results					
		$w(A), v(A)$ (four nodes)	$w(R), v(R)$ (four nodes)	$w(A), v(A)$ (five nodes)	$w(R), v(R)$ (five nodes)	$w(A), v(A)$ (six nodes)	$w(R), v(R)$ (six nodes)
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.1	1.0643	1.0900	1.0802	1.0910	1.0878	1.0923	1.0687
0.2	1.0725	1.1000	1.0918	1.0858	1.0715	1.0922	1.0812
0.3	1.0427	1.0993	1.0691	1.0997	1.0518	1.0542	1.0420
0.4	0.9885	0.9953	0.9732	0.9780	0.9741	0.8879	0.9851
0.5	0.9197	0.9208	0.9072	0.9650	0.9114	0.9790	0.9122
0.6	0.8433	0.8506	0.8207	0.8591	0.8497	0.8340	0.8082
0.7	0.7645	0.7840	0.7790	0.7819	0.7782	0.7723	0.7626
0.8	0.6864	0.7286	0.6991	0.7262	0.6545	0.6940	0.6844
0.9	0.6116	0.6552	0.5987	0.6412	0.6215	0.6527	0.6119
1.0	0.5413	0.5599	0.5467	0.5604	0.5341	0.5547	0.5445
1.1	0.4765	0.4724	0.4847	0.4900	0.4755	0.4555	0.4634
1.2	0.4173	0.4081	0.4035	0.4298	0.4202	0.4282	0.4172
1.3	0.3639	0.3849	0.3467	0.3907	0.3761	0.3619	0.3622
1.4	0.3162	0.3501	0.3315	0.3318	0.3274	0.3252	0.3100
1.5	0.2738	0.2980	0.2413	0.2942	0.2663	0.2773	0.2759
1.6	0.2364	0.2636	0.2507	0.2620	0.2439	0.2375	0.2320
1.7	0.2036	0.2183	0.2140	0.2161	0.2107	0.2177	0.1921
1.8	0.1749	0.2018	0.2007	0.1993	0.1916	0.1622	0.1705
1.9	0.1499	0.1740	0.1695	0.1665	0.1625	0.1512	0.1501
2.0	0.1282	0.1209	0.1204	0.1371	0.1299	0.1368	0.1245
2.1	0.1095	0.1236	0.1203	0.1368	0.1162	0.1029	0.1094
2.2	0.0933	0.0961	0.0942	0.0972	0.0949	0.0855	0.09207
2.3	0.0794	0.0818	0.0696	0.0860	0.0763	0.0721	0.0761
2.4	0.0675	0.0742	0.0715	0.0849	0.0706	0.0526	0.0640
2.5	0.0573	0.0584	0.0419	0.0609	0.0543	0.0582	0.0492
2.6	0.0485	0.0702	0.0335	0.0533	0.0458	0.0569	0.0477
2.7	0.0411	0.0674	0.0602	0.0581	0.0468	0.0462	0.0409
2.8	0.0348	0.0367	0.0337	0.0387	0.0328	0.0357	0.03460
2.9	0.0294	0.0380	0.0360	0.0346	0.0318	0.0316	0.0270
3.0	0.0248	0.0261	0.0207	0.0252	0.0250	0.0302	0.0247
3.1	0.0209	0.0429	0.0333	0.0324	0.0249	0.0241	0.0214
3.2	0.0176	0.0162	0.0179	0.0154	0.0169	0.0166	0.0174
3.3	0.0148	0.0159	0.0137	0.0158	0.0140	0.0153	0.0148
3.4	0.0125	0.0138	0.0135	0.0133	0.0130	0.0133	0.0129
3.5	0.0105	0.0179	0.0167	0.0121	0.0132	0.0100	0.0101
3.6	0.0088	0.0097	0.0096	0.0085	0.0923	0.0095	0.0090
3.7	0.0074	0.0094	0.0092	0.0091	0.0093	0.0064	0.0071
3.8	0.0062	0.0081	0.0078	0.0083	0.0070	0.0061	0.0060
3.9	0.0052	0.0063	0.0060	0.0068	0.0058	0.0058	0.0055
4.0	0.0044	0.0054	0.0052	0.0049	0.0049	0.0075	0.0046

Table 4: Analytical and neural solutions with arbitrary- and regression-based weights (Example 3).

Input data	Analytical	Euler	Runge-Kutta	Neural results					
				$w(A), v(A)$ (four nodes)	$w(R), v(R)$ (four nodes)	$w(A), v(A)$ (five nodes)	$w(R), v(R)$ (five nodes)	$w(A), v(A)$ (six nodes)	$w(R), v(R)$ (six nodes)
0	0	0	0	0	0	0	0	0	0
0.1	0.0671	0.1000	0.0671	0.0440	0.0539	0.0701	0.0602	0.0565	0.0670
0.2	0.0905	0.1241	0.0904	0.0867	0.0938	0.0877	0.0927	0.0921	0.0907
0.3	0.0917	0.1169	0.0917	0.0849	0.0926	0.0889	0.0932	0.0931	0.0918
0.4	0.0829	0.0991	0.0829	0.0830	0.0876	0.0806	0.0811	0.0846	0.0824
0.5	0.0705	0.0797	0.0705	0.0760	0.0748	0.0728	0.0714	0.0717	0.0706
0.6	0.0578	0.0622	0.0577	0.0492	0.0599	0.0529	0.0593	0.0536	0.0597
0.7	0.0461	0.0476	0.0461	0.0433	0.0479	0.0410	0.0453	0.0450	0.0468
0.8	0.0362	0.0360	0.0362	0.0337	0.0319	0.0372	0.0370	0.0343	0.0355
0.9	0.0280	0.0271	0.0280	0.0324	0.0308	0.0309	0.0264	0.0249	0.0284
1.0	0.0215	0.0203	0.0215	0.0304	0.0282	0.0255	0.0247	0.0232	0.0217

Table 5: Analytical and neural solutions with arbitrary- and regression-based weights (Example 4).

Input data	Analytical	Neural results					
		$w(A), v(A)$ (four nodes)	$w(R), v(R)$ (four nodes)	$w(A), v(A)$ (five nodes)	$w(R), v(R)$ (five nodes)	$w(A), v(A)$ (six nodes)	$w(R), v(R)$ (six nodes)
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.1000	1.1052	1.1069	1.1061	1.1093	1.1060	1.1075	1.1051
0.2000	1.2214	1.2337	1.2300	1.2250	1.2235	1.2219	1.2217
0.3000	1.3499	1.3543	1.3512	1.3600	1.3502	1.3527	1.3498
0.4000	1.4918	1.4866	1.4921	1.4930	1.4928	1.4906	1.4915
0.5000	1.6487	1.6227	1.6310	1.6412	1.6456	1.6438	1.6493
0.6000	1.8221	1.8303	1.8257	1.8205	1.8245	1.8234	1.8220
0.7000	2.0138	2.0183	2.0155	2.0171	2.0153	2.0154	2.0140
0.8000	2.2255	2.2320	2.2302	2.2218	2.2288	2.2240	2.2266
0.9000	2.4596	2.4641	2.4625	2.4664	2.4621	2.4568	2.4597
1.0000	2.7183	2.7373	2.7293	2.7232	2.7177	2.7111	2.7186

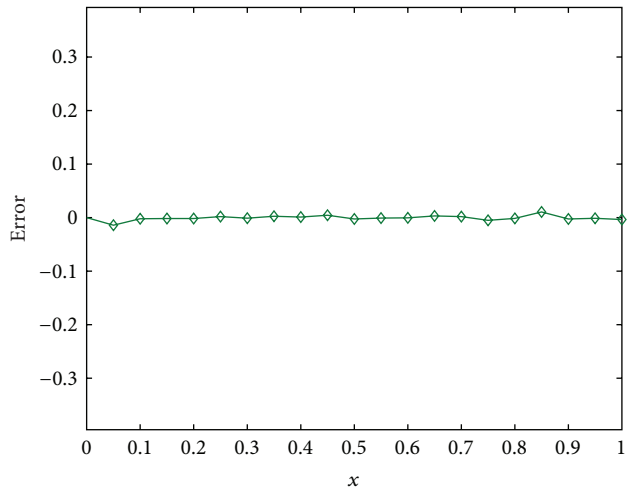

Figure 4: Error plot between analytical- and regression-based weights approximation solution (Example 1).

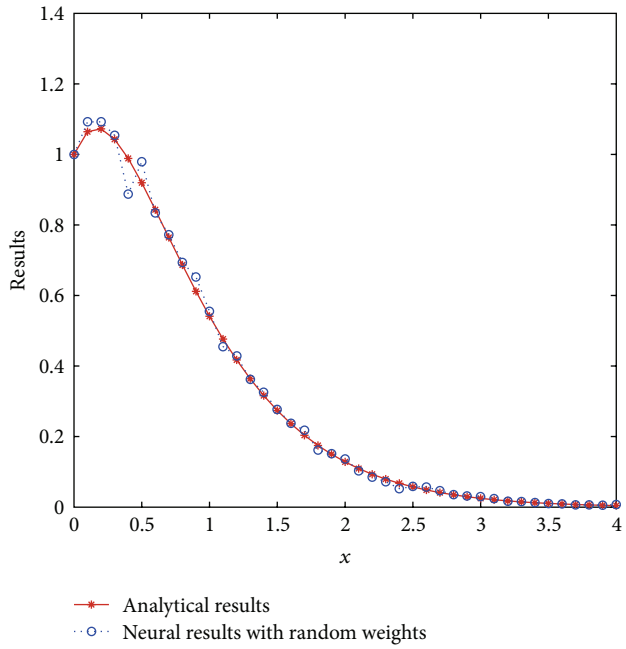

Figure 5: Plot of comparison between (analytical results) and (neural results) with arbitrary weights (for six nodes) (Example 2).

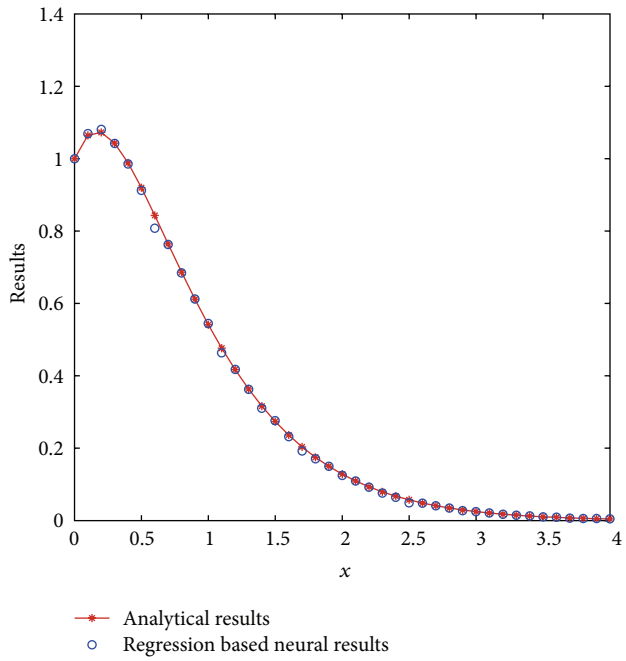

Figure 6: Plot of comparison between (analytical solutions) and (neural solutions) with regression based weights (for six nodes) (Example 2).

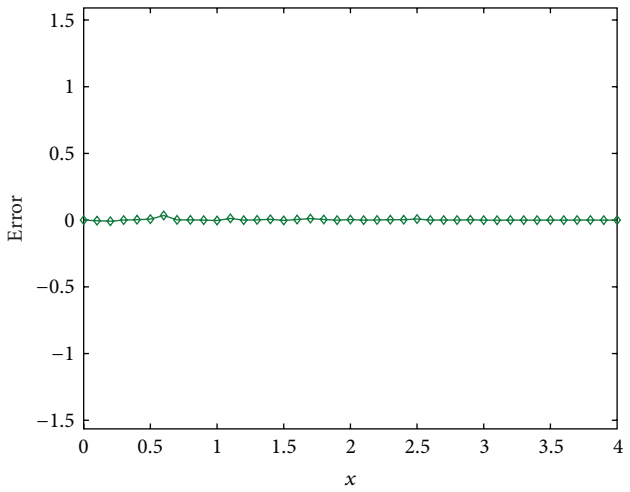

Figure 7: Error plot between analytical, and regression-based weights solution (Example 2).

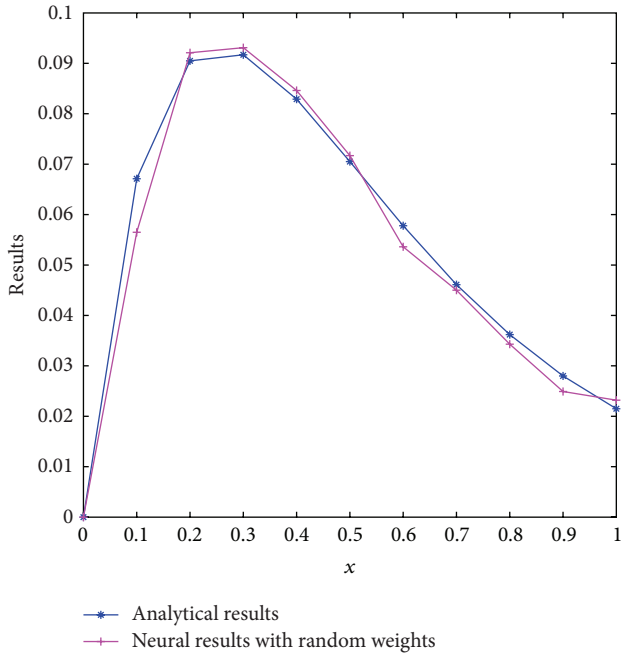

Figure 8: Plot of comparison between (analytical results) and (neural results) with arbitrary weights (for six nodes) (Example 3).

technique to minimize the error. In this investigation, a regression-based artificial neural network with combinations of initial weights (arbitrary and regression based) in the connections is considered. We have fixed the number of nodes in hidden layer according to the degree of polynomial of regression fitting. The initial weights from input to hidden and hidden to output layer are taken by using regression-based weight generation. Back propagation algorithm has been employed for modification of the parameters without use of any optimization technique. Also, time of computation is less than traditional artificial neural architecture. Table 6 shows the computation of training time in hours with four, five, and six hidden nodes.

It is well known that the other numerical methods are usually iterative in nature, where we fix the step size before the start of 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.

Table 6: Time of computation.

Problems	Time of computation in hours					
	Traditional ANN			Proposed ANN (four nodes)	Proposed ANN (five nodes)	Proposed ANN (six nodes)
	Four nodes	Five nodes	Six nodes			
Example 1	1.57 hrs	1.51	1.49	1.31	1.27	1.09
Example 2	3.06	3.00	2.44	2.23	1.55	1.38

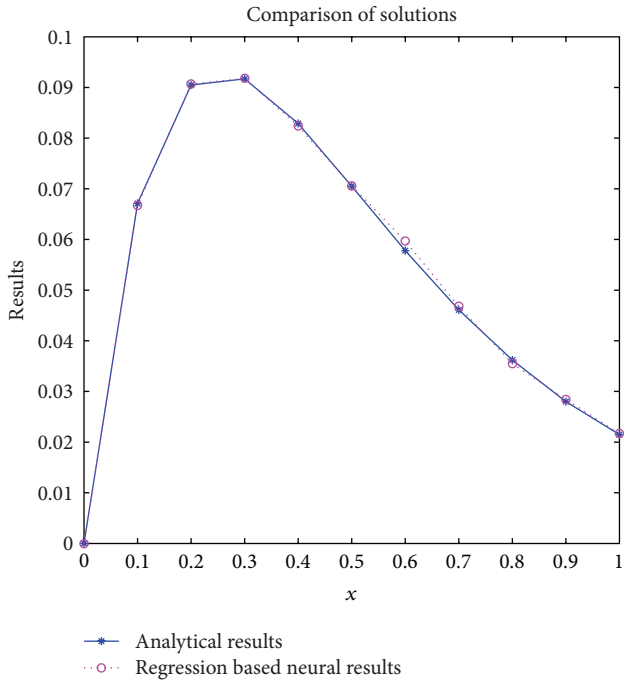

Figure 9: Plot of comparison between (analytical solutions) and (neural solutions) with regression based weights (for six nodes) (Example 3).

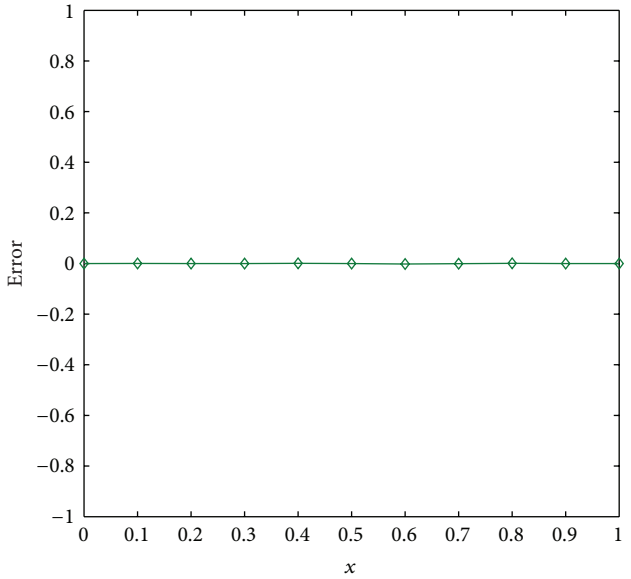

Figure 10: Error plot between analytical, and regression-based weights solution (Example 3).

ANN may be one of the reliefs where we may overcome this repetition of iterations. The authors are not claiming that the method presented is most accurate. As it may be seen by the comparison in Tables 2 and 4 that Runge-Kutta method although it gives better result but the above repetitive nature is required for each step size. Here, after getting the converged ANN, we may use it as a black box to get numerical results of any arbitrary point in the domain.

Here, we have considered three, four, and five degree polynomial for regression fitting. One may consider higher degree polynomial in the simulation but it has been seen that by increasing the degree of the polynomials, the accuracy does not usually increase. In the future, it needs to develop a methodology about what degree polynomial one should use to get a result with acceptable accuracy. This is however not of the scope of this paper and the authors are working in this direction and hope to communicate the findings in the future.

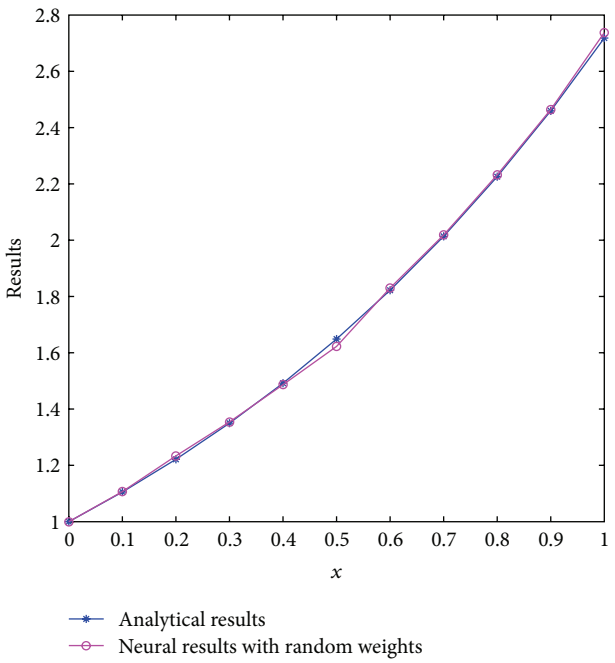

Figure 11: Plot of comparison between (analytical results) and (neural results) with arbitrary weights (for six nodes) (Example 4).

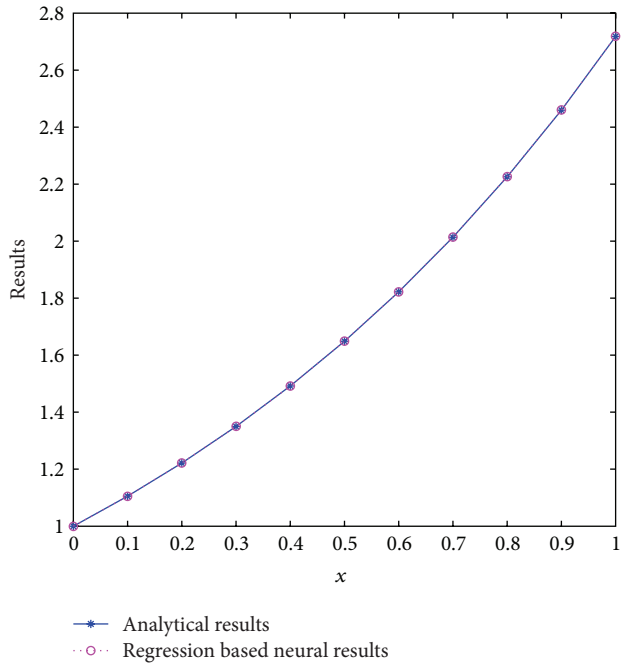

FIGURE 12: Plot of comparison between (analytical solutions) and (neural solutions) with regression based weights (for six nodes) (Example 4).

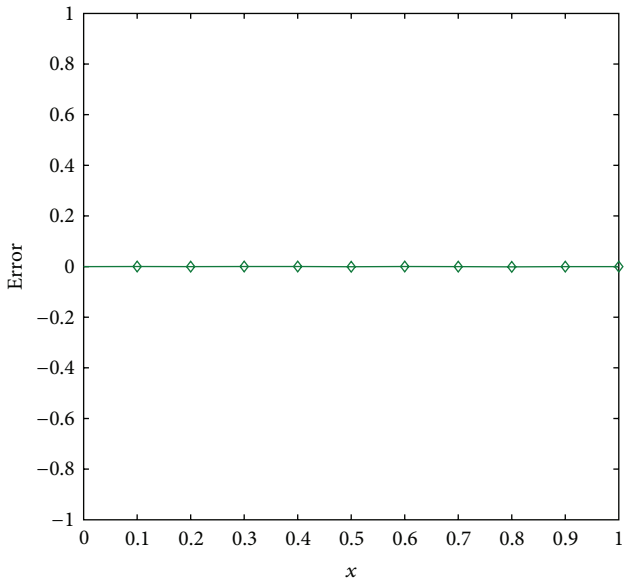

FIGURE 13: Error plot between analytical and regression based weights solution (Example 4).

7. Conclusion

This paper presents a new approach to solve ordinary differential equations by using regression based artificial neural network model. Accuracy of the proposed method has been examined by solving a first order and a second order damped free vibration problem. The main value of the paper is that the numbers of nodes in hidden layer are fixed according to the degree of polynomial in the regression. Accordingly, here, comparisons of different neural architectures corresponding to different regression models are investigated. Moreover, the algorithm is unsupervised and error back propagation algorithm is used to minimize the error function. Corresponding initial weights from input to hidden and hidden to output are all obtained by the proposed procedure. The trail solution is closed and differentiable. One may see from the tables and graphs that the initial weights generated by regression model make the results more accurate. Lastly, it may be mentioned that the implemented Regression Based Neural Network (RBNN) algorithm is simple, computationally efficient, and straight forward.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The first author is thankful to the Department of Science and Technology (DST), Government of India for the financial support under Women Scientist Scheme-A.

References

[1] H. J. Ricardo, *A Modern Introduction to Differential Equations*, Elsevier, 2nd edition, 2009.

[2] H. Lee and I. S. Kang, "Neural algorithm for solving differential equations," *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, 1990.

[3] A. J. Meade Jr. and A. A. Fernandez, "The numerical solution of linear ordinary differential equations by feedforward neural networks," *Mathematical and Computer Modelling*, vol. 19, no. 12, pp. 1–25, 1994.

[4] A. J. Meade Jr. and A. A. Fernandez, "Solution of nonlinear ordinary differential equations by feedforward neural networks," *Mathematical and Computer Modelling*, vol. 20, no. 9, pp. 19–44, 1994.

[5] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[6] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou, "Neural-network methods for boundary value problems with irregular boundaries," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–1049, 2000.

[7] D. R. Parisi, M. C. Mariani, and M. A. Laborde, "Solving differential equations with unsupervised neural networks," *Chemical Engineering and Processing*, vol. 42, no. 8-9, pp. 715–721, 2003.

[8] A. Malek and R. Shekari Beidokhti, "Numerical solution for high order differential equations using a hybrid neural network-Optimization method," *Applied Mathematics and Computation*, vol. 183, no. 1, pp. 260–271, 2006.

[9] B. Choi and J.-H. Lee, "Comparison of generalization ability on solving differential equations using backpropagation and reformulated radial basis function networks," *Neurocomputing*, vol. 73, no. 1–3, pp. 115–118, 2009.

[10] H. S. Yazdi, M. Pakdaman, and H. Modaghegh, "Unsupervised kernel least mean square algorithm for solving ordinary differential equations," *Neurocomputing*, vol. 74, no. 12-13, pp. 2062–2071, 2011.

[11] N. Selvaraju and J. Abdul Samant, "Solution of matrix Riccati differential equation for nonlinear singular system using neural networks," *International Journal of Computer Applications*, vol. 29, pp. 48–54, 2010.

[12] S. He, K. Reif, and R. Unbehauen, "Multilayer neural networks for solving a class of partial differential equations," *Neural Networks*, vol. 13, no. 3, pp. 385–396, 2000.

[13] M. Kumar and N. Yadav, "Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey," *Computers and Mathematics with Applications*, vol. 62, no. 10, pp. 3796–3811, 2011.

[14] I. G. Tsoulos, D. Gavrilis, and E. Glavas, "Solving differential equations with constructed neural networks," *Neurocomputing*, vol. 72, no. 10–12, pp. 2385–2391, 2009.

[15] L. Jianyu, L. Siwei, Q. Yingjian, and H. Yaping, "Numerical solution of elliptic partial differential equation using radial basis function neural networks," *Neural Networks*, vol. 16, no. 5-6, pp. 729–734, 2003.

[16] Y. Shirvany, M. Hayati, and R. Moradian, "Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations," *Applied Soft Computing Journal*, vol. 9, no. 1, pp. 20–29, 2009.

[17] N. Mai-Duy and T. Tran-Cong, "Numerical solution of differential equations using multiquadric radial basis function networks," *Neural Networks*, vol. 14, no. 2, pp. 185–199, 2001.

[18] T. Leephakpreeda, "Novel determination of differential-equation solutions: universal approximation method," *Journal of Computational and Applied Mathematics*, vol. 146, no. 2, pp. 443–457, 2002.

[19] C. Franke and R. Schaback, "Solving partial differential equations by collocation using radial basis functions," *Applied Mathematics and Computation*, vol. 93, no. 1, pp. 73–82, 1998.

[20] N. Smaoui and S. Al-Enezi, "Modelling the dynamics of nonlinear partial differential equations using neural networks," *Journal of Computational and Applied Mathematics*, vol. 170, no. 1, pp. 27–58, 2004.

[21] I. G. Tsoulos and I. E. Lagaris, "Solving differential equations with genetic programming," *Genetic Programming and Evolvable Machines*, vol. 7, no. 1, pp. 33–54, 2006.

[22] K. S. McFall and J. R. Mahan, "Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions," *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1221–1233, 2009.

[23] S. A. Hoda and H. A. Nagla, "Neural network methods for mixed boundary value problems," *International Journal of Nonlinear Science*, vol. 11, pp. 312–316, 2011.

[24] J. M. Zurada, *Introduction to Artificial Neural Network*, West Publishing, 1994.

[25] S. Haykin, *Neural Networks a Comprehensive Foundation*, Prentice Hall, New York, NY, USA, 1999.

[26] S. Chakraverty, V. P. Singh, and R. K. Sharma, "Regression based weight generation algorithm in neural network for estimation of frequencies of vibrating plates," *Computer Methods in Applied Mechanics and Engineering*, vol. 195, no. 33–36, pp. 4194–4202, 2006.

[27] V. P. Singh, S. Chakraverty, R. K. Sharma, and G. K. Sharma, "Modeling vibration frequencies of annular plates by regression based neural network," *Applied Soft Computing Journal*, vol. 9, no. 1, pp. 439–447, 2009.

[28] S. Mall and S. Chakraverty, "Regression Based Neural network training for the solution of ordinary differential equations," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, pp. 136–149, 2013.