Hindawi Publishing Corporation
ISRN Applied Mathematics
Volume 2013, Article ID 650467, 7 pages
http://dx.doi.org/10.1155/2013/650467

Research Article

Design Feed Forward Neural Network to Solve Singular Boundary Value Problems

Luma N. M. Tawfiq and Ashraf A. T. Hussein

Department of Mathematics, College of Education Ibn Al-Haitham, Baghdad University, Iraq

Correspondence should be addressed to Ashraf A. T. Hussein; ashraf_adnan88@yahoo.com

Received 14 May 2013; Accepted 9 June 2013

Academic Editors: Z. Huang and X. Wen

The aim of this paper is to design feed forward neural network for solving second-order singular boundary value problems in ordinary differential equations. The neural networks use the principle of back propagation with different training algorithms such as quasi-Newton, Levenberg-Marquardt, and Bayesian Regulation. Two examples are considered to show that effectiveness of using the network techniques for solving this type of equations. The convergence properties of the technique and accuracy of the interpolation technique are considered.

1. Introduction

The study of solving differential equations using artificial neural network (Ann) was initiated by Agatonovic-Kustrin and Beresford in [1]. Lagaris et al. in [2] employed two networks, a multilayer perceptron and a radial basis function network, to solve partial differential equations (PDE) with boundary conditions defined on boundaries with the case of complex boundary geometry. Tawfiq [3] proposed a radial basis function neural network (RBFNN) and Hopfield neural network (unsupervised training network). Neural networks have been employed before to solve boundary and initial value problems. Malek and Shekari Beidokhti [4] reported a novel hybrid method based on optimization techniques and neural networks methods for the solution of high order ODE which used three-layered perceptron network. Akca et al. [5] discussed different approaches of using wavelets in the solution of boundary value problems (BVP) for ODE, also introduced convenient wavelet representations for the derivatives for certain functions, and discussed wavelet network algorithm. Mc Fall [6] presented multilayer perceptron networks to solve BVP of PDE for arbitrary irregular domain where he used logsig. transfer function in hidden layer and pure line in output layer and used gradient decent training algorithm; also, he used RBFNN for solving this problem and compared between them. Junaid et al. [7] used Ann with genetic training algorithm and log sigmoid function for solving first-order ODE. Abdul Samath et al. [8] suggested the solution of the matrix Riccati differential equation (MRDE) for nonlinear singular system using Ann. Ibraheem and Khalaf [9] proposed shooting neural networks algorithm for solving two-point second-order BVP in ODEs which reduced the equation to the system of two equations of first order. Hoda and Nagla [10] described a numerical solution with neural networks for solving PDE, with mixed boundary conditions. Majidzadeh [11] suggested a new approach for reducing the inverse problem for a domain to an equivalent problem in a variational setting using radial basis functions neural network; also he used cascade feed forward to solve two-dimensional Poisson equation with back propagation and Levenberg-Marquardt train algorithm with the architecture three layers and 12 input nodes, 18 tansig. transfer functions in hidden layer, and 3 linear nodes in output layer. Oraibi [12] designed feed forward neural networks (FFNNs) for solving IVP of ODE. Ali [13] designed fast FFNN to solve two-point BVP. This paper proposed FFNN to solve two point singular boundary value problem (TPSBVP) with back propagation (BP) training algorithm.

2. Singular Boundary Value Problem

The general form of the 2nd-order two-point boundary value problem (TPBVP) is

$$y'' + P(x)\, y' + Q(x)\, y = 0, \quad a \le x \le b$$
$$y(a) = A, \quad y(b) = B, \quad \text{where } A, B \in R. \tag{1}$$

there are two types of a point $x_0 \in [0, 1]$: ordinary point and singular point.

A function $y(x)$ is analytic at x_0 if it has a power series expansion at x_0 that converges to $y(x)$ on an open interval containing x_0. A point x_0 is an ordinary point of the ODE (1), if the functions $P(x)$ and $Q(x)$ are analytic at x_0. Otherwise x_0 is a singular point of the ODE. On the other hand, if $P(x)$ or $Q(x)$ are not analytic at x_0, then x_0 is said to be a singular point [14, 15].

There is at present no theoretical work justifying numerical methods for solving problems with irregular singular points. The main practical occurrence of such problems seems to be semianalytic technique [16].

3. Artificial Neural Network

Ann is a simplified mathematical model of the human brain. It can be implemented by both electric elements and computer software. It is a parallel distributed processor with large numbers of connections; it is an information processing system that has certain performance characters in common with biological neural networks [17].

The arriving signals, called inputs, multiplied by the connection weights (adjusted) are first summed (combined) and then passed through a transfer function to produce the output for that neuron. The activation (transfer) function acts on the weighted sum of the neuron's inputs and the most commonly used transfer function is the sigmoid function (tansig.) [13].

There are two main connection formulas (types): feedback (recurrent) and feed forward connections. Feedback is one type of connection where the output of one layer routes back to the input of a previous layer, or to the same layer. Feed forward neural network (FFNN) does not have a connection back from the output to the input neurons [18].

There are many different training algorithms, but the most often used training algorithm is the Delta rule or back propagation (BP) rule. A neural network is trained to map a set of input data by iterative adjustment of the weights. Information from inputs is fed forward through the network to optimize the weights between neurons. Optimization of the weights is made by backward propagation of the error during training phase.

The Ann reads the input and output values in the training data set and changes the value of the weighted links to reduce the difference between the predicted and target (observed) values. The error in prediction is minimized across many training cycles (iteration or epoch) until network reaches specified level of accuracy. A complete round of forward-backward passes and weight adjustments using all input-output pairs in the data set is called an epoch or iteration. If a network is left to train for too long, however, it will be overtrained and will lose the ability to generalize.

In this paper, we focused on the training situation known as supervised training, in which a set of input/output data patterns is available. Thus, the Ann has to be trained to produce the desired output according to the examples.

In order to perform a supervised training we need a way of evaluating the Ann output error between the actual and the expected outputs. A popular measure is the mean squared error (MSE) or root mean squared error (RMSE) [19].

4. Description of the Method

In the proposed approach the model function is expressed as the sum of two terms: the first term satisfies the boundary conditions (BCs) and contains no adjustable parameters. The second term can be found by using FFNN which is trained so as to satisfy the differential equation and such technique called collocation neural network.

In this section, we will illustrate how our approach can be used to find the approximate solution of the general form, a 2nd-order TPSBVP:

$$x^m y''(x) = F\left(x, y(x), y'(x)\right), \tag{2}$$

where a subject to certain BCs and $m \in Z$, $x \in R$, $D \subset R$ denotes the domain and $y(x)$ is the solution to be computed.

If $y_t(x, p)$ denotes a trial solution with adjustable parameters p, the problem is transformed to a discretize form:

$$\min_p \sum_{\vec{x}_i \in \bar{D}} F\left(x_i, y_t(x_i, p), y_t'(x_i, p)\right) \tag{3}$$

subject to the constraints imposed by the BCs.

In the our proposed approach, the trial solution y_t employs an FFNN and the parameters p correspond to the weights and biases of the neural architecture. We choose a form for the trial function $y_t(x)$ such that it satisfies the BCs. This is achieved by writing it as a sum of two terms:

$$y_t(x_i, p) = A(x) + G\left(x, N(x, p)\right), \tag{4}$$

where $N(x, p)$ is a single-output FFNN with parameters p and n input units fed with the input vector x. The term $A(x)$ contains no adjustable parameters and satisfies the BCs. The second term G is constructed so as not to contribute to the BCs, since $y_t(x)$ satisfy them. This term can be formed by using an FFNN whose weights and biases are to be adjusted in order to deal with the minimization problem.

An efficient minimization of (3) can be considered as a procedure of training the FFNN, where the error corresponding to each input x_i is the value $E(x_i)$ which has to forced near zero. Computation of this error value involves not only the FFNN output but also the derivatives of the output with respect to any of its inputs.

Therefore, in computing the gradient of the error with respect to the network weights consider a multilayer FFNN with n input units (where n is the dimensions of the domain), one hidden layer with H sigmoid nodes, and a linear output unit.

Table 1: Analytic and neural solutions of Example 1.

Input	Analytic solution	Out of suggested FFNN $y_t(x)$ for different training algorithms		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	1	1	1.00026931058869	1.00000028044679
0.1	0.995004165278026	0.994992000703617	0.995004165151447	0.995003016942414
0.2	0.980066577841242	0.980066577841242	0.980038331595444	0.980068202697384
0.3	0.955336489125606	0.955336489125606	0.955326539544390	0.955327719921018
0.4	0.921060994002885	0.921060994002885	0.921060993873107	0.921057767471849
0.5	0.877582561890373	0.877582561890373	0.877583333411869	0.877587507015204
0.6	0.825335614909678	0.825335614909678	0.825335614867241	0.825332390929038
0.7	0.764842187284489	0.764838404395133	0.764842187227892	0.764831348535762
0.8	0.696706709347165	0.696656420761032	0.696706709263878	0.696708274506624
0.9	0.621609968270664	0.621454965504409	0.621609968278774	0.621608898908218
1.0	0.540302305868140	0.540302305868140	0.540302305877365	0.540302558954826

Table 2: Accuracy of solution for Example 1.

The error $E(x) = \lvert y_t(x) - y_a(x)\rvert$ where $y_t(x)$ computed by the following training algorithm		
Trainlm	Trainbfg	Trainbr
0	0.000269310588686622	$2.80446790679179e-07$
$1.21645744084464e-05$	$1.26578747483563e-10$	$1.14833561148942e-06$
0	$2.82462457977806e-05$	$1.62485614274566e-06$
0	$9.94958121636191e-06$	$8.76920458825481e-06$
0	$1.29778077173626e-10$	$3.22653103579373e-06$
0	$7.71521496467642e-07$	$4.94512483140142e-06$
0	$4.24377200047843e-11$	$3.22398064012130e-06$
$3.78288935598548e-06$	$5.65963942378289e-11$	$1.08387487263162e-05$
$5.02885861338731e-05$	$8.32875990397497e-11$	$1.56515945903823e-06$
0.000155002766255130	$8.10940203876953e-12$	$1.06936244681499e-06$
0	$9.22539822312274e-12$	$2.53086685830795e-07$

For a given input x, the output of the FFNN is

$$N = \sum_{i=1}^{H} v_i \sigma(z_i), \quad \text{where } z_i = \sum_{j=1}^{n} w_{ij} x_j + b_i. \tag{5}$$

w_{ij} denotes the weight connecting the input unit j to the hidden unit i, v_i denotes the weight connecting the hidden unit i to the output unit, b_i denotes the bias of hidden unit i, and $\sigma(z)$ is the sigmoid transfer function (tansig.).

The gradient of FFNN with respect to the parameters of the FFNN can be easily obtained as

$$\begin{aligned} \frac{\partial N}{\partial v_i} &= \sigma(z_i), \\ \frac{\partial N}{\partial b_i} &= v_i \sigma'(z_i), \\ \frac{\partial N}{\partial w_{ij}} &= v_i \sigma'(z_i) x_j. \end{aligned} \tag{6}$$

Once the derivative of the error with respect to the network parameters has been defined, then it is a straight forward to employ any minimization technique. It must also be noted that the batch mode of weight updates may be employed.

5. Illustration of the Method

In this section we describe solution of TPSBVP using FFNN.

To illustrate the method, we will consider the 2nd-order TPSBVP:

$$\frac{x^m d^2 y(x)}{dx^2} = f\left(x, y, y'\right), \tag{7}$$

where $x \in [a, b]$ and the BC: $y(a) = A$, $y(b) = B$; a trial solution can be written as

$$\begin{aligned} y_t(x, p) = {} & \frac{(bA - aB)}{(b-a)} + \frac{(B-A)x}{(b-a)} \\ & + (x-a)(x-b) N(x, p), \end{aligned} \tag{8}$$

where $N(x, p)$ is the output of an FFNN with one input unit for x and weights p.

Table 3: The performance of the train with epoch and time for Example 1.

TrainFcn	Performance of train	Epoch	Time	MSE
Trainlm	0:00	75	0:00:01	$2.1859e-009$
Trainbfg	$6.42e-21$	6187	0:04:23	$6.6751e-009$
Trainbr	$5.88e-12$	1861	0:00:30	$2.0236e-011$

Note that $y_t(x)$ satisfies the BC by construction. The error quantity to be minimized is given by

$$E[p] = \sum_{i=1}^{n} \left\{ \frac{d^2 y_t(x_i, p)}{dx^2} - f\left(x_i, y_t(x_i, p), \frac{dy_t(x_i, p)}{dx}\right) \right\}^2, \tag{9}$$

where the $x_i \in [a, b]$. Since

$$\begin{aligned} \frac{dy_t(x,p)}{dx} &= \frac{(B-A)}{(b-a)} + \{(x-a) + (x-b)\} N(x,p) \\ &\quad + (x-a)(x-b)\frac{dN(x,\vec{p})}{dx}, \\ \frac{d^2 y_t(x,p)}{dx^2} &= 2N(x,p) + 2\{(x-a) + (x-b)\}\frac{dN(x,\vec{p})}{dx} \\ &\quad + (x-a)(x-b)\frac{d^2 N(x,p)}{dx^2}, \end{aligned} \tag{10}$$

it is straightforward to compute the gradient of the error with respect to the parameters p using (6). The same holds for all subsequent model problems.

6. Example

In this section we report numerical result, using a multi-layer FFNN having one hidden layer with **5** hidden units (neurons) and one linear output unit. The sigmoid activation of each hidden unit is tansig.; the analytic solution $y_a(x)$ was known in advance. Therefore we test the accuracy of the obtained solutions by computing the deviation:

$$\Delta y(x) = \left| y_t(x) - y_a(x) \right|. \tag{11}$$

In order to illustrate the characteristics of the solutions provided by the neural network method, we provide figures displaying the corresponding deviation $\Delta y(x)$ both at the few points (training points) that were used for training and at many other points (test points) of the domain of equation. The latter kind of figures are of major importance since they show the interpolation capabilities of the neural solution which to be superior compared to other solution obtained by using other methods. Moreover, we can consider points outside the training interval in order to obtain an estimate of the extrapolation performance of the obtained numerical solution.

Example 1. Consider the following 2nd-order TPSBVP:

$$y'' + \left(\frac{1}{x}\right) y' + \cos(x) + \frac{\sin(x)}{x} = 0, \quad x \in [0, 1], \tag{12}$$

Table 4: Weight and bias of the network for different training algorithm Example 1.

(a)

Weights and bias for trainlm		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.2858	0.0759	0.1299
0.7572	0.0540	0.5688
0.7537	0.5308	0.4694
0.3804	0.7792	0.0119
0.5678	0.9340	0.3371

(b)

Weights and bias for trainbfg		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.4068	0.8334	0.2601
0.1126	0.4036	0.0868
0.4438	0.3902	0.4294
0.3002	0.3604	0.2573
0.4014	0.1403	0.2976

(c)

Weights and bias for trainbr		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.1696	0.8803	0.4075
0.2788	0.4711	0.8445
0.1982	0.4040	0.6153
0.1951	0.1792	0.3766
0.3268	0.9689	0.8772

with BC: $y'(0) = 0$, $y(1) = \cos(1)$. The analytic solution is $y_a(x) = \cos(x)$; according to (8) the trial neural form of the solution is taken to be

$$y_t(x) = \cos(1)\, x + x(x-1) N(x, p). \tag{13}$$

The FFNN trained using a grid of ten equidistant points in $[0, 1]$. Figure 1 displays the analytic and neural solutions with different training algorithm. The neural results with different types of training algorithm such as Levenberg-Marquardt (trainlm), quasi-Newton (trainbfg), and Bayesian Regulation (trainbr) introduced in Table 1 and its errors given in Table 2, Table 3 gives the performance of the train with epoch and time, and Table 4 gives the weight and bias of the designer network,

Ramos in [20] solved this example using C^1-linearization method and gave the absolute error $4.079613e-04$; also, Kumar in [21] solved this example by the three-point finite difference technique and gave the absolute error $4.4e-05$.

Table 5: Analytic and neural solutions of Example 2.

Input	Analytic solution	Out of suggested FFNN $y_t(x)$ for different training algorithms		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	1	1	0.999999999999781	0.999999991504880
0.1	1.10517091807565	1.10514495970783	1.10517935005948	1.10518327439088
0.2	1.22140275816017	1.22139291634314	1.22140560576571	1.22140301047712
0.3	1.34985880757600	1.34985880757600	1.34985880757533	1.34985789841558
0.4	1.49182469764127	1.49182469764127	1.49182469764283	1.49182590291778
0.5	1.64872127070013	1.64872127070013	1.64872127069896	1.64871934058054
0.6	1.82211880039051	1.82211880039051	1.82211880039100	1.82211668702999
0.7	2.01375270747048	2.01374698451417	2.01375253599035	2.01375577873317
0.8	2.22554092849247	2.22554092849247	2.22554092849241	2.22553872338720
0.9	2.45960311115695	2.45965304884168	2.45961509099396	2.45960396018800
1.0	2.71828182845905	2.71828182845905	2.71828182845889	2.71828168663973

Table 6: Accuracy of solutions for Example 2.

The error $E(x) = \lvert y_t(x) - y_a(x)\rvert$ where $y_t(x)$ computed by the following training algorithm		
Trainlm	Trainbfg	trainbr
0	$2.19047002758543e-13$	$8.49512027389920e-09$
$2.59583678221542e-05$	$8.43198383004840e-06$	$1.23563152347739e-05$
$9.84181703400644e-06$	$2.84760554247754e-06$	$2.52316951554477e-07$
0	$6.74571509762245e-13$	$9.09160424944489e-07$
0	$1.55675472512939e-12$	$1.20527650837587e-06$
0	$1.16662235427611e-12$	$1.93011959059852e-06$
$2.22044604925031e-16$	$4.86721773995669e-13$	$2.11336051969546e-06$
$5.72295631107167e-06$	$1.71480126542889e-07$	$3.07126269616376e-06$
0	$5.72875080706581e-14$	$2.20510526371953e-06$
$4.99376847331590e-05$	$1.19798370104007e-05$	$8.49031051686211e-07$
0	$1.51878509768721e-13$	$1.41819320731429e-07$

Example 2. Consider the following 2nd-order TPSBVP:

$$y'' = -\left[\frac{(1-2x)}{x}\right] y' - \left[\frac{(x-1)}{x}\right] y, \quad x \in [0,1]. \tag{14}$$

with BC: $y(0) = 1$, $y(1) = \exp(1)$ and the analytic solution is $y_a(x) = \exp(x)$; according to (8) the trial neural form of the solution is

$$y_t(x) = 1 + (\exp(1) - 1)\, x + x(x-1)\, N(x, p). \tag{15}$$

The FFNN trained using a grid of ten equidistant points in $[0, 1]$. Figure 2 displays the analytic and neural solutions with different training algorithms. The neural network results with different types of training algorithm such as trainlm, trainbfg, and trainbr, introduced in Table 5 and its errors given in Table 6, Table 7 gives the performance of the train with epoch and time and Table 8 gives the weight and bias of the designer network.

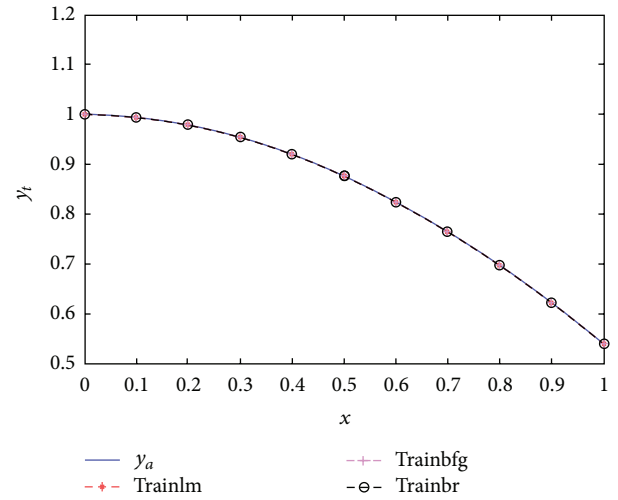

Figure 1: Analytic and neural solutions of Example 1, using trainbfg, trainbr, and trainlm.

Table 7: The performance of the train with epoch and time of Example 2.

TrainFcn	Performance of train	Epoch	Time	MSE
Trainlm	$7.04e-33$	1902	0:00:30	$2.6977e-010$
Trainbfg	$4.00e-28$	2093	0:01:04	$1.8225e-011$
Trainbr	$2.43e-12$	3481	0:00:56	$1.458e-011$

Table 8: Weight and bias of the network for different training algorithm Example 2.

(a)

Weights and bias for trainlm		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.7094	0.1626	0.5853
0.7547	0.1190	0.2238
0.2760	0.4984	0.7513
0.6797	0.9597	0.2551
0.6551	0.3404	0.5060

(b)

Weights and bias for trainbfg		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.7094	0.1626	0.5853
0.7547	0.1190	0.2238
0.2760	0.4984	0.7513
0.6797	0.9597	0.2551
0.6551	0.3404	0.5060

(c)

Weights and bias for trainbr		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.9357	0. 7406	0.2122
0.4579	0.7437	0.0985
0.2405	0.1059	0.8236
0.7639	0.6816	0.1750
0.7593	0.4633	0.1636

7. Conclusion

From the previous mentioned problems it is clear that the proposed network can be handle effectively TPSBVP and provide accurate approximate solution throughout the whole domain and not only at the training points. As evident from the tables, the results of proposed network are more precise as compared to the method suggested in [20, 21].

In general, the practical results on FFNN show that the Levenberg-Marquardt algorithm (trainlm) will have the fastest convergence, then trainbfg and then Bayesian Regulation (trainbr). However, "trainbr" does not perform well on function approximation problems. The performance of the various algorithms can be affected by the accuracy required of the approximation.

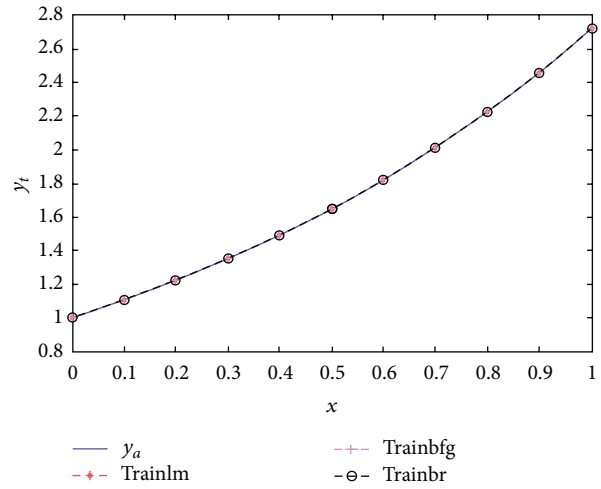

Figure 2: Analytic and neural solutions of Example 2 using different training algorithms.

References

[1] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research," *Journal of Pharmaceutical and Biomedical Analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[2] 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.

[3] L. N. M. Tawfiq, *Design and training artificial neural networks for solving differential equations [Ph.D. thesis]*, University of Baghdad, College of Education Ibn-Al-Haitham, 2004.

[4] 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.

[5] H. Akca, M. H. Al-Lail, and V. Covachev, "Survey on wavelet transform and application in ODE and wavelet networks," *Advances in Dynamical Systems and Applications*, vol. 1, no. 2, pp. 129–162, 2006.

[6] K. S. Mc Fall, *An artificial neural network method for solving boundary value problems with arbitrary irregular boundaries [Ph.D. thesis]*, Georgia Institute of Technology, 2006.

[7] A. Junaid, M. A. Z. Raja, and I. M. Qureshi, "Evolutionary computing approach for the solution of initial value problems in ordinary differential equations," *World Academy of Science, Engineering and Technology*, vol. 55, pp. 578–5581, 2009.

[8] J. Abdul Samath, P. S. Kumar, and A. Begum, "Solution of linear electrical circuit problem using neural networks," *International Journal of Computer Applications*, vol. 2, no. 1, pp. 6–13, 2010.

[9] K. I. Ibraheem and B. M. Khalaf, "Shooting neural networks algorithm for solving boundary value problems in ODEs," *Applications and Applied Mathematics*, vol. 6, no. 11, pp. 1927–1941, 2011.

[10] S. A. Hoda I. and H. A. Nagla, "On neural network methods for mixed boundary value problems," *International Journal of Nonlinear Science*, vol. 11, no. 3, pp. 312–316, 2011.

[11] K. Majidzadeh, "Inverse problem with respect to domain and artificial neural network algorithm for the solution," *Mathematical Problems in Engineering*, vol. 2011, Article ID 145608, 16 pages, 2011.

[12] Y. A. Oraibi, *Design feed forward neural networks for solving ordinary initial value problem [M.S. thesis]*, University of Baghdad, College of Education Ibn Al-Haitham, 2011.

[13] M. H. Ali, *Design fast feed forward neural networks to solve two point boundary value problems [M.S. thesis]*, University of Baghdad, College of Education Ibn Al-Haitham, 2012.

[14] I. Rachůnková, S. Staněk, and M. Tvrdý, *Solvability of Nonlinear Singular Problems for Ordinary Differential Equations*, Hindawi Publishing Corporation, New York, USA, 2008.

[15] L. F. Shampine, J. Kierzenka, and M. W. Reichelt, "Solving Boundary Value Problems for Ordinary Differential Equations in Matlab with bvp4c," 2000.

[16] H. W. Rasheed, *Efficient semi-analytic technique for solving second order singular ordinary boundary value problems [M.S. thesis]*, University of Baghdad, College of Education Ibn-Al-Haitham, 2011.

[17] A. I. Galushkin, *Neural Networks Theory*, Springer, Berlin, Germany, 2007.

[18] K. Mehrotra, C. K. Mohan, and S. Ranka, *Elements of Artificial Neural Networks*, Springer, New York, NY, USA, 1996.

[19] A. Ghaffari, H. Abdollahi, M. R. Khoshayand, I. S. Bozchalooi, A. Dadgar, and M. Rafiee-Tehrani, "Performance comparison of neural network training algorithms in modeling of bimodal drug delivery," *International Journal of Pharmaceutics*, vol. 327, no. 1-2, pp. 126–138, 2006.

[20] J. I. Ramos, "Piecewise quasilinearization techniques for singular boundary-value problems," *Computer Physics Communications*, vol. 158, no. 1, pp. 12–25, 2004.

[21] M. Kumar, "A three-point finite difference method for a class of singular two-point boundary value problems," *Journal of Computational and Applied Mathematics*, vol. 145, no. 1, pp. 89–97, 2002.