P- ISSN 1991-8941 E-ISSN 2706-6703 *Journal of University of Anbar for Pure Science (JUAPS)* Open Access
2013,(7), (1) :149-157

Design Feed Forward Neural Network To Solve Boundary Value Problems

Luma. N. M. Tawfiq **Muna. H. Ali**

Baghdad University - College of Education Ibn Al-Haitham.

A R T I C L E I N F O

Received: 6 / 3 /2012
Accepted: 18 / 9 /2012
Available online: 30/11/2013
DOI: 10.37652/juaps.2013.84593

Keywords:
Artificial neural network,
Feed Forward neural network,
Training Algorithm ,
ODE.

A B S T R A C T

The aim of this paper is to design fast feed forward neural network to present a method to solve second order boundary value problem for ordinary differential equations. That is to develop an algorithm which can speedup the solution times, reduce solver failures, and increase possibility of obtaining the globally optimal solution and we use several different training algorithms many of them having a very fast convergence rate for reasonable size networks. Finally, we illustrate the method by solving model problem and present comparison with solutions obtained using other different method.

1. Introduction

Many methods have been developed so far for solving differential equations. Some of them produce a solution in the form of an array that contains the value of the solution at a selected group of points, others use basis functions to represent the solution in analytic form and transform the original problem usually to a system of algebraic equations.[1]

Most of the previous study in solving differential equations using Artificial neural network(ANN) is restricted to the case of solving the systems of algebraic equations which result from the discretization of the domain. 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. Ann have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that : [1]

1- Information processing occurs at many simple elements called neurons that is fundamental to the operation of ANN's.
2- Signals are passed between neurons over connection links.
3- Each connection link has an associated weight which, in a typical neural net, multiplies the signal transmitted.
4- Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

The units in a network are organized into a given topology by a set of connections or weights .

ANN is Characterized by[2] :

1- Architecture: its pattern of connections between the neurons.
2- Training Algorithm : its method of determining the weights on the connections.
3- Activation function.

ANN are often classified as single layer or multilayer. In determining the number of layers, the input units are not counted as a layer, because they perform no computation. Equivalently, the number of layers in the net can be defined to be the number of layers of weighted interconnects links between the slabs of neurons [3].

2. Multilayer Feed Forward Architecture [4]

In a layered neural network the neurons are organized in the form of layers. We have at least two layers: an input and an output layer. The layers between the input and the output layer (if any) are called hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units.

* Corresponding author at: Baghdad University - College of Education Ibn Al-Haitham.;
ORCID: https://orcid.org/0000-0001-5859-6212 .Mobil:777777
E-mail address:

Extra hidden neurons raise the network's ability to extract higher-order statistics from (input) data .

The ANN is said to be fully connected in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer , otherwise the network is called partially connected. Each layer consists of a certain number of neurons; each neuron is connected to other neurons of the previous layer through adaptable synaptic weights w and biases b.

3. 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 (BC) and contains no adjustable parameters. The second term can be found by using feed forward neural network(FFNN) which is trained so as to satisfy the differential equation and such technique we called collocation neural network. Since it is known that a multilayer FFNN with one hidden layer can approximate any function to arbitrary accuracy[5], [6] , thus our FFNN contains one hidden layer.

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

$y''(x) = F(x, y(x), y'(x))$, (1)

where a subject to certain BC's and x = (x1, x2, …, xn) $\in$ Rn, D $\subset$ Rn denotes the domain and y(x) is the solution to be computed.

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

$$\text{Min}_p \sum_{\vec{x}_i \in \hat{D}} F(x_i , y_t(x_i ,p), y_t'(x_i ,p)) \quad , (2)$$

subject to the constraints imposed by the BC's.

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

$y_t(x_i , p) = A(x) + G(x, N(x, p))$, (3)

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 BC's. The second term G is constructed so as not to contribute to the BC's, since yt(x) satisfy them. This term can be formed by using a FFNN whose weights and biases are to be adjusted in order to deal with the minimization problem.

4. Computation of the Gradient

An efficient minimization of (2) can be considered as a procedure of training the FFNN, where the error corresponding to each input vector xi is the value E(xi) 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 multi layer FFNN with n input units (where n is the dimensions of the domain) one hidden layer with H sigmoid units and a linear output unit .

For a given input vector x = (x1, x2, …, xn) the output of the FFNN is :

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

wij denotes the weight connecting the input unit j to the hidden unit i

vi denotes the weight connecting the hidden unit i to the out put unit ,

bi denotes the bias of hidden unit i, and

σ (z) is the sigmoid transfer function (tansig.).

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

$$\frac{\partial N}{\partial v_i} = \sigma (z_i), \quad (4)$$

$$\frac{\partial N}{\partial b_i} = v_i \sigma'(z_i), \quad (5)$$

$$\frac{\partial N}{\partial w_{ij}} = v_i \sigma'(z_i)\, x_j , \quad (6)$$

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, the batch mode of weight updates may be employed.

5. Illustration Of The Method

In this section we describe solution of single BVP using FFNN .

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

$d^2y(x) / dx^2 = f(x, y, y')$, (7)

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

yt(x, p) = (bA– aB)/(b–a) + (B–A)x /(b–a) + (x–a)(x–b)N(x, p), (8)

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

Note that

yt(x) satisfies the BC by construction. The error quantity to be minimized is given by :

$$E[p] = \sum_{i=1}^{n}\{ d2yt(xi ,p) / dx2 – f(xi , yt(xi ,p) , dyt(xi ,p) / dx) \}2 \quad ,(9)$$

where the xi ∈ [a , b]. Since :

$$dyt(x, p)/dx = (B–A)/(b–a)+ \{(x–a)+(x–b)\}N(x,p) + (x–a) (x–b) \frac{dN(x,\vec{p})}{dx}$$

and

$$d2yt(x, p) /dx2 = 2N(x, p) + 2\{(x–a)+(x–b)\} \frac{dN(x,\vec{p})}{dx} + (x–a) (x–b) d2 N(x, p) /dx2$$

it is straightforward to compute the gradient of the error with respect to the parameters p using (4) – (6). The same holds for all subsequent model problems.

6.Algorithm:

the main steps of the algorithm are the following:

Step1: Determine the variable interval of the x, i.e. (x ∈ [a,b]).

Step2: input the analytic solution .

Step3: Determine the Boundary condition.

Step4: Determine the structure of the neural network for solving BVP.

Step5: Determine the activation function and corresponding training algorithm Complete the design.

Step6: Determine the trial solution.

Step7: Implementation.

Step8: compared the neural results and the exact results.

Step9: stop after obtain the globally optimal solution.

Step10: if no.

Step9: Go to 7.

7. Example

In this section we report numerical result, we use 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 ya(x) was known in advance. Therefore we test the accuracy of the obtained solutions by computing the deviation :

Δy(x) = | yt(x) – ya(x) |.

In order to illustrate the characteristics of the solutions provided by the neural network method, we provide figures displaying the corresponding deviation Δy(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 BVP : d2y/dx2 = - dy/dx + 2y

with BC: y(0) = 1 , y(1) = e and x ∈ [0, 1]. The analytic solution is : ya(x) = exp(x) , according to (8) the trial neural form of the solution is taken to be :

yt(x) = 1 + (e -1) x + x (x - 1) N(x, p) .

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

Ibraheem and Khalaf [7] solve this example by using (integration and interpolation techniques) and Neural Networks and gave the maximum error value is max | yexact - yNN | = 1.2089E-008 and solution time is 5.9070 sec. and the result obtained by the neural network given in figure 2

Example 2

Consider the following 2nd order BVP :

$$\frac{d^2y}{dx^2} = \frac{1}{2x^2}(y^3 - 2y^2),$$

with BC: y(0) = 0 , y(1) = 1 and x ∈ [0, 1]. The analytic solution is : ya(x) = 2x /(x+1), according to (8) the trial neural form of the solution is taken to be :

yt(x) = x + x (x – 1) N(x, p) .

The FFNN trained using a grid of ten equidistant points in [0, 1]. Figure(3) display the analytic and neural solutions with Levenberg – Marquardt (trainlm) training. The neural results with different types of training algorithm such as : Levenberg – Marquardt (trainlm), conjugate gradient (traincgp) , quasi – Newton (trainbfg) , Bayesian Regulation (trainbr) introduced in table (5) and its errors given in table (6), table(7) gives the weight and bias of the designer network ,table(8) gives the performance of the train with epoch and time .

Ibraheem and Khalaf [7] solve this example by using (integration and interpolation techniques) and Neural Networks and gave the maximum error value is max | yexact - yNN | = 44.3729E-004 and solution time is 3.8750 sec. and the result obtained by the neural network given in figure 4.

8. Conclusion

From the above problems it is clear that the method which proposed can handle effectively ODE 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 method are more precise as compared to neural network suggested in [7].

It is very difficult to know which training algorithm will be the fastest for a given problem. It will depend on many factors including the complexity of the problem, the number of data points in the training set, the number of weights and biases in the FFNN, the error goal, and whether the FFNN is being used for pattern recognition (discriminant analysis) or function approximation (regression).

In general, the practical results on FFNN show which contain up to a few hundred weights the Levenberg-Marquardt algorithm (trainlm) will have the fastest convergence, then trainbr and then trainbfg. However, "trainbr" it does not perform well on function approximation on problems. The "traincg", algorithms have relatively modest memory requirements in particular "traincgp", but the computation required does increase geometrically with the size of the FFNN . The performance of the various algorithms can be affected by the accuracy required of the approximation.

References

[1] I. A.Galushkin, " Neural Networks Theory", Berlin Heidelberg , 2007.

[2] R. M. Hristev , " The ANN Book ", Edition 1, 1998.

[3] T.Villmann, U.Seiffert and A.Wismüller , " Theory and Applications of Neural maps ", ESANN2004 PROCEEDINGS - European Symposium on Ann, pp.25 - 38, April 2004 .

[4] L.N.M.Tawfiq and R.S.Naoum , " On Training of Artificial Neural Networks " , AL-Fath Jornal , No 23, 2005 .

[5] L.N.M.Tawfiq and R.S.Naoum " Density and approximation by using feed forward Artificial neural networks ", Ibn Al-Haitham Journal for Pure & Applied Sciences, Vol. 20 (1) 2007.

[6] A. K. Jabber ," On Training Feed Forward Neural Networks for Approximation Problem ", MSc Thesis, Baghdad University, College of Education (Ibn Al-Haitham), 2009.

[7] K. I. Ibraheem and B. M. Khalaf , Shooting Neural Networks Algorithm for Solving Boundary Value Problems in ODEs , Applications and Applied Mathematics: An International Journal , Vol. 6, Issue 11 , pp. 1927 – 1941, 2011.

Table1: Analytic and Neural solution of example 1

Input	Exact solution	Out of FFNN $y_t(x)$			
x	$y_a(x)$	Trainlm	Trainbfg	Traincgp	Trainbr
0.0	1	1.00000983 089905	1.00000000 166988	0.99745055 5522462	0.99999952 5460578
0.1	1.10517091 807565	1.10517091 807565	1.10517091 844109	1.10461372 177480	1.10522925 669824
0.2	1.22140275 816017	1.22140275 816017	1.22140960 609182	1.22197600 838746	1.22140972 385794

1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3
2.71828182845905	2.45960311115695	2.22554092849247	2.01375270747048	1.82211880039051	1.64872127070013	1.49182469764127	1.34985880757600
2.71828182845905	2.45965082692107	2.22554092849247	2.01374615271598	1.82211880039051	1.64872258180316	1.49182469764127	1.34985880757600
2.71828182955048	2.45960311303682	2.22549240027140	2.01373151851359	1.82211880042970	1.64872127042285	1.49182470075597	1.34986453991830
2.70187395125676	2.45951313281121	2.22811020044383	2.01458575984956	1.82143024037242	1.64792438410036	1.49180213370646	1.35052935864027
2.71828161850524	2.46054572419418	2.22588757540956	2.01375612150624	1.82211148696120	1.64875643044583	1.49183921872639	1.34984175883680

Table2 : Accuracy of solutions for example 1

Deviation $\Delta y(x) = \lvert y_t(x) - y_a(x) \rvert$ w h e r e $y_t(x)$ computed by the following training algorithm		
T rainlm	9.83089905348678e-06	
Trainbfg	1.66987534910845e-09	
T raincgp	0.00254944447753802	
Trainbr	4.74539422312681e-07	

4.44089209850063e-16	4.77157641158854e-05	0	6.55475449207188e-06	2.22044604925031e-16	1.31110302747928e-06	0	4.44089209850063e-16	2.22044604925031e-16	2.22044604925031e-16
1.09143893922692e-09	1.87987447830551e-09	4.85282210678228e-05	2.11889568917378e-05	3.91950916167616e-11	2.77281309024602e-10	3.11470271796566e-09	5.73234229195307e-06	6.84793165306452e-06	3.65442787142456e-10
0.0164078772022869	8.99783457422032e-05	0.00256927195136258	0.000833052379080268	0.000688560018087481	0.000796886599769398	2.25639348054862e-05	0.000670551064267722	0.000573250227294153	0.000557196300851492
2.09953809005015e-07	0.000942613037234086	0.000346646917092741	3.41403576564758e-06	7.31342931303836e-06	3.51597457055597e-05	1.45210851227873e-05	1.70487392048280e-05	6.96569776748035e-06	5.83386225889715e-05

Table3 : the performance of the train with epoch and time

TrainFcn	Performance of train	Epoch	Time
Trainlm	7.75e-32	148	0:00:02
Trainbfg	2.49e-18	282	0:00:05
Traincgp	5.87e-07	48	0:00:01
Trainbr	6.15e-10	396	0:00:06

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

Weights and bias for trainlm		
Net.IW{1,1}	**Net.LW{2,1}**	**Net.B{1}**
0.0521	0.8604	0.5134
0.9312	0.9344	0.1776
0.7287	0.9844	0.3986
0.7378	0.8589	0.1339
0.0634	0.7856	0.0309
Weights and bias for trainbfg		
Net.IW{1,1}	**Net.LW{2,1}**	**Net.B{1}**
0.7112	0.4242	0.0292
0.2217	0.5079	0.9289
0.1174	0.0855	0.7303
0.2967	0.2625	0.4886
0.3188	0.8010	0.5785
Weights and bias for traincgp		
Net.IW{1,1}	**Net.LW{2,1}**	**Net.B{1}**
0.9431	0.8978	0.6511
0.1127	0.4972	0.1336
0.6483	0.7713	0.6385
0.4808	0.0604	0.3849
0.0665	0.2625	0.7657
Weights and bias for trainbr		
Net.IW{1,1}	**Net.LW{2,1}**	**Net.B{1}**
0.9158	0.6153	0.0321
0.1355	0.5831	0.8271
0.3321	0.6983	0.3400
0.8975	0.0293	0.8467
0.4996	0.5279	0.2461

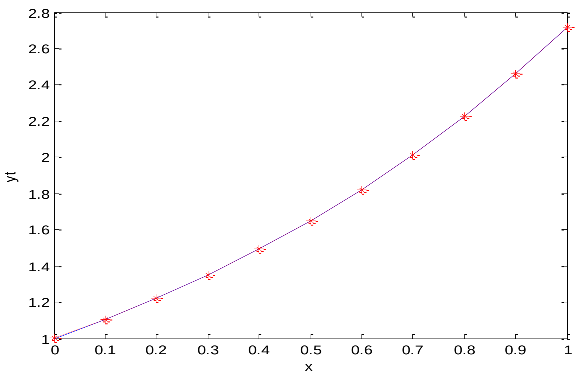

Figure 1: analytic and neural solution of example 1 using : trainlm training algorithm

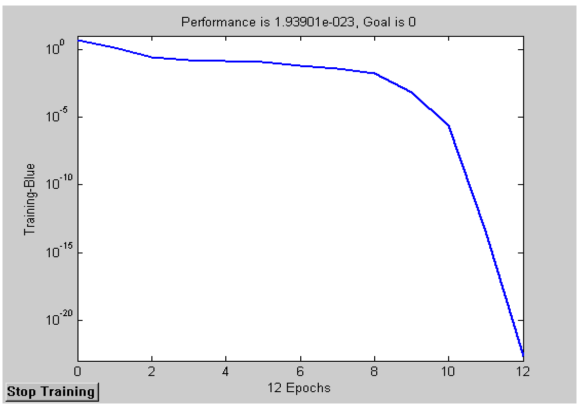

Figure2a. Learning curve of NN gave in [7] for Example 1

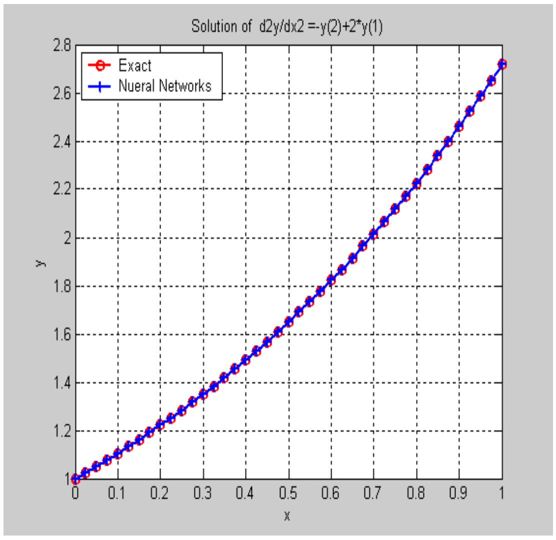

Figure2b. Curve of NN gave in [7] and exact for Example 1

Table5: Analytic and Neural solution of example 2

Input	**Exact solution**	**Out of FFNN $y_t(x)$**			
X	**$y_a(x)$**	**Trainlm**	**Trainbfg**	**Traincgp**	**Trainbr**
0.0	0	0	1.28139499011581 e-10	2.95252630078124 e-05	2.10062376188036 e-06
0.1	0.18181818 18181182	0.18163249 9010170	0.18146081 3389434	0.17981391 8093008	0.18138474 8754504
0.2	0.33333333 3333333	0.33333333 3333333	0.33324812 1125568	0.33298636 1976869	0.33332948 6006840
0.3	0.46153846 1538462	0.46153846 1538462	0.46153846 2505036	0.46214064 4579796	0.46155449 4719676
0.4	0.57142857 1428572	0.57139523 1164368	0.57142857 0525489	0.57191901 6357442	0.57140931 9653527

0.5	0.666666666666667	0.666636873182718	0.666666666283074	0.666695284986192	0.666680409893763
0.6	0.750000000000000	0.749989760186767	0.750000001680590	0.749761550738823	0.749996629680552
0.7	0.823529411764706	0.823529411764706	0.823525879431565	0.823344837511379	0.823412831163550
0.8	0.888888888888889	0.888888888888889	0.888888889258379	0.888895714160738	0.888695332141908
0.9	0.947368421052632	0.947368421052631	0.947379133347466	0.947379790486359	0.947368464573081
1.0	1	1.00000000000000	1.00000000011712	0.999490064878165	1.00068490668473

Table 6 : Accuracy of solutions for example 2

Deviation $\Delta y(x) = \vert y_t(x) - y_a(x) \vert$ where $y_t(x)$ computed by the following training algorithm			
T rainlm	Trainbfg	T raincgp	Trainbr
0	1.28139499011581e-10	2.95252630078124e-05	2.10062376188036e-06
0.000185682808011678	9.03082386649601e-10	0.00200426372517432	0.000433433063678046
0	0.000357368428747840	0.000346971356464498	3.84732649355568e-06
1.11022302462516e-16	8.52122077651396e-05	0.000602183041334026	1.60331812142367e-05
3.33402642036518e-05	9.66574043026469e-10	0.000490444928870515	1.92517750449150e-05
2.97934839488256e-05	3.83592380082121e-10	2.86183195253864e-05	1.37432270962412e-05
1.02398132326709e-05	1.68059044458602e-09	0.000238449261177109	3.37031944841470e-06
1.11022302462516e-16	3.53233314132062e-06	0.000184574253326408	0.000116580601156269
2.22044604925031e-16	3.69490105178727e-10	6.82527184903137e-06	0.000193556746981294
5.55111512312578e-16	1.07122948340699e-05	1.13694337268155e-05	4.35204492443830e-08
3.33066907387547e-16	1.17118315046127e-10	0.000509935121834837	0.000684906684728537

Table 7: Weight and bias of the network for different training algorithm

Weights and bias for trainbfg		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.2040	0.2021	0.7613
0.6241	0.4691	0.4027
0.7252	0.3784	0.6743
0.8344	0.3404	0.5511
0.0189	0.0639	0.0515

Weights and bias for trainlm		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.4884	0.5932	0.0018
0.7290	0.3044	0.7118
0.2026	0.9677	0.8677
0.2163	0.8960	0.1183
0.9763	0.1900	0.0390

Weights and bias for traincgp		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.2039	0.5066	0.7058
0.3867	0.7169	0.1331
0.0650	0.3012	0.6655
0.4323	0.7060	0.3756
0.6897	0.9152	0.5024

Weights and bias for trainbr

Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.4559	0.6603	0.5711
0.2428	0.6805	0.6902
0.0019	0.8506	0.8956
0.6153	0.0373	0.2669
0.6612	0.6808	0.0686

Table 8 : the performance of the train with epoch and time

TrainFcn	Performance of train	Epoch	Time
Trainlm	6.87e-32	530	0:00:08
Trainbfg	6.98e-19	1157	0:00:20
Traincgp	6.48e-08	26	0:00:00
Trainbr	8.47e-10	92	0:00:01

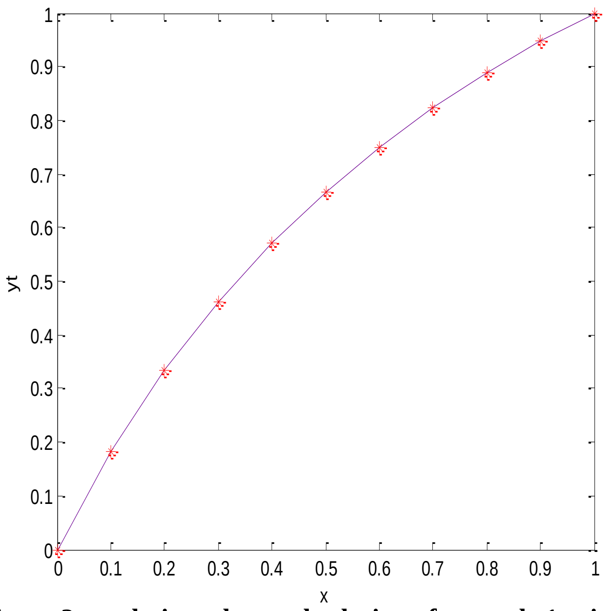

Figure 3: analytic and neural solution of example 1 using : trainlm training algorithm

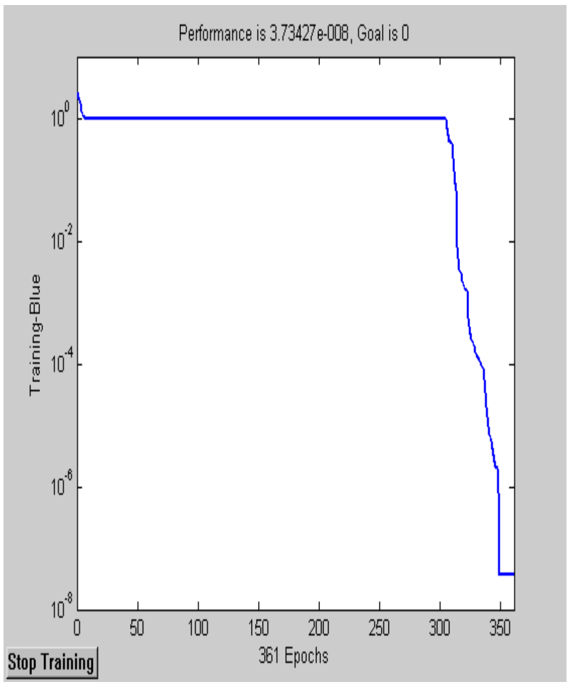

Figure4a. Learning curve of NN for Example 2 gave in [7]

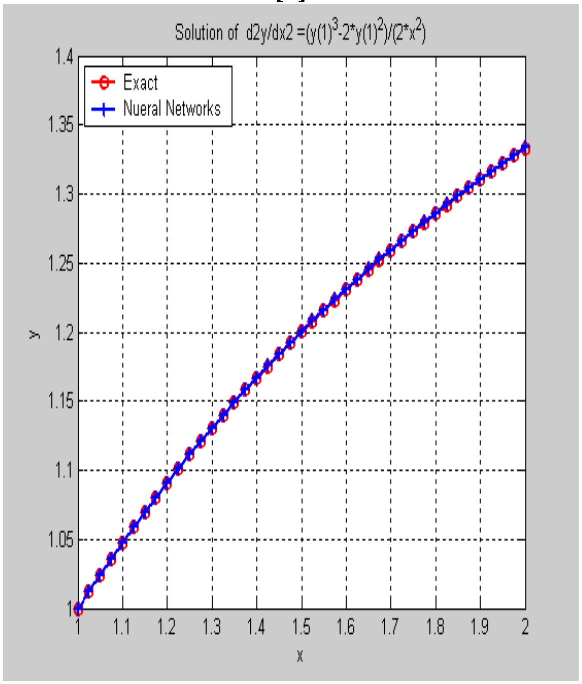

Figure4b. Curve of NN gave in [7] and exact solution for Example 2

P- ISSN 1991-8941 E-ISSN 2706-6703
2013,(7), (1) :149-157

Open Access

تصميم شبكة عصبية ذات تغذية تقدمية لحل مسائل قيم حدودية

لمى ناجي محمد توفيق منى حسين علي

الخلاصة

الهدف من البحث هو تصميم شبكة عصبية ذات تغذية تقدمية تمثل طريقة لحل مسائل قيم حدودية للمعادلات التفاضلية الاعتيادية وهذا يعني تطوير خوارزمية التدريب بحيث تسرع زمن الحل وتقلل من حالات الفشل في الحصول على الحل و تزيد أمكانية الحصول على الحل المثالي الرئيسي واستخدمنا في ذلك عدد من خوارزميات التدريب المختلفة بعضها يمتلك نسبة تقارب سريعة جدا في حالة الشبكات التي تمتلك أحجام معقولة أخيرا وضحنا الطريقة من خلال حل مثالين وقارنا نتائج الشبكة المقترحة مع نتائج المصدر[7] .