The effect of the threshold function for design feed forward neural network for solving Partial differential equations

Dr.Khalid. Mindeel. M. Al-Abrahemee

Department of Mathematics, College of Education / University of AL-Qadisiyha

Dywaneia-Iraq

(Email: Khalid.mohammed@qu.ed)

Abstract--- *In this paper we disperse the outcome of threshold functions for designate feed forward neural network for solution partial differential equations. We utility a multi-layer network having one hidden layer with 7 hidden units (neurons) and one linear output unit with different of threshold function of each unit are logsig , tansig, purelin, tribas and hardlim and use Levenberg – Marquardt (trainlm) training algorithmic rule. Finally the terminate of numerical experience are compare to with the true solution in illustrative examples to ratify the precision and effectiveness of the immediate plan.*

Keywords--- partial differential equations, Artificial neural network, feed forward neural network, Levenbrg-Marquardt training.

1. INTRODUCTION

The approach capabilities of feed forward neural networks with a single hidden layer has been learned by many authors, e.g., [1, 2, 3]. In [4], we have shown that such a network using practically any nonlinear activation function can approaching any continuous function of any number of real variables on any nonlinear threshold function can approaching any continuous function of any number of real variables on any compact set to any desired degree of accuracy.Neural networks of an artificial nature are an exciting kind of artificial intelligence; they are unequaled in itself. Neural networks imitative the learning procedure of the human brain in order to extraction patterns from historic data [5]. For a big many of years this new typify of expertise has been efficaciously employment to a difference of real-world problems [6]. The first introduction of perceptrons was intend by Rosenblatt [7]. Simple perceptrons are supervised networks; in the casing that is largely unsupervised, the network fully acclimatizes to its inputs and additional itself conformably. These networks can be taught to discriminate construction and archetype from their input. Multi-layered perceptrons (those that have more than three layers) utility multiple hidden layers.

Differential equations are utility as a efficacious tool in solution many problems in different fields of human knowledge, such as physics, chemistry, dynamics, economics, etc. One application of the differential equation is turn problems and essential phenomena into differential equations, then, by solution the differential equation the answer is relate and the phenomena are calculated. Usually many of these problems do not have analytical solutions or their solutions may have certain implications. Many researchers have tried to approaching the solutions of these equations and intend a lot of algorithms such as Runge-Kotta, Adomian, Adam- Beshforth, Adam-Molton, Predictor-Corrector methods and other methods. In novel years neural networks for calculation of ordinary differential equations (ODE) and partial differential equations (PDE) as well as the fuzzy differential equation (FDEs) have been utility. In 1990, Lee and Kang [8] used parallel processor computers to solution a first order differential equation with Hopfield neural network models. Meade and Fernandez [9, 10] solved linear and nonlinear ordinary differential equations using feed forward neural networks architecture and B1 - splines. It is not easy to extend these techniques to multidimensional domains [11]. Lagaris and colleagues (1998) used artificial neural networks to solve ordinary differential equations (ODE s) and partial differential equations (PDE) with the initial value and boundary value problems [12].

2. ARTIFICIAL NEURAL NETWORK

An Artificial Neural Network [13] is a computational design inhaled in the functioning of the human brain. It is calm by a set of artificial neurons (assumed as procedure units) that are interrelated with other neurons. Each junction has a weight accompanying that personate the control from one neuron on the other. The first constituent model of a artificial neuron was design in 1943 by McCulloch and Pitts. They show that such model was competent to fulfill the calculation of any countable office using a bounded numerousness of affected neurons and synaptic weights adjustable. The talk network in Neural Network suggest to the interconnection between neurons immediate in different layers of a system. Every system is basically a 3 layered system, which are Input layer, Hidden Layer and Output Layer. The input layer has input neurons which give data via synapses to the hidden layer, and likewise the hidden layer transfers this data to the output layer via more synapses. The synapses plenty values appeal to weights which helps them to handle the input and output to different layers. An ANN can be explain based on the following three characteristics:

1. The Architecture show the number of layers and the no. of nodes in each of the layers. 2. The learning machinery refer for updating the weights ofthe connections.3. The activation functions used in various layers.

3. ACTIVATION FUNCTION TYPES [14]

The most necessary unit in neural network construction is their net inputs by worn a scalar-to-scalar function called "the activation function or threshold function or transfer function", output a result value name the "unit's activation". An activation function for termination the breadth of the output of a neuron. Some of the most commonly used activation functions are to solve non-linear problems. These functions are: Uni-polar sigmoid, Bi-polar sigmoid, Tanh, Conic Section, and Radial Bases Function (RBF). We did not solicitude around some activation function such as identity function, step function or binary step functions as they are not used to solve linear problems.

4. LEVENBERG-MARQUARDT ALGORITHM (LM) [15]

For LM algorithm, the performance index to be optimized is defined

$$E(w) = \sum_{P=1}^{P}[\sum_{k=1}^{K}(d_{KP} - O_{KP})^2] \quad (1).$$

Where $w = [w_1\, w_2 \dots\dots w_N]$ consists of all weights of the network, d_{kp} is the desired value of the k^{th} output and the p^{th}pattern, o_{th} is the actual value of the k^{th} output and the p^{th} pattern, N is the number of the weights, P is the number of pattern, and K is the number of the network output.

Equation (1) can be written its

$$E(w) = E^T E \quad (2).$$

Where

$$E = [e_{11} \dots . e_{K1} e_{12} \dots \dots . e_{K2} \dots . e_{1p} \dots . . e_{KP}]$$

$$e_{kp} = d_{kp} - o_{kp} \qquad k = 1, \dots \dots . , K \qquad p = 1, \dots \dots \dots , P$$

Where E is the cumulative error vector (for all pattern) from equation (2) the jacobian matrix is define as=

$$\begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \dots & \frac{\partial e_{1,1}}{\partial w_N} \\ \frac{\partial e_{2,1}}{\partial w_1} & \frac{\partial e_{2,1}}{\partial w_2} & \dots & \frac{\partial e_{2,1}}{\partial w_N} \\ \dots & \dots & \dots. & \dots \\ \frac{\partial e_{k,1}}{\partial w_1} & \frac{\partial e_{k,1}}{\partial w_2} & \dots & \frac{\partial e_{k,1}}{\partial w_N} \\ \dots & \dots & \dots & \dots \\ \frac{\partial e_{1,p}}{\partial w_1} & \frac{\partial e_{1,p}}{\partial w_2} & \dots & \frac{\partial e_{1,p}}{\partial w_N} \\ \frac{\partial e_{2,p}}{\partial w_1} & \frac{\partial e_{2,p}}{\partial w_2} & \dots & \frac{\partial e_{2,p}}{\partial w_N} \\ \dots & \dots & \dots & \dots \\ \frac{\partial e_{k,p}}{\partial w_1} & \frac{\partial e_{k,p}}{\partial w_2} & \dots & \frac{\partial e_{k,p}}{\partial w_N} \end{bmatrix} \quad (3).$$

And the weights are calculated using the following equation

$$w_{t+1} = w_t - (J_t^T J_t)^{-1} J_t E_t \quad (4).$$

Where I is identity unit matrix, μ is the learing parameter and J is jacobian of m output error with respect to n weights of the neural network. The μ parameter is automatically adjusted at each iteration in order to secure convergence, the LM algorithm

requires computation of the jacobian J matrix at each iteration step and the inversion of $J^T J$ square matrix, the dimension of which is N*N.

5. ILLUSTRATION OF THE DESIGN

We will consider a two - dimensional of partial differential equation (P.D.E).

$$\frac{\partial^2 \Psi(x,y)}{\partial x^2} = f(\frac{\partial \Psi(x,y)}{\partial x}, \frac{\partial \Psi(x,y)}{\partial y}, \frac{\partial^2 \Psi(x,y)}{\partial y^2}, \frac{\partial^2 \Psi(x,y)}{\partial x \partial y}, x, y) \quad (5)$$

x ∈ [0 ,1] , y ∈ [0 ,1] with Dirichlet BC:

$$\Psi(0,y) = f_0(y), \Psi(1,y) = f_1(y), \Psi(x,0) = g_0(x) and\ \Psi(x,1) = g_1(x)$$

$, where\ f_0, f_1, g_0\ and\ g_1$ are continuous function. The trial solution is written as

$$\Psi_t(x,y) = A(x,y) + x(1-x)\, y(1-y)\, N(x,y,\vec{p}) \quad (6)$$

where A(x ,y) is chosen so as to satisfy the BC, namely:

$$A(x,y) = (1-x) f_0(y) + x f_1(y) + (1-y)\{g_0(x) - [(1-x) g_0(0) + x g_0(1)]\} + y\{g_1(x) - [(1-x) g_1(0) + x g_1(1)]\} \quad (7).$$

For mixed boundary conditions of the form: $\Psi(0,y) = f_0(y), \Psi(1,y) = f_1(y), \Psi(x,0) = g_0(x)\ and\ (\partial \Psi(x,1)/\partial y) = g1(x)$ (i.e., Dirichlet on part of the boundary and Neumann elsewhere), the trial solution can be written as

$$\Psi_t(x,y) = B(x,y) + x(1-x) y[N(x,y,\vec{p}) - N(x,1,\vec{p}) - [\frac{\partial N(x,1,\vec{p})}{\partial y}] \quad (8).$$

And B(x ,y) is again chosen so as to satisfy the BC's:

$$B(x,y) = (1-x) f_0(y) + x f_1(y) + g_0(x) - [(1-x) g_0(0) + x g_0(1)] + y\{g1(x) - [(1-x) g1(0) + x g1(1)]\} \quad (9)$$

Note that the second term of the trial solution does not affect the boundary conditions.

In all the above PDE problems the error that should be minimized is given by:

$$E[\vec{p}] = \sum_{i=1}^{n} \{\frac{\partial^2 \Psi(x,y)}{\partial x^2} + \frac{\partial^2 \Psi(x,y)}{\partial y^2} - f(\frac{\partial \Psi(x,y)}{\partial x}, \frac{\partial \Psi(x,y)}{\partial y}, x, y)$$

Where(x_i, y_i) are points in [0,1] × [0,1] [10].

6. THE EFFECT OF THE ACTIVATION FUNCTIONS FOR DESIGN FFNN

A harmless problems are utility as an exemplify has to beSolving by design a FFNN. A neural network is created with an input, 7 hidden units with distinct typify ofactivation functions such as tansig , logsig , purelin, tribas and hardlim and a linear output unit and the neural results with Levenberg- Marquardt algorithm (trainlm) . Through these examples, we concluded that the activation function are better than the stay ofthe activation functions in terms of the approximate to the exact solution and also those functions need to be less time down to the target function.

7. NUMERICAL EXAMPLES

In the part we describe some numerical terminate and the solution of a number of standard problems of PDE. In all casing we usage a three-layer FFNN possession two input units, one hidden layer with 7 hidden units (neurons) and one output unit, and we usage different types of activation functions such as , logsig , purelin, tribas and hardlim of each hidden units and we disperse the result of these typify on neural network design. For each test problem, the analytical solution $\Psi_a(\vec{x})$ was known in advance, therefore we test the accuracy of the obtained solutions by computing the deviation:

$$\Delta\Psi(\vec{x}) = |\Psi_t(\vec{x}) - \Psi_a(\vec{x})| \quad (11)$$

Example 1[16]

"Consider the following 2nd order of two-dimensional of partial differential equation for the function $U(x,y)$: $\frac{\partial^2 U(x,y)}{\partial x^2} + \frac{\partial^2 U(x,y)}{\partial y^2} = 0$ with BC′s (Dirishlit case)":

$$U(0,y) = 0, U(1,y) = 0, U(x,0) = 0, U(x,1) = \sin(\pi x)$$

, the equation has the analytic solution $U_a(x,y) = \sin(pix)\sin(piy)/\sinh(pi)$.

The ANN trail using a grid of ten equidistant points in x∈[0,1] , y∈[0,1] . Figure (1) expand the analytic and neural solutions with different types of activation functions with the neural results with dissimilar types of activation functions with Levenberg-Marquardt algorithmic rule (trainlm) and performance of the trail with epoch and time insert in table (1) , table (2) gave the initial weight and bias of the design network.

Example 2 [17]

Consider the following 2nd order of two-dimensional of nonlinear partial differential equation for the function $U(x,y)$.

$$\nabla^2 U(x,y) = e^{-x}(x - 2 + y^3 + 6y)$$

$with\ x,y\ \in [0,1]$ and BC′s (Dirishlit case)

$\mathrm{U}(0,\mathrm{y}) = y^3$ $\mathrm{U}(1,\mathrm{y}) = (1+\mathrm{y}^3)\mathrm{e}^{-1}$, $\mathrm{U}(\mathrm{x},0) = \mathrm{x}\mathrm{e}^{-\mathrm{x}}$, $\mathrm{U}(\mathrm{x},1) = \mathrm{e}^{-\mathrm{x}}(\mathrm{x}+1)$,
the equation has the analytic solution $U_a(x,y) = e^{-x}(x+y^3)$.

The ANN trail using a grid of ten equidistant points in x∈[0,1] , y∈[0,1] . Figure (2) expand the analytic and neural solutions with distinct types of activation functions. The neural results with separate typify of activation functions with Levenberg-Marquardt algorithmic rule (trainlm) and performance of the trail with epoch and time insert in table (3) , table (4) gave the initial weight and bias of the design network.

Example 3 [16]

Consider the following 2nd order of two-dimensional of partial differential equation for the function

$U(x,y)$: $\nabla^2 U(x,y) + U(x,y)\frac{\partial U(x,y)}{\partial y} = \sin(\pi x)\,(2 - \pi^2 y^2 + 2y^3\sin(\pi x))$; $x \in [0,1]$, $y \in [0,1]$ with BC′s (Dirishlit case): $\mathrm{U}(0,\mathrm{y}) = 0$, $\mathrm{U}(1,\mathrm{y}) = 0$, $\mathrm{U}(\mathrm{x},0) = 0, \frac{\partial \mathrm{U}(\mathrm{x},1)}{\partial \mathrm{y}} = 2\sin(\pi \mathrm{x})$, the equation has the analytic solution $U_a(x,y) = y^2\sin(\pi x)$.

The ANN trail using a grid of ten equidistant points in x∈[0,1] , y∈[0,1] . Figure (3) exhibition the analytic and neural solutions with distinct types of threshold functions. The neural results with different types of threshold functions withLevenberg-Marquardt algorithmic rule (trainlm) and performance of the trail with epoch and time insert in Table (5), Table (6) gave the initial weight and bias of the design network.

Table 1: Analytic and Neural solution of example

input	input	Analytic solution	U-activation function with trainlm				
x	y	$U_a(x)$	Tansig	Logsig	Purelin	Tribas	Hardlim
0.0	0.0	0	-2.70819664139944e-10	-0.00433615233480657	0.0297505949975393	0.00115293109570976	0.0138361023582332
0.1	0.1	0.00826856506658583	0.00826856584286174	0.00826856506658589	0.0325401952527529	0.00928314304368850	0.0138361023582332
0.2	0.2	0.0299159494490976	0.0299159487867058	0.0299159494490976	0.0353297955079664	0.0306614862581160	0.0331189236295999
0.3	0.3	0.0566735880809494	0.0578902610972764	0.0569642403485118	0.0381193957631799	0.0569006754122053	0.0512052112367790
0.4	0.4	0.0783209724634611	0.0783209725092818	0.0783209724634609	0.0409089960183934	0.0756347187752336	0.0512052112367790
0.5	0.5	0.0865895375300469	0.0853456428493873	0.0865895375300471	0.0436985962736070	0.0866473827233342	0.0372797218041548
0.6	0.6	0.0783209724634611	0.0783209721735542	0.0794408466447368	0.0464881965288205	0.0812679489862926	0.0372797218041548
0.7	0.7	0.0566735880809494	0.0572124833060563	0.0566735880809493	0.0492777967840340	0.0566956502045704	0.0372797218041548
0.8	0.8	0.0299159494490976	0.0299159493443435	0.0280243868507734	0.0520673970392475	0.0321233514228482	0.0372797218041548
0.9	0.9	0.0082685650658583	0.0098851742943530 3	0.0082685650658597	0.0548569972944611	0.0131499328474210	0.0372797218041548
1.0	1.0	1.29863505986854e-33	-2.63646475203803e-11	-6.72935887802516e-17	0.0576465975496746	-2.76404011817711e-06	0.0372797218041548
Performers			5.392182421979792e-07	2.156256641504942e-06	0.001103580795378	4.324623382311578e-06	7.068614676256698e-04
time			0:00:00	0:00:00	0:00:00	0:00:01	0:00:00
epochs			21	13	6	36	8

Table 2: Initial weight and bias of the network for different training algorithm

"Weights and bias for trainlm"				
Net.IW{1,1}	Net.IU{1,1}	Net.LW{2,1}	Net.B{1}	B-Output
0.4294	0.1249	0.7639	0.6816	0.6633
0.0244	0.2902	0.7593	0.4633	
0.3175	0.6537	0.7406	0.2122	
0.9569	0.9357	0.7437	0.0985	
0.4579	0.2405	0.1059	0.8236	

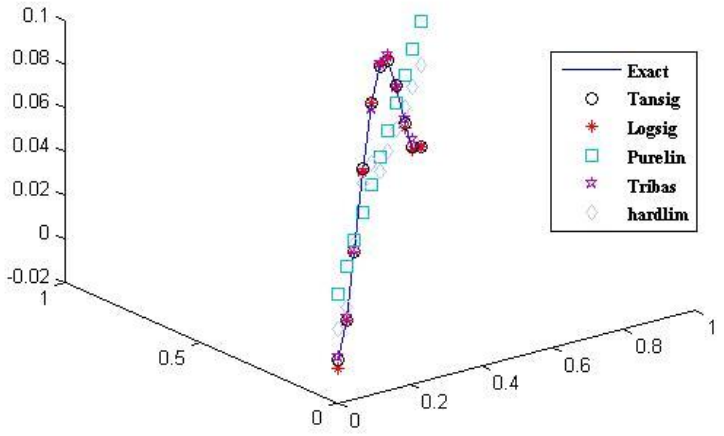

Figure (1): Analytic and neural solutions with different case of threshold functions.

Table 3: Analytic and Neural solution of example

input	input	Analytic solution	U-activation function **with** trainlm				
x	**y**	**$U_a(x)$**	Tansig	Logsig	Purelin	Tribas	hardlim
0.0	**0.0**	0	3.25881118559898e-06	0	0.0161804973955508	0.695282485105193	0.0456951038676357
0.1	**0.1**	0.0913885792216319	0.0913506580270492	0.0913885792216319	0.0885361458691941	0.651371399386103	0.0456951038676357
0.2	**0.2**	0.1702959966400220	0.1703287923200795	0.1703226207400669	0.160891794342838	0.546751261866624	0.2062710817000557
0.3	**0.3**	0.2422475581629222	0.2422521510348335	0.2422475581629222	0.233247442816481	0.455561008033229	0.2062710817000557
0.4	**0.4**	0.3110285013605370	0.3110050294669952	0.3110285013605370	0.305603091290124	0.367670747406037	0.4221917569993049
0.5	**0.5**	0.3790816623203960	0.3790675684212224	0.3790970606115513	0.377958739763768	0.392126770509323	0.6625268802183392
0.6	**0.6**	0.4478302950527726	0.4478387991244418	0.4478440650528340	0.450314388237411	0.517564082636165	0.6625268802183392
0.7	**0.7**	0.5179384718544440	0.5179540382115551	0.5179384718544440	0.522670036711054	0.654610416195834	0.6625268802183392
0.8	**0.8**	0.5895196009217795	0.5895220228908670	0.5895196009217795	0.595025685184698	0.695282485105193	0.6625268802183392
0.9	**0.9**	0.6623019757174436	0.6622900939558446	0.6623019757174436	0.667381333658341	0.695282485105193	0.6625268802183392
1.0	**1.0**	0.7357588823428885	0.7357626879259999	0.7356480324268860	0.739736982131985	0.695282485105193	0.6625268802183392
Performens			3.428191810704968e-10	1.220296989057379e-09	5.187114022489001e-05	0.093186551670436	0.016105125620520
time			0:00:02	0:00:02	0:00:00	0:00:01	0:00:00
epochs			198	169	8	61	4

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

Weights and bias for trainlm				
Net.IW{1,1}	**Net.IU{1,1}**	**Net.LW{2,1}**	**Net.B{1}"**	B-Output
0.0680	0.9685	0.8113	0.7888	0.1719
0.0988	0.5470	0.5768	0.4730	
0.4030	0.1070	0.9440	0.8288	
0.7242	0.6137	0.8715	0.3225	
0.7830	0.5666	0.5076	0.9761	

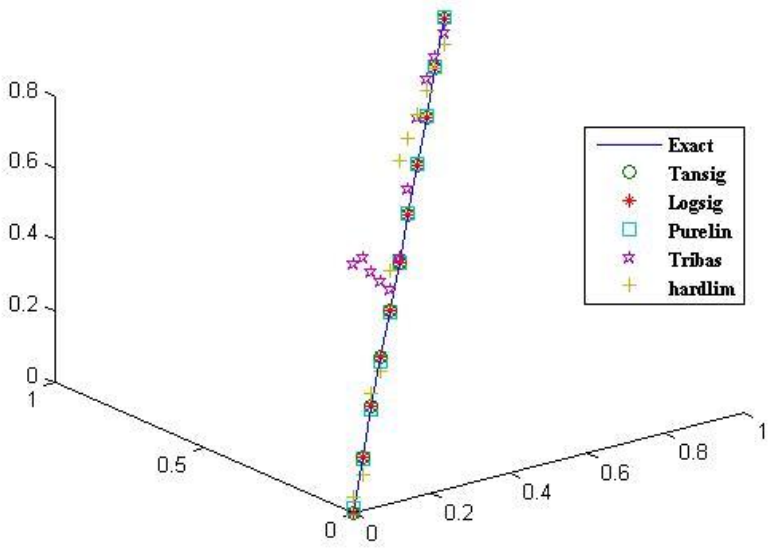

Figure (2): Analytic and neural solutions with different case of threshold functions

Table 3: Analytic and Neural solution of example

input x	input y	Analytic solution $U_a(x)$	U-activation function with trainlm Tansig	Logsig	Purelin	Tribas	hardlim
0.0	**0.0**	0	-0.000744886921732656	-3.815777579688886e-14	-3.815777579688886e-14	0.039887865401288 8	0.0867831827603239
0.1	**0.1**	0.00309016994374947	0.003170864275206 69	0.00309016994379937	0.00309016994379937	-0.007815346910333 62	0.0867831827603239
0.2	**0.2**	0.02351141009169 89	0.023994564573765 1	0.02427789633241395	0.02427789632413 95	0.014488266073475 1	0.1762649387411 82
0.3	**0.3**	0.07281152949374 53	0.072869386981818 1	0.07281152949373 16	0.07281152949373 16	0.077962751307777 9	0.0728115294939714
0.4	**0.4**	0.15216904260722 5	0.152058866911187	0.15156094199371 6	0.15156094199371 6	0.165662428059564	0.2692141666701 13
0.5	**0.5**	0.25000000000000 0	0.249946424463744	0.24979572646460 6	0.24979572646460 6	0.253362104811350	0.2692141666701 13
0.6	**0.6**	0.34238034586625 5	0.343531437356674	0.34238034586625 2	0.34238034586625 2	0.287234114680657	0.2692141666701 13
0.7	**0.7**	0.39641832724372 4	0.396415997049807	0.39641832724371 8	0.39641832724371 8	0.250083423992595	0.2692141666701 13
0.8	**0.8**	0.37618256146718 3	0.376142297413746	0.37618256146718 2	0.37618256146718 2	0.225312902814218	0.2692141666701 13
0.9	**0.9**	0.25030376544370 7	0.250249289807542	0.24403697387387 8	0.24403697387387 8	0.225139063395213	0.2692141666701 13
1.0	**1.0**	0	-3.421819982328 61e-05	-3.256834381741 36e-15	-3.256834381741 36e-15	0.224965223976209	0.2692141666701 13
Performens			1.949035990090646e-07	3.661062887056452e-06	0.030820802400484	0.009133688942856	0.0143407748662 01
time			0:00:00	0:00:00	0:00:00	0:00:01	0:00:00
epochs			17	17	6	52	5

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

Weights and bias for trainlm				
B-Output	B-Output	B-Output	B-Output	B-Output
0.8879	0.8429	0.4241	0.3381	0.1654
0.8988	0.9390	0.3411	0.8595	
0.8154	0.0014	0.5414	0.3405	
0.0031	0.0875	0.9262	0.1381	
0.2607	0.0228	0.2985	0.5078	

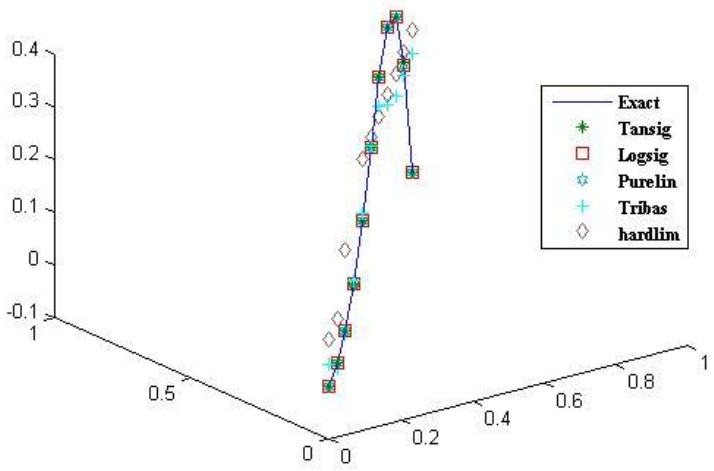

Figure (3):analytic and neural solutions with different case of threshold functions

CONCLUSIONS

It is very painful to have which activation functions will be the fastest for a assumed problem. It will impend on many agent conclude the complexity of the problem, the number of data points in the training set, the number of weights and biases in the Ann, and the error goal, whether the Ann . In general, the practical arise on Ann explain the tansig threshold function will have the fastest convergency, Then the logsig.

REFERENCES

[1] BARRON, A. R., Universal approximation bounds for superposition of a sigmoidal function, IEEE Trans. on Information Theory, 39.
[2] CYBENKO, G., Approximation by superposition of sigmoidal functions,Mathematics of Control, Signals and Systems, 2, # 4 (1989), 303-314.
[3] HORNIK, K., STINCHCOMBE, M. AND WHITE, H . , Multilayer feedforward networks are universal approximators, Neural Networks, 2 (1989),359-366.
[4] MHASKAR, H. N. AND MICCHELLI, C. A. , Approximation by superposition of asigmoidal function and radial basis functions, Advances in Applied Mathematics, 13 (1992),350-373 .
[5] Schalkoff, R.J. (1997) Artificial Neural Networks.McGraw-Hill, New York.
[6] Picton, P. (2000) Neural Networks. 2nd ed. Palgrave, Great Britain.
[7] Rosenblatt, F. (1957) The perceptron - a perceiving and recognizing automation. Comell Aeronautical Laboratory, Report 85–460.
[8] Lee, H. and kang, I.S., Neural algorithms for solving differential equations, journal of computational physics 91 (1990) 110-131.
[9] Meade Jr, A.J. and Fernandez, A.A., The numerical solution of linear ordinary differential equations by feed forward neural networks, Mathematical and Computer Modelling 19(12) (1994), 1 -25.
[10] Meade Jr, A.J. and Fernandez, A.A., Solution of nonlinear ordinary differential equations by feedforward neural networks, Mathematical and Computer Modelling 20 (9)(1994)19-44.

[11] Dissanayake, M.W.M.G. and Phan-Thien, N., Neural network based approximations forsolving partial differential equations, Communications in Numerical Methods in Engineering 10 (1994) 195-201.
[12] Lagaris, I.E. and Likas, Fotiadis, A, D.I., Artificial neural networks for solving ordinary and partial differential equations, IEEE Transactions on Neural Networks 9 (5)(1998) 987-1000.
[13] Pai, G. V and Rajasekaran, S, (2006), 'Neural Networks, Fuzzy Logic and Genetic Algorithms Synthesis and Applications', 6th ed, Prentice Hall of India Pvt. Ltd.

[14] Bekir Karlik and A Vehbi Olgac ,Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks, International Journal of Artificial Intelligence And Expert Systems (IJAE), 1(2010),111-122.
[15] B. M. wilamowski and S. Iplikei, an Algorithm for fast convergence in training neural network, IEEE.(2001):1778-1782.
[16] M. Kameswar Rao And K.P. Ravindran, An Evolutionary Approach For Solving differential Equations,2011.
[17] I. E. Lagaris, A. Likas and D. I. Fotiadis , Artificial Neural Net works for Solving Ordinary and Partial Differential Equations,1997.

تأثير دوال التنشيط في تصميم شبكة عصبية ذات التغذية الأمامية لحل المعادلات التفاضلية الجزئية

د. خالد منديل محمد
جامعة القادسية- كلية التربية -قسم الرياضيات
الديوانية- العراق
البريد الإلكتروني :Khalid_math98@yahoo.com

المستخلص *ــ في هذا البحث نناقش تأثير دوال التنشيط في تصميم الشبكة العصبية ذات التغذية الأمامية لحل المعادلات التفاضلية الجزئية. استخدمنا شبكة متعددة الطبقات ذات طبقة خفية واحدة ذو سبعة وحدات (عصبونات) خفية و وحدة أخراج خطية ودوال الاستثارة لكل حدة خفية هي مختلفة*

logsig , tansig, purelin, tribas and hardlim

باستخدام خوارزمية التدريب لـ ڤنبرك - ماركوادت و بايسن(trainlm). أخيرا النتائج للاختبارات العددية قورنت مع الحل المضبوط في أمثلة توضيحية لتعزيز و تأكيد الدقة و كفاءة للتقنية المقترحة .

الكلمات المفتاحية_ المعادلات التفاضلية الجزئية،الشبكات العصبية الصناعية الشبكة العصبية ذات التغذية الأمامية، خوارزمية التدريب لـ ڤنبرك – ماركوادت.