Journal of AL-Qadisiyah for computer science and mathematics
Vol.8 No.1 Year 2016

Page 14 - 24

On training Neural Network To Solve Some Class of Boundary Value Problems

Alaa Kamel Jaber
Email: alaa77_math@yahoo.co.uk
Mathematics Department - College of Education - AL-Qadisiyah University

Recived : 11\8\2014 Revised : 30\11\2014 Accepted : 9\2\2016

Abstract

The aim of this paper is to train feed forward neural network for solving some class of boundary value problems for ordinary differential equations. Using backpropagation training algorithm, where the Levenberg – Marquardt training algorithm is used to train the network .The existence of the proposed solution was proved. The suggested networks have been studied intensively for a few decades and have provided an option for modeling complex systems. Therefore this option was utilized to reduce the computation of solution, and finally the method is demonstrated through illustrative examples.

Keywords : Differential equations, Boundary value problems, Feed Forward Neural Network.

Mathematics subject classification : 34B45, 65L10, 92B20.

1- Introduction

Artificial neural networks have been studied over the last decades and are an excellent option for modeling complex systems. The study of solving differential equations using artificial neural network (Ann) was initiated by Agatonovic-Kustrin and Beresford in [1]. Lagaris et al. in [8] 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 [15] 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 [10] 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 etal. [2] 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 [11] 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. [13] suggested the solution of the matrix Riccati differential equation (MRDE) for nonlinear singular system using Ann. Ibraheem and Khalaf [5] 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 [4] described a numerical solution with neural networks for solving PDE, with mixed boundary conditions. Majidzadeh [9] 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 [3] 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.

Alaa. K

The direct problem under consideration consists of the following boundary value problem :-

$$y^{(n)}=F(x,y,y',\ldots,y^{(n-1)}) \quad \ldots\ldots\ldots\ldots\ldots\ldots..(1)$$

Where $x\in[a,b]$ with BC $\{y(a)=A\ ,\ y(b)=B\}$, We wont to approximate y(x) in (1) by using FFNN.

2- Artificial Neural Network [16]

An 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. Ann have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

i. Information processing occurs at many simple elements called neurons that is fundamental to the operation of Ann's.
ii. Signals are passed between neurons over connection links.
iii. Each connection link has an associated weight which, in a typical neural net, multiplies the signal transmitted.
iv. Each neuron applies an action 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, shown as lines in a diagram .

Ann is characterized by:

i. Architecture: its pattern of connections between the neurons.
ii. Training algorithm : its method of determining the weights on the connections.
iii. 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. This view is motivated by the fact that the weights in a net contain extremely important information.

2.1- Multilayer Feed Forward Neural Network Architecture [16]

In a layered neural network the neurons are organized in the form of layers. We have at least two layers: an input and an outputlayer. The layers between the input and the output layer (if any) are called hidden layers, whose computation nodes are correspondingly called hidden neuronsor hidden units. Extra hidden neurons raise the network's ability to extract higher-order statistics from (input) data.

The Ann is said to be fullyconnectedin 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 .

2.2- Training Feed Forward Neural Network [16]

Training is the process of adjusting connection weights w and biases b. In the first step, the network outputs and the difference between the actual (obtained) output and the desired (target) output (i.e., the error) is calculated for the initialized weights and biases (arbitrary values). During the second stage, the initialized weights in all links and biases in all neurons are adjusted to minimize the error by propagating the error backwards (the back propagation algorithm). The network outputs and the error are calculated again with the adapted weights and biases, and the process (the training of the Ann) is repeated at each epoch until a satisfied output y_k(corresponding to the values of the input variables x) is obtained and the error is acceptably small. In most of the training algorithms a learning rate is used to determine the length of the weight update (step size) .

3- Solving The Method

To illustrate the method we will write the approximate solution as[15] :

$$y_t(x)\,\frac{x-a}{b-a}B\,\frac{b-x}{b-a}A(x-a)(x-b)N(x)$$

$$\ldots\ldots\ldots\ldots\ldots\ldots..(2)$$

Where $N(x)$ is the output of a FFNN with one input units for x.

It's clear that y satisfy the BC of (1).

The our goal in this paper is to design a FFNN $N(x)$ such that y_t Fit Informatics unknown function y(x) in any accuracy .

Now rewrite (2) to be as following:-

$$(x) = \frac{y_t(x) - \frac{x-a}{b-a}B - \frac{b-x}{b-a}A}{(x-a)(x-b)} \quad , \ x \neq a,b \quad \ldots\ldots(3)$$

Then the right side of (3) is unknown function of one variable , denoted by G(x), and the needed FFNN in (2) is the same network required to approximate the function y(x),

which means that problem (1) has been converted from differential equation problem to approximation function problem by FFNN, which will discuss in the next section.

4- The Existence [6]

One of the earliest works on FFNN with ridge activation functions is in Hecht-Nielson. The author used an improved version of Kolmogorov's theorem due to Sprecher which states that:

Every continuous function $f:[0,1]^N \rightarrow R$ can be written as:

$$f(x)= \sum_{h=1}^{2N+1} \phi_h \left(\sum_{k=1}^{N} \lambda^h \psi(x_k + \varepsilon h) + h \right) \quad \ldots\ldots\ldots(4)$$

where the real λ and the continuous monotonically increasing function ψ are independent of f, the constant ε is a positive number and the continuous function φ_h, $1 \leq h \leq 2N+1$, depends on f. This equation can be interpreted as a three-layered neural network where the h^{th} hidden node computes the function

$$z_h = \sum_{k=1}^{N} \lambda^h \psi(x_k + \varepsilon h) + h ,$$

and the output nodes compute $\sum_{h=1}^{2N+1} \phi_h(z_h)$. However, this is not the network architecture commonly used in practice.

One of the most elegant approaches to prove universal approximation is proposed by Cybenko. By using the Hahn-Banach Theorem and the Riesz Representation Theorem, he showed that if the ridge activation function, σ, is a continuous sigmoid, then the set of $\sum_{i=1}^{N} C_i \sigma(\theta_i^T x + b_i)$ is dense in C(K), where K is a compact set of R^N, with respect to uniform norm.

Later, his approach was adopted by many authors to prove their results.

Chui and Li adopted another approach to prove universal approximation. They showed that if the ridge activation function σ, is continuous sigmoid and the direction vector θ satisfies some interpolation conditions, then the set of $\sum_{i=1}^{N} c_i \sigma(\theta_i^T x + b_i)$ is dense in C(K) with respect to uniform norm. They constructed their proof by showing that it is possible to realize *polynomials* as a sum of ridge activation functions.

Since *polynomials* are dense in $C(R^N)$, it follows that the three-layered neural networks are dense in $C(R^N)$ with respect to uniform norm.

In Chen et. al., showed that the continuity assumption usually imposed on the sigmoid functions is unnecessary. Instead, they proved that if the ridge activation function σ, is a bounded sigmoid, then the set of $\sum_{i=1}^{N} C_i \sigma(\theta_i^T x + b_i)$ is dense in C(K) with respect to uniform norm. They also pointed out that in order to prove the neural network in the n-dimensional case, all one needs to do is to prove the case for one dimension .

In Hornik, the author adopted Cybenko's approach to prove universal approximation. He showed the sigmoid assumption usually imposed on the ridge function is unnecessary. Instead, he proved that if the ridge activation function σ, is continuous, bounded and non-constant, then the set of $\sum_{i=1}^{N} C_i \sigma(\theta_i^T x + b_i)$ is dense in C(K) with respect to uniform norm. At the same time, he proved that if the ridge activation function σ, is bounded and non-constant, then the set of $\sum_{i=1}^{N} C_i \sigma(\theta_i^T x + b_i)$ is dense in $L_p(\mu)$ with respect to L_p norm for $1 \leq p < \infty$ and a finite measure μ.

Leshno et. Al. provides one of the most general results. They showed that if the ridge activation function σ, is continuous almost everywhere, locally essentially

bounded, and not a *polynomial*, then the set of $\sum_{i=1}^{N} c_i \sigma(\theta_i^T x+b_i)$ is dense in C(K) with respect to uniform norm.

4.1- Theorem [6]

Standard Feed Forward Networks with only a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function to any desired degree of accuracy.

Therefore from the above theorem we have the following:-

1- To approximate any function on R^N we want to determine the number of the hidden nodes , activation functions to hidden layer and training functions.

2- The parameters to this approximation are the weights and biases of nodes in the layers which can calculate by training the FFNN.

3- Any lack of success in applications must arise from inadequate training, insufficient number of hidden units, or the lack of a deterministic relationship between the input and the target.

5- Examples

Now in this section we give some example which illustrate the suggested network.

5.1 Example 1

Let us have the following differential equations

$y''-4(y-x)=0$, $x\in[0,1]$ …… (5)

with BC: $y(0)=0$ and $y(1)=2$

And the exact solution to (5) is

$y(x)=e^2(e^4-1)^{-1}(e^{2x}-e^{-2x})+x$ ……(6)

Our solution to equation (5) by using (2) is

$y_t(x)=2x+(x^2-x)N(x)$ ……(7)

Then the FFNN $N(x)$ in (7) is the same as to approximate the following function :-

$G(x)=\frac{y_t(x)-2x}{x^2-x}$ x≠0,1 …… (8)

To design FFNN which approximate G(x) by Theorem 4.1, we choose 3 nodes to hidden layer and the activation function is *'tansig'*, then to training the FFNN we use the function *'trainlm'*.

After we training the FFNN we obtain the parameters illustrated in Table1, the result is giving in Table 2 and we display the analytic and neural solutions in the Figure 1.

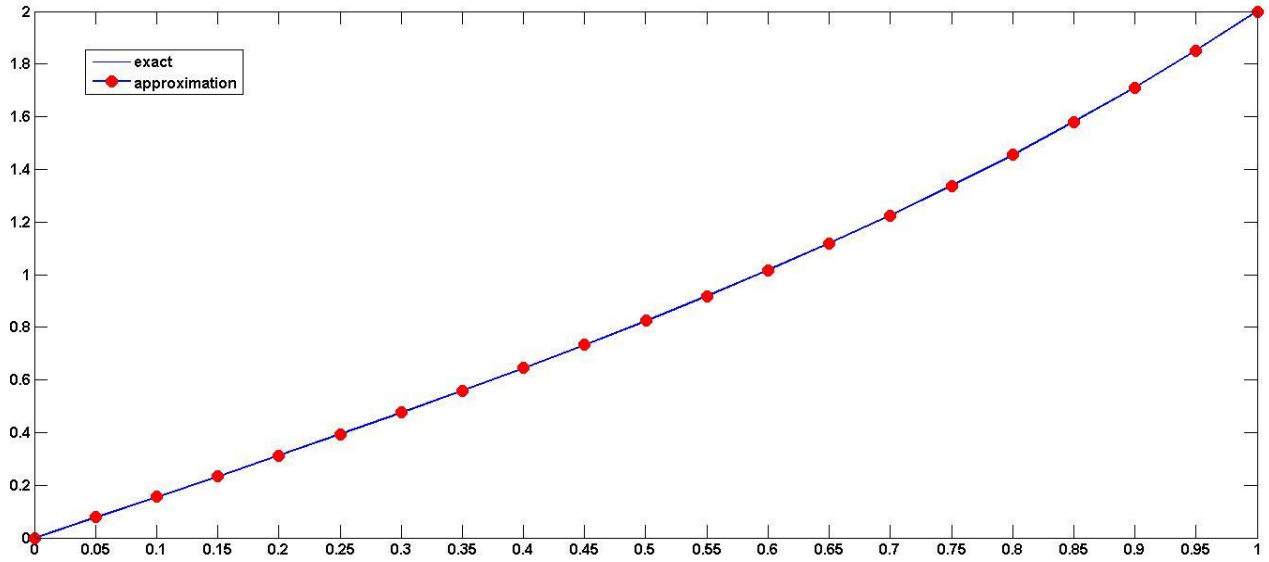
Table1: Training parameter of the suggested network for example 1

Input Weight	Hidden Bias	Hidden Weight	Output Bias
0.025118602453792	0.171440917519520	0.797663289393945	1.153586732255470
0.596614946259474	-1.333818433845600	0.791987233939725	
0.187536230762526	0.117995335006529	0.885756736525519	

Table 2: The results of the example1 using suggested network

X	Y-exact	Y-approximate	$(Y-Y_t)^2$
0	0	0	0
0.1	0.155512476	0.1555155	0.00000000000914490185
0.2	0.313252863	0.31325665	0.00000000001434217580
0.3	0.475538485	0.475536608	0.00000000000352482217
0.4	0.644869083	0.644866155	0.00000000000857073793
0.5	0.824027137	0.824030907	0.00000000001421507743
0.6	1.016189537	1.016199017	0.00000000008986681044
0.7	1.225055085	1.225060292	0.00000000002710271818
0.8	1.454992938	1.454990613	0.00000000000540431067
0.9	1.711217956	1.711224143	0.00000000003827465872
1	2	2	0
		MSE	0.00000000001721832653

Figure 1:Analytic and neural solutions of Example 1



5.2 Example 2

Let us have the following differential equations

$y''-4y=8\cos 2x$, $x\in[0, 2\pi]$ ……..…. (9)

BC: $y(0)=1$, $y(2\pi)=1$

And the exact solution to (9) is

$y(x)=\cos 2x+\sin 2x+2x\sin 2x$……. (10)

Our solution to equation (9) by using (2) is

$y_t(x)=1+(x^2-2\pi x)N(x)$ ……..(11)

Then the FFNN $N(x)$ in (11) is the same as to approximate the following function :-

$$G(x)=\frac{y_t(x)-1}{x^2-2\pi x} \quad , \quad x\neq 0,2\pi \quad \ldots\ldots\ldots\ldots(12)$$

To design FFNN which approximate G(x) by Theorem 4.1, we choose 3 nodes to hidden layer and the activation function is *'tansig'*, then to training the FFNN we use the function *'trainlm'*.

After we training the FFNN we obtain the parameters illustrated in Table 3, the result is giving in Table 4 and we display the analytic and neural solutions in the Figure 2.

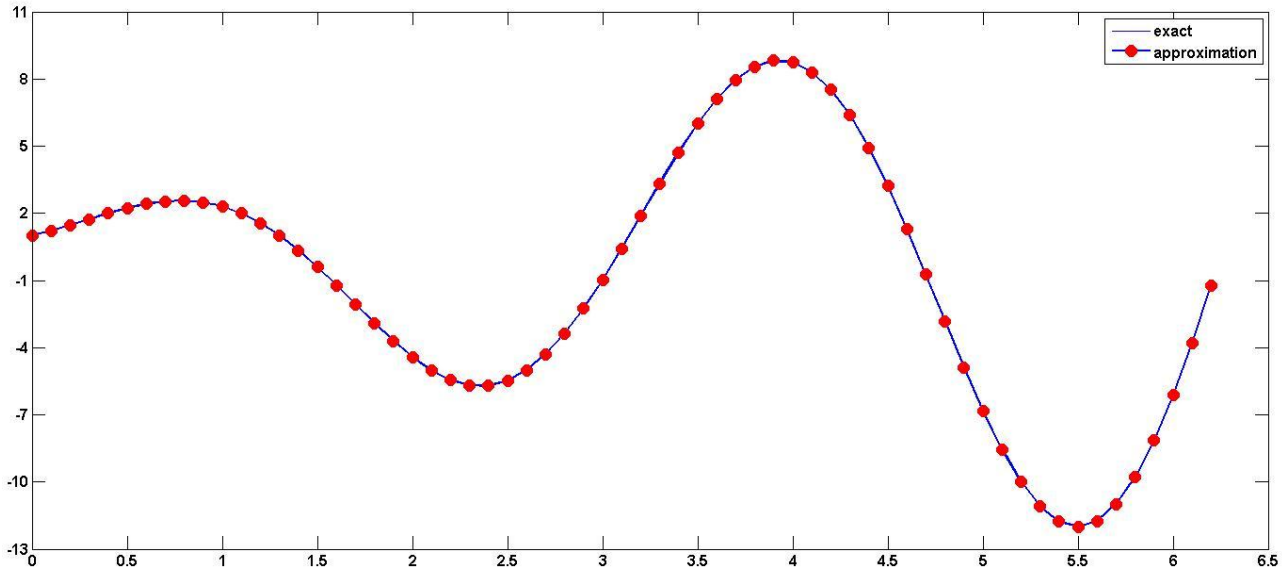
Table3: Training parameter of the suggested network for example 2

Input Weight	Hidden Bias	Hidden Weight	Output Bias
-0.96288232488637	0.652840912117375	6.23664109302719	-0.537041239957662
1.89750554327429	0.762014152249968	0.944806411478224	
-1.597958760677	1.091948365944060	-4.375547392315350	

Table 4: The results of the example2 using suggested network

X	Y-exact	Y-approximate	$(Y-Y_t)^2$
0	1	1	0
0.65	2.483137499	2.483682655	0.00000029719461491706
1.3	1.006661952	0.998916185	0.00005999690605684330
1.95	-4.091058127	-4.095986484	0.00002428870080022000
2.6	-5.013702487	-5.008902194	0.00002304280965994200
3.25	2.597896663	2.589987536	0.00006255428884933510
3.9	8.815226652	8.84113686	0.00067133884744063800
4.55	2.289918648	2.275171858	0.00021746783892625600
5.2	-10.00609845	-9.998206005	0.00006229072184669800
5.85	-9.025219534	-9.029595177	0.00019146248909700 20
2π	1	1	0
		MSE	0.00009330738194491770

Figure 2:Analytic and neural solutions of Example 2



5.3 Example 3

Let us have the following differential equations

$y'' = y^3 - yy'$, $x \in [0,1]$ ……..(13)

with BC: $y(0)=1$ and $y(1)=\frac{1}{2}$

And the exact solution to (13) is

$y(x)=\frac{1}{x+1}$ ……(14)

Our solution to equation (13) by using (2) is :

$y_t(x)=\frac{2-x}{2}+(x^2-x)N(x)$ ……(15)

Then the FFNN $N(x)$ in (15) is the same as to approximate the following function :-

$G(x) = \frac{2y_t(x)-2-x}{2(x^2-x)}$, x≠0,1

……(16)

To design FFNN which approximate G(x) by Theorem 4.1, we choose 3 nodes to hidden layer and the activation function is *'tansig'*, then to training the FFNN we use the function *'trainlm'*.

After we training the FFNN we obtain the parameters illustrated in Table5, the result is giving in Table 6 and we display the analytic and neural solutions in the Figure 3

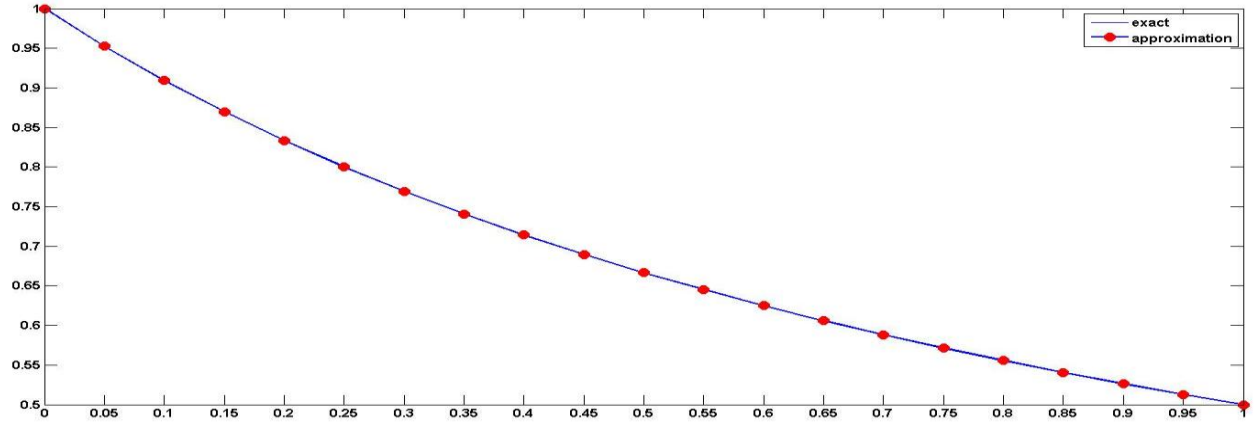
Table5: Training parameter of the suggested network for example 3

Input Weight	Hidden Bias	Hidden Weight	Output Bias
-0.20011524470189	0.747097442730006	0.374611207364815	-0.226032037045629
-1.1102786850145	-2.10935734110926	1.02228340776327	
0.427525944912668	0.663165049634819	-1.18051725183527	

Table 6: The results of the example3 using suggested network

X	Y-exact	Y-approximate	$(Y-Y_t)^2$
0	1	1	0
0.1	0.909088244	0.909090909	0.00000000000710520078
0.2	0.833333307	0.833333333	0.00000000000000070811
0.3	0.769233234	0.769230769	0.00000000000607552435
0.4	0.71428537	0.714285714	0.00000000000011866059
0.5	0.666664177	0.666666667	0.00000000000619845573
0.6	0.624999089	0.625	0.00000000000083049139
0.7	0.588236461	0.588235294	0.00000000000136085788
0.8	0.555555102	0.555555556	0.00000000000020588373
0.9	0.526311433	0.526315789	0.00000000001897858118
1	0.5	0.5	0
		MSE	0.00000000000334426612

Figure 3:Analytic and neural solutions of Example 3



5.4 Example 4

Let us have the following differential equations

$2x^2 y''-y^3+2y^2=0$, $x\in[0,1]$ ……...(17)

with BC: y(0)=0 and y(1)=1

And the exact solution to (17) is

$y(x)=\frac{2x}{x+1}$ ……..(18)

Our solution to equation (17) by using (2) is :

$y_t(x)=x+(x^2-x)N(x)$ ……(19)

Then the FFNN $N(x)$ in (19) is the same as to approximate the following function :-

$G(x)=\frac{y_t(x)-x}{x^2-x}$ x≠0,1 ……(20)

To design FFNN which approximate G(x) by Theorem 4.1, we choose 3 nodes to hidden layer and the activation function is *'tansig'*, then to training the FFNN we use the function *'trainlm'*.
After we training the FFNN we obtain the parameters illustrated in Table7, the result is giving in Table 8 and we display the analytic and neural solutions in the Figure 4.

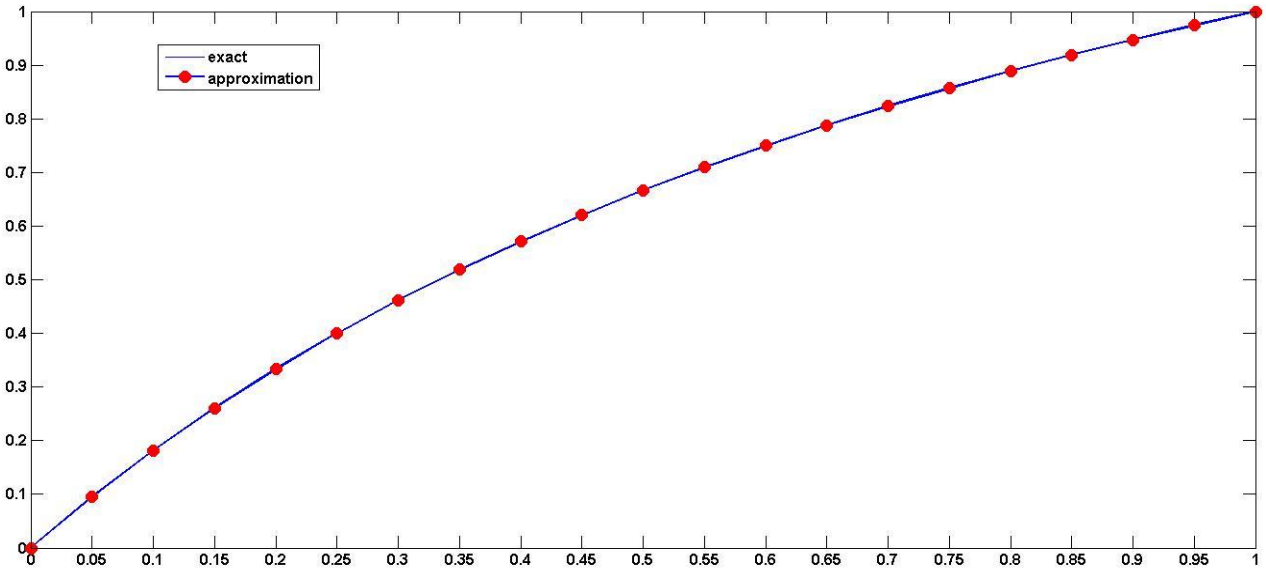
Table7: Training parameter of the suggested network for example 4

Input Weight	Hidden Bias	Hidden Weight	Output Bias
0.528120570989692	-0.137703111821045	0.445604498360927	-2.230706087313750
0.895588508943409	1.388813866032050	1.065113059207930	
-0.00562960643696	1.002926456060790	-1.28734987876764	

Table 8: The results of the example4 using suggested network

X	Y-exact	Y-approximate	$(Y-Y_t)^2$
0	0	0	0
0.1	0.181758368	0.181818182	0.00000000357773814096
0.2	0.333386397	0.333333333	0.00000000281577701670
0.3	0.461580499	0.461538462	0.00000000176714350830
0.4	0.571432543	0.571428571	0.00000000001577714927
0.5	0.666662563	0.666666667	0.00000000001683997884
0.6	0.750002519	0.75	0.00000000000634739672
0.7	0.823530716	0.823529412	0.00000000000170181536
0.8	0.888886141	0.888888889	0.00000000000755137177
0.9	0.947373363	0.947368421	0.00000000002442290931
1	1	1	0
		MSE	0.00000000067363357805

Figure 4:Analytic and neural solutions of Example 4



6- References

[1] Agatonovic-Kustrin and R. Beresford, (2000), "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.

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

[3] Ali M. H., (2012), "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.

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

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

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

[7] Junaid A., M. A. Z. Raja, and I. M. Qureshi, (2009), "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.

[8] Lagaris I. E., A. C. Likas, and D. G. Papageorgiou, (2004), "Neural network methods for boundary value problems with irregular boundaries", IEEE Transactions on Neural Networks, vol. 11, no.5, pp. 1041–1049, 2000.

[9] Majidzadeh K., (2011), "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.

[10] Malek A. and R. Shekari Beidokhti, (2006), "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.

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

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

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

[14] Tawfiq Luma N. M., Hussein Ashraf A. T., (2013), " Design Feed Forward Neural Network to Solve Singular Boundary Value Problems", ISRN Applied Mathematics, Volume 2013, Article ID 650467, 7 pages.

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

[16] Tawfiq Luma N. M., Oraibi Yaseen A., (2013), "Fast Training Algorithms for Feed Forward Neural Networks", Ibn Al-Haitham Journal for Pure and Applied Science , NO. 1, Vol. 26:275-280.

حول تدريب الشبكات العصبية لحل صنف من مسائل القيم الحدودية

م. علاء كامل جابر

Email: alaa77_math@yahoo.co.uk

جامعة القادسية / كلية التربية / قسم الرياضيات

المستخلص :

الهدف من البحث هو تدريب شبكة عصبية ذات تغذية تقدمية لحل صنف من مسائل القيم الحدودية للمعادلات التفاضلية الاعتيادية .تم استخدام خوارزمية التدريب ذات التغذية الخلفية وتم استخدام خوارزمية التدريب (*trainlm*). تم اثبات وجود الحل للشبكة المقترحة. تم دراستها بشكل مكثف منذ بضعة عقود حيث قدمت خيارا لنمذجة الانظمة الصعبة. لذلك هذا الخيار استخدم لتقليل الحسابات في اثناء الحل ، واخيرا تم توضيح الشبكة المقترح .