Al-Qadisiyah journal for pure science Vol. 21 No. 4 Year 2016

Design Collocation Neural Network for Solve regularly perturbed problems with Initial and Boundary conditions

Received :11/1/2016 Accepted : 28/3/2016

Khalid. Mindeel. M. Al-Abrahemee
Department of Mathematics, College of Education / University of AL-Qadisiyha
(Email: khalid_math98@yahoo.com)

Abstract

Recently, there has been an increasing interest in the study of regular and perturbed systems. The aim of this paper is to design artificial neural networks for solve regular perturbation problems with initial and boundary conditions. We design a multi-layer collocation neural network having one hidden layer with 5 hidden units (neurons) and one linear output unit the sigmoid activation function of each hidden unit is ridge basis function where the network trained by back propagation with different training algorithms such as quasi-Newton, Levenberg-Marquardt, and Bayesian Regulation. Finally the results of numerical experiments are compared with the exact solution in illustrative examples to confirm the accuracy and efficiency of the presented scheme.

Keywords: Artificial neural network, back propagation training algorithm, regular perturbed problems.

Mathematics Classification QA 1 - 939

1. Introduction

Many methods have been developed so far solving regularly perturbed initial value problems (RPIVP) and perturbed boundary value problems (RPBVP) , nowadays there is a new way of computing denominated artificial intelligence which through different methods is capable of managing the imprecision's and uncertainties that appear when trying to solve problems

related to the real world , offering strong solution and of easy implementation.

One of those techniques is known as Artificial Neural Networks (ANN). Inspired, in their origin, in the functioning of the human brain, and entitled with some intelligence. These are the combination of a great amount of elements of process– artificial neurons interconnected that operating in a parallel way get to solve problems related to aspects of classification . The construction of any given ANN we can identify, depending on the location in

the network, three kind of computational neurons: input, output and hidden.

Very often, a mathematical problem cannot be solved exactly or, if the exact solution is available, it exhibits such an intricate dependency in the parameters that it is hard to use as such. It may be the case, however, that a parameter can be identified, say ε, such that the solution is available and reasonably simple for $\varepsilon = 0$. Then, one may

wonder how this solution is altered for non-zero but small ε. [9]

Regularly perturbed problems (RPP) with initial or boundary conditions in ordinary differential equations (ODE) are characterized by the presence of a small parameter ε. These problems have been treated numerically by means of exponential-fitting, adaptive meshes, and ideas based on the method of matched. This paper is organized as follows: The next section definition the Regularly perturbed problems .In section 3,We define Artificial Neural Network(ANN). Section 4 ANN characterized. Neural Network Topology in section 5. Description of method given in section 6. Section 7 illustrate the method, section 8 report our numerical finding accuracy of method. Finally conclusions the last part of the paper.

2. Regularly Perturbed Problems

A perturbation problem is called *regular* if its solution y features smooth dependence on the parameter, i.e., Since ε usually represents a physically meaningful parameter, letting ε tend to 0 corresponds to neglecting the effect of small perturbations.

The 2nd order regular perturbation problem has the form:

$$\begin{cases} y'' = f(x, y, y', \epsilon) & a \le x \le b \\ initail\ or\ boundary\ condition \end{cases} \quad (1)$$

where f is in general nonlinear functions of their arguments, and

f (x, y, y′, ε) ϵ C^3([a, b]×R^2 ×[0, 1])

$\frac{\partial f}{\partial \varepsilon}$ (x, y, y′, ε) ≠ 0, (x, y, y′, ε) ϵ ([a, b] ×R^2× [0, 1]).

Assume that our problem contains only one small, positive parameter ϵ (0 < ϵ ≪1), denote the problem by Pϵ. What happens if ϵ⟶0? we have the reduced problem P_0. We want to study the relationship between the solution of P_ϵ and the solution of P_0 under appropriate assumptions. A perturbation problem (1) is called regular perturbation problems (RPP) , if $y_\epsilon(\epsilon, x) \xrightarrow[convearge]{} y_{reduced}(x)$ uniformly as $\epsilon \longrightarrow 0$, $y(\epsilon, x)$ and $y_{reduced}(x)$ denote the solutions to a RPP when $0 < \epsilon << 1$ and $\epsilon = 0$ respectively [7].

3. Artificial Neural Networks [8]

An Artificial Neural Network (Ann) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. Ann's, like people, learn by example. An Ann is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of Ann's as well. That is Artificial Neural Networks are relatively crude electronic models based on the neural structure of the brain. The brain basically learns from experience. It is natural proof that some problems that are beyond the scope of current computers are indeed solvable by small energy efficient packages. This brain modeling also promises a less technical way to develop machine solutions. This new approach to computing also provides a more graceful degradation during system overload than its more traditional counterparts.

These biologically inspired methods of computing are thought to be the next major advancement in the computing industry. Even simple animal brains are capable of functions that are currently impossible for computers. Computers do rote things well, like keeping ledgers or performing complex math. But computers have trouble recognizing even simple patterns much less

generalizing those patterns of the past into actions of the future.

4. ANN characterized [5]

1-Architecture: its pattern of connections between the neurons.
2-training Algorithm: its method of determining the weight on the connections.
3- Activations function.

5. Neural Network Topology [10]

In an ANN expressions structure, architecture or topology, express the way in which computational neurons are organized in the network. Particularly, these terms are focused in the description of how the nodes are connected and in how the information is transmitted through the network. As it has been mentioned, the distribution of computational in the following:

Number of layers: neurons in the neural network is done forming levels or layers of a determined number of nodes each one. As there are input, output and hidden neurons, we can talk about an **input layer**, an **output layer** and **single layer or multilayer hidden layers**. By the peculiarity of the behavior of the input nodes some authors consider just two kinds of layers in the ANN, the hidden and the output.

Connection patterns: Depending on the links between the elements of the different layers. the ANN can be classified as: **totally connected,** when all the outputs from a level get to all and each one of the nodes in the following level, if some of the links in the network are lost, then we say that the network is **partially connected.**

Information flow: Another classification of the ANN is obtained by considering the direction of the flow of the through the layers, when any output of the neurons is input of neurons of the same level or preceding levels, the network is described as **feed forward**. In counter position if there is at least one connected exit as entrance of neurons of previous levels or of the same level, including themselves, the network is denominated of **feedback.**

6. Description of the Method

In this section we will explain how this approach can be used to find the approximate solution the regular perturbation with initial value problems (RPIVPs) and boundary value problems (RPBVPs)

$\frac{d^2y}{dx^2} = F(x, y, y', \varepsilon)$, with initial or boundary condition (2)

where a subject to certain IC's and BC's $\in$ 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 discrete form $Min_p \sum_{x_i \epsilon D} F(x_i, y_t(x_i, p), y_t(x_i, p), \varepsilon)$ (3)

subject to the constraints imposed by the IC's and BC's.

In the proposed approach, the trial solution y_t employs a FFNN and the parameters p corresponding 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 IC's and BC's. This is achieved by writing it as a sum of two terms :

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

where $N(x, p)$ is a single-output ANN with parameters p and n input units fed with the input vector x. The term $A(x)$ contains no adjustable parameters and satisfies the IC's and BCs, the second term G is not depending on the IC's or BC's by constructed, this term can be formed by using suggested networks whose weights and biases are to be adjusted using the minimization technique.

This term can be formed by using an ANN whose weights and biases are to be adjusted in order to deal with the minimization problem.

7. Illustration of the Method

7.1 Illustration of the Method with initial conditions

In this section we describe solution of RPIVPs using ANN. To illustrate the method, we will consider the 2nd order SPIVPs:

$$\begin{cases} y'' = f(x, y, y', \epsilon) & a \le x \le b \\ iniail\ condition \end{cases} \quad (5)$$

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

$$y_t(x,p) = A + A'(x-a) + (x-a)^2 N(x,p) \quad (6)$$

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

7.2 Illustration of the Method with boundary condition

In this section we describe solution of RPBVPs using ANN. To illustrate the method, we will consider the 2nd order SPBVPs:

$$\begin{cases} y'' = f(x, y, y', \epsilon) & a \le x \le b \\ boundary\ condition \end{cases} \quad (7)$$

where $x \in$ [a,b] and the BC case of Dirichlet BC: y (a) = A, y (b) = B or in case of Neumann BC.

y' (a) = A, y' (b) = B or In the case of mixed BC: y(a) = A, y′(b) = B, or y′(a) = A, y(b) = B; ; a trial solution can be written as:

$$y_t(x,p) = \frac{(bA - aB)}{(b-a)} + \frac{(B-A)}{b-a}x + (x-a)(x-b)N(x,p) \quad (8)$$

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

8. Numerical result

In this section we report some numerical result and the solution of number of model problem. In all cases we used a multi-layer FFNN having one hidden layer with 5 hidden units (neurons) and one linear out output unit. The sigmoid activation of each hidden is radbas (radial basis function). For each test problem the exact analytic solution $y_a(x)$ were known in advance. Therefore we test the accuracy of obtained solutions computing the mean square error (MSE).

8.1 linear examples

Example 1 [3]

Consider the following 2nd order of liner regular perturbation problem (RPP) $y'' = -\varepsilon y' - 1$, with I.C: $y(0) = 0$, $y'(0) = 1$ x ∈[0,1] and the analytic solution :

$$y = \frac{(1+\varepsilon)(1-e^{-\varepsilon x})}{\varepsilon^2} - \frac{x}{\varepsilon}$$ and s.t ε =10^{-6}

According to the equation (6) the trial neural form of the solution is taken to be: $y_t(x) = x + x^2 N(x,p)$.

The FFNN trained using a grid of ten equidistant points in [0,1]. Figure (8.1) display the analytic and neural solution with different training algorithms. The neural result with different types of training algorithms such as: Levenberg – Marquardt (trainlm), quasi – Newton (trainbfg), Bayesian Regulation (trainbr) introduced in table (8.1) and its errors gave in table (8.2), table (8.3) gave the performance of the train for epoch and time, table (8.4) gave the initial weight and bias of the design network.

Example 2 [3]

Consider the following 2nd order of liner regular perturbation problem (RPP) $y'' + (1 - \varepsilon x)y = 0$, With I.C $y(0) = 1$, $y'(1) = 0$, x ∈[0,1] and the analytic solution :

$$y = cosx + (\tfrac{\varepsilon}{4})[x^2 sinx + xcosx - sinx] + \varepsilon^2[\left(-\tfrac{1}{32}\right)x^4 cosx\, x^3 sinx + \left(\tfrac{7}{16}\right)x^2 cosx - \left(\tfrac{7}{16}\right)xsinx]$$ and s.t ε =10^{-5}

According to the equation (6) the trial neural form of the solution is taken to be: $y_t(x) = x + x(x-1)N(x,p)$.

The FFNN trained using a grid of ten equidistant points in [0,1]. Figure (8.2) display the analytic and neural solution

with different training algorithms. The neural result with different types of training algorithms such as: Levenberg – Marquardt (trainlm), quasi – Newton (trainbfg), Bayesian Regulation (trainbr) introduced in table (8.5) and its errors gave in table (8.6), table (8.7) gave the performance of the train for epoch and time, table (8.8) gave the initial weight and bias of the design network.

8.2 Nonlinear examples

Example 3 [6]

Consider the following 1st order non-liner regular perturbed boundary-value problem (RPBVPs)

$y' = y^2 \sin(\varepsilon x)$, With B.C $y(0) = 1$, $y(1) = \frac{\varepsilon}{(\varepsilon-1)+\cos\varepsilon}$, x $\in$[0,1] and the nalytic solution :

$y = \frac{\varepsilon}{(\varepsilon-1)+\cos\varepsilon x}$ and s.t $\varepsilon = 10^{-1}$

According to the equation (8) the trial neural form of the solution is taken to be:

$$y_t(x) = 1 + 0.0525x + x(x-1)\,N(x,p).$$

The FFNN trained using a grid of ten equidistant points in [0,1]. Figure (8.3) display the analytic and neural solution with different training algorithms. The neural result with different types of training algorithms such as: Levenberg – Marquardt (trainlm), quasi – Newton (trainbfg), Bayesian Regulation (trainbr) introduced in table (8.9) and its errors gave in table (8.10), table (8.11) gave the performance of the train for epoch and time, table (8.12) gave the initial weight and bias of the design network.

Example 4 [1]

Consider the following second order non-liner regular perturbed initial-value problem (RPIVPs)

$y'' + y + \varepsilon y^3$, With I.C :$y(0) = 1$, $y'(0) = 0$, x $\in$[0,1] and the analytic solution :

$$y = \cos(x) + \varepsilon\left(\frac{1}{32}[cos3x - cosx] - \frac{3}{8}\,xsinx\right)$$ [2] , and s.t $\varepsilon = 10^{-5}$

According to the equation (6) the trial neural form of the solution is taken to be:

$$y_t(x) = 1 + x^2\,N(x,p).$$

The FFNN trained using a grid of ten equidistant points in [0,1]. Figure (8.4) display the analytic and neural solution with different training algorithms. The neural result with different types of training algorithms such as: Levenberg – Marquardt (trainlm), quasi – Newton (trainbfg), Bayesian Regulation (trainbr) introduced in table (8.13) and its errors gave in table (8.14), table (8.15) gave the performance of the train for epoch and time, table (8.16) gave the initial weight and bias of the design network.

8. Conclusion

This paper present new technique to solve 2nd order regular perturbed problems using artificial neural network which have the regularly perturbed ,the suggested architecture of the ANN is efficient and more accurate than other numerical method and the practical results show which contain up to a few hundred weights the Levenberg-Marquardt algorithm (trainlm) will have the fastest convergence, then trainbfg and then trainbr. However, "trainbr" it does perform well on function approximation on problems, in contrast to his performance in the solution of singular perturbation problems. The performance of the various algorithms can be affected by the accuracy required of the approximation.

References

[1]Abdul-Majid Wazwaz, Partial Differential Equations and Solitary Waves Theory, Springer Nonlinear Physical Science,2009.

[2]A.M. Wazwaz, Reliable analysis for the nonlinear Schrodinger equations with a cubic and a power law nonlinearities, Math. Comput. Modelling, 43, 178–184, (2006).
[3] David Sherrill , Perturbation Theory , 2006.
[4] Dean W ang , Introduction to Perturbation Methods,2005.
[5] E. O'Malley, Singular Perturbation Methods for Ordinary Differential Equations, Springer Verlag, New York, 1991.
[6] H. S. Greenside , Some Notes on Singular Perturbation Theory,2008.
[7] M. Jianzhong , " Some Singular Singularly Perturbed Problem", M.Sc. Thesis, Calgary, Alberta ,1997.
[8] R. E. O'Malley, Introduction to Singular Perturbations, Academic Press, New York, 1974.
[9] S. Albert , " Numerical Treatment of Non-Linear Singular Perturbation Problems" , MSc. thesis, Department of Mathematics and Applied Mathematics, University of the Western Cape, 2007.
[10] S. Haykin, Neural networks: A comprehensive foundation ,1993.

Table 8.1: Analytic and Neural solution of example

input	Analytic solution	Out of suggested FFNN $y_t(x)$ for different training algorithm		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	0	4.36184828137977e-05	6.92555742637888e-06	5.25038450309868e-08
0.1	0.0949513557425235	0.0947693565460115	0.0946685580708633	0.0939425585113695
0.2	0.180021716660121	0.180004497690708	0.180024967046889	0.180021600793005
0.3	0.254989037988707	0.255046494855935	0.254994743523004	0.255229076482784
0.4	0.319964342052117	0.319965260810589	0.319765430379792	0.319965314046176
0.5	0.374947628995869	0.374928695695493	0.374696657569078	0.374944302314382
0.6	0.420049921027385	0.419988617596578	0.419844040667538	0.420057369738515
0.7	0.455049173557200	0.455064011110410	0.455013106541516	0.455037944688613
0.8	0.479945386294276	0.480024534008685	0.480018841562928	0.479956026177754
0.9	0.494960604351945	0.494879462028247	0.494899138071445	0.494954951924237
1.0	0.499983805231750	0.500018103388820	0.499997553425343	0.499985071343206

Table 8.2 : Accuracy of solutions for example

The error $E(x) = \| y_t(x) - y_a(x) \|$ where $y_t(x)$ computed by the following training algorithm		
Trainlm	Trainbfg	Trainbr
4.36184828137977e-05	6.92555742637888e-06	5.25038450309868e-08
0.00018199919651194 8	0.000282797671660132	0.00100879723115392
1.72189694137848e-05	3.25038676732281e-06	1.15867116123880e-07
5.74568672274123e-05	5.70553429618537e-06	0.000240038494077000
9.18758471690762e-07	0.000198911672325275	9.71994058684977e-07
1.89333003760006e-05	0.000250971426791369	3.32668148728121e-06
6.13034308065696e-05	0.000205880359846256	7.44871113061985e-06
1.48375532101896e-05	3.60670156835941e-05	1.12288685870654e-05
7.91477144092001e-05	7.34552686526557e-05	1.06398834784360e-05
8.11423236978803e-05	6.14662805000221e-05	5.65242770866892e-06
3.42981570702339e-05	1.37481935932882e-05	1.26611145634392e-06

Table 8.3:The performance of the train with epoch and time

Train Function	Performance of train	Epoch	Time	MSE
Trainlm	3.08-33	31	0:00:00	5.180585694283459e-09
Trainbfg	1.51-09	323	0:00:05	2.142440547089971e-08
Trainbr	3.78-11	913	0:00:11	9.778460826104464e-08

Table 8.4 : Initial 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}
0.2057	0.4845	0.2374
0.3883	0.1518	0.5309
0.5518	0.7819	0.0915
0.2290	0.1006	0.4053
0.6419	0.2941	0.1048

Weights and bias for trainbfg		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.5518	0.9962	0.4547
0.5836	0.3545	0.4134
0.5118	0.9713	0.2177
0.0826	0.3464	0.1257
0.7196	0.8865	0.3089

Weights and bias for trainbr		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.3983	0.9791	0.7565
0.7498	0.5493	0.4139
0.8352	0.3304	0.4923
0.3225	0.6195	0.6947
0.5523	0.3606	0.9727

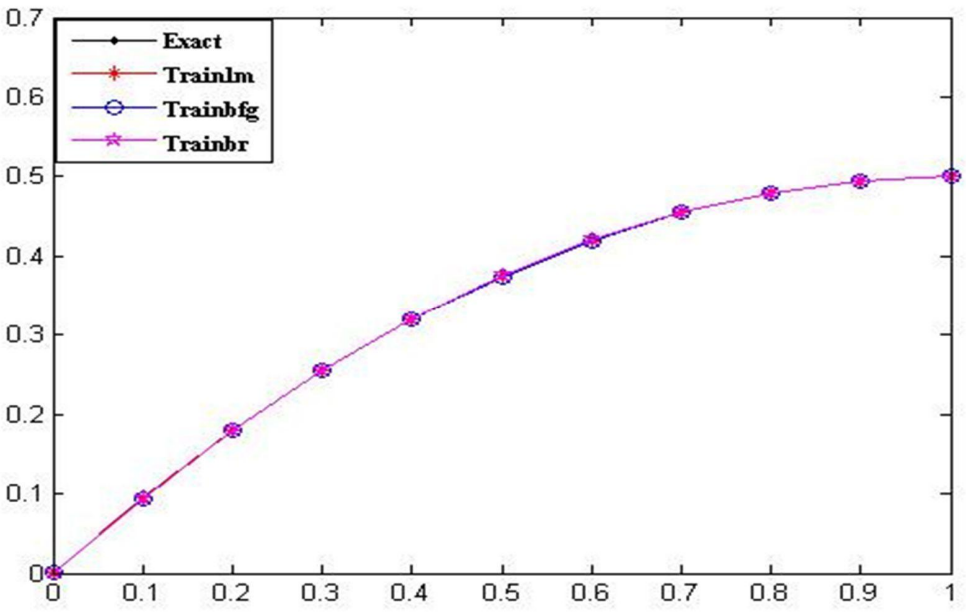

Figure 8.1: analytic and neural solution of example using : trainlm , trainbfg and trainbr training algorithm

Table 8.5: Analytic and Neural solution of example

input	Analytic solution	Out of suggested FFNN $y_t(x)$ for different training algorithm		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	1	0.999999999363522	1.00000000175704	1.00000007391302
0.1	0.995004166941360	0.995004166222954	0.994718744265259	0.995003545972025
0.2	0.980066591068125	0.980068291833021	0.979918140854827	0.980067374279565
0.3	0.955336533319445	0.955335975715681	0.955303854276101	0.955329194911241
0.4	0.921061097285179	0.921061096682491	0.921061096822551	0.921059209077894
0.5	0.877582759945254	0.877582759157436	0.877582759635566	0.877585905496474
0.6	0.825335949484257	0.825335948943901	0.825335949856014	0.825333157491326
0.7	0.764842704379160	0.764849495639380	0.764843495678654	0.764843858849604
0.8	0.696707457137437	0.696721651794313	0.696707457330868	0.696726991738074
0.9	0.621610994808670	0.621610994271581	0.621610995692091	0.621610746473634
1.0	0.540303656617805	0.540303656163387	0.540303658154140	0.540303716758258

Table 8.6 : Accuracy of solutions for example

The error $E(x) = \| y_t(x) - y_a(x) \|$ where $y_t(x)$ computed by the following training algorithm		
Trainlm	**Trainbfg**	**Trainbr**
6.36478314497424e-10	1.75703629423651e-09	7.39130205928973e-08
7.18405779309705e-10	0.000285422676100877	6.20969335507482e-07
1.70076489591775e-06	0.000148450213298568	7.83211439481235e-07
5.57603763162717e-07	3.26790433432533e-05	7.33840820343890e-06
6.02687455497630e-10	4.62627491870649e-10	1.88820728519445e-06
7.87818033032295e-10	3.09688052979595e-10	3.14555122005178e-06
5.40355538092285e-10	3.71757624684221e-10	2.79199293040655e-06
6.79126022007370e-06	7.91299493707598e-07	1.15447044413131e-06
1.41946568760210e-05	1.93431159978275e-10	1.95346006370833e-05
5.37088484797721e-10	8.83421780173421e-10	2.48335035646363e-07
4.54417947715058e-10	1.53633428201516e-09	6.01404528399741e-08

Table 8.7:The performance of the train with epoch and time

Train Function	Performance of train	Epoch	Time	MSE
Trainlm	1.76-33	561	0:00:07	2.280118409860440e-11
Trainbfg	4.14-24	484	0:00:08	9.506555990582628e-09
Trainbr	2.63-12	1029	0:00:12	4.173731601304228e-11

Table 8.8 : Initial 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}**
0.7112	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 trainbfg		
Net.IW{1,1}	**Net.LW{2,1}**	**Net.B{1}**
0.1068	0.9037	0.0305
0.6538	0.8909	0.7441
0.4942	0.3342	0.5000
0.7791	0.6987	0.4799
0.7150	0.1978	0.9047

Weights and bias for trainbr		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.2691	0.9831	0.6981
0.4228	0.3015	0.6665
0.5479	0.7011	0.1781
0.9427	0.6663	0.1280
0.4177	0.5391	0.9991

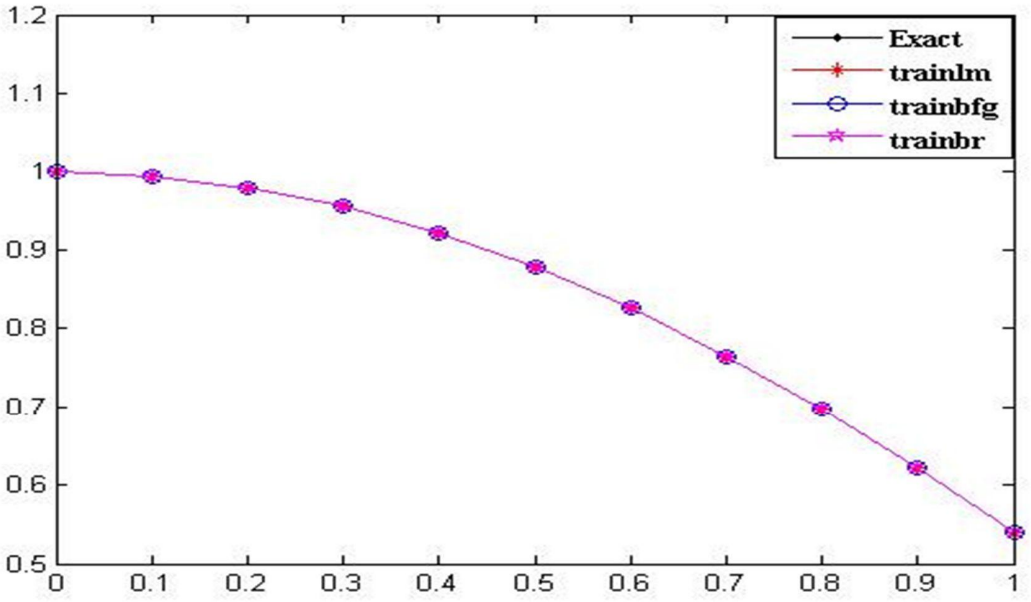

Figure 8.2: analytic and neural solution of example using : trainlm , trainbfg and trainbr training algorithm.

Table 8.5: Analytic and Neural solution of example

input	Analytic solution	Out of suggested FFNN $y_t(x)$ for different training algorithm		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	1	0.999999999363522	1.00000000175704	1.00000007391302
0.1	0.995004166941360	0.995004166222954	0.994718744265259	0.995003545972025
0.2	0.980066591068125	0.980068291833021	0.979918140854827	0.980067374279565
0.3	0.955336533319445	0.955335975715681	0.955303854276101	0.955329194911241
0.4	0.921061097285179	0.921061096682491	0.921061096822551	0.921059209077894
0.5	0.877582759945254	0.877582759157436	0.877582759635566	0.877585905496474
0.6	0.825335949484257	0.825335948943901	0.825335949856014	0.825333157491326
0.7	0.764842704379160	0.764849495639380	0.764843495678654	0.764843858849604
0.8	0.696707457137437	0.696721651794313	0.696707457330868	0.696726991738074
0.9	0.621610994808670	0.621610994271581	0.621610995692091	0.621610746473634
1.0	0.540303656617805	0.540303656163387	0.540303658154140	0.540303716758258

Table 8.6 : Accuracy of solutions for example

The error $E(x) = \| y_t(x) - y_a(x) \|$ where $y_t(x)$ computed by the following training algorithm		
Trainlm	**Trainbfg**	**Trainbr**
6.36478314497424e-10	1.75703629423651e-09	7.39130205928973e-08
7.18405779309705e-10	0.000285422676100877	6.20969335507482e-07
1.70076489591775e-06	0.000148450213298568	7.83211439481235e-07
5.57603763162717e-07	3.26790433432533e-05	7.33840820343890e-06
6.02687455497630e-10	4.62627491870649e-10	1.88820728519445e-06
7.87818033032295e-10	3.09688052979595e-10	3.14555122005178e-06
5.40355538092285e-10	3.71757624684221e-10	2.79199293040655e-06
6.79126022007370e-06	7.91299493707598e-07	1.15447044413131e-06
1.41946568760210e-05	1.93431159978275e-10	1.95346006370833e-05
5.37088484797721e-10	8.83421780173421e-10	2.48335035646363e-07
4.54417947715058e-10	1.53633428201516e-09	6.01404528399741e-08

Table 8.7:The performance of the train with epoch and time

Train Function	Performance of train	Epoch	Time	MSE
Trainlm	1.76-33	561	0:00:07	2.280118409860440e-11
Trainbfg	4.14-24	484	0:00:08	9.506555990582628e-09
Trainbr	2.63-12	1029	0:00:12	4.173731601304228e-11

Table 8.8 : Initial 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}**
0.7112	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 trainbfg		
Net.IW{1,1}	**Net.LW{2,1}**	**Net.B{1}**
0.1068	0.9037	0.0305
0.6538	0.8909	0.7441
0.4942	0.3342	0.5000
0.7791	0.6987	0.4799
0.7150	0.1978	0.9047

Weights and bias for trainbr		
Net.IW{1,1}	**Net.LW{2,1}**	**Net.B{1}**
0.2691	0.9831	0.6981
0.4228	0.3015	0.6665
0.5479	0.7011	0.1781
0.9427	0.6663	0.1280
0.4177	0.5391	0.9991

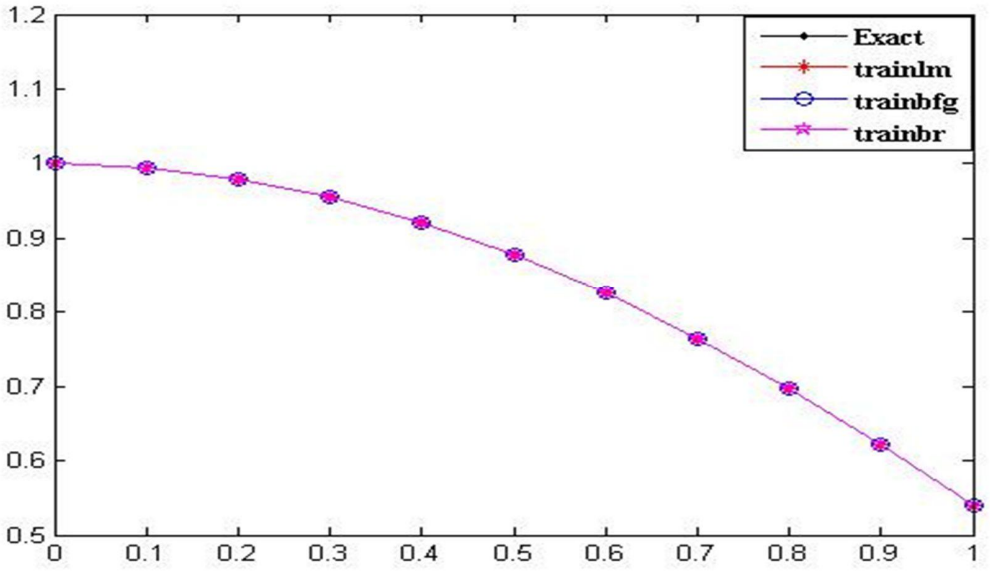

Figure 8.2: analytic and neural solution of example using : trainlm , trainbfg and trainbr training algorithm.

Table 8.13: Analytic and Neural solution of example

input	Analytic solution	Out of suggested FFNN $y_t(x)$ for different training algorithm		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	1	1.00000007655193	0.999999997808015	1.00000010685247
0.1	0.995004160294658	0.995003827257243	0.995015934200908	0.995003695463547
0.2	0.980066558105699	0.980067075925372	0.980073040397284	0.980067199050823
0.3	0.955336445450629	0.955330837130449	0.955336670360704	0.955330217949129
0.4	0.921060918130658	0.921059638277053	0.921060348005196	0.921059327559432
0.5	0.877582446784167	0.877584736056700	0.877583059974158	0.877585041548535
0.6	0.825335454973318	0.825333424474878	0.825335234897047	0.825333095283871
0.7	0.764841978499587	0.764842852400115	0.764840294707549	0.764842902029670
0.8	0.696706449324700	0.696719952490333	0.696706484316267	0.696722326499033
0.9	0.621609656220266	0.621609491223232	0.621613015073610	0.621609460508001
1.0	0.540301942494808	0.540301992377265	0.540301963121660	0.540302033982766

Table 8.14 : Accuracy of solutions for example

The error $E(x) = \| y_t(x) - y_a(x) \|$ where $y_t(x)$ computed by the following training algorithm		
Trainlm	**Trainbfg**	**Trainbr**
7.65519345691246e-08	2.19198481588023e-09	1.06852471981611e-07
3.33037414645965e-07	1.17739062507338e-05	4.64831110824626e-07
5.17819672363196e-07	6.48229158461966e-06	6.40945123420167e-07
5.60832018015045e-06	2.24910074786422e-07	6.22750149947837e-06
1.27985360465210e-06	5.70125461285542e-07	1.59057122550710e-06
2.28927253276279e-06	6.13189991560681e-07	2.59476436814676e-06
2.03049844083036e-06	2.20076271539860e-07	2.35968944706233e-06
8.73900528164384e-07	1.68379203746571e-06	9.23530083407620e-07
1.35031656330886e-05	3.49915669772827e-08	1.58771743330455e-05
1.64997034701742e-07	3.35885334390440e-06	1.95712265393944e-07
4.98824570538403e-08	2.06268523372799e-08	9.14879579871908e-08

Table 8.15:The performance of the train with epoch and time

Train Function	Performance of train	Epoch	Time	MSE
Trainlm	1.35e-12	410	0:00:05	2.054261645454189e-11
Trainbfg	1.07-13	651	0:00:11	1.777852198148876e-11
Trainbr	1.82e-12	871	0:00:10	2.793047001635929e-11

Table 8.16 : Initial 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}**
0.1112	0.0965	0.0598
0.7803	0.1320	0.2348
0.3897	0.9421	0.3532
0.2417	0.9561	0.8212
0.4039	0.5752	0.0154

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

Weights and bias for trainbr		
Net.IW{1,1}	**Net.LW{2,1}**	**Net.B{1}**
0.6915	0.6981	0.9831
0.4228	0.6665	0.3015
0.5479	0.1781	0.7011
0.9427	0.1280	0.6663
0.4177	0.9991	0.5391

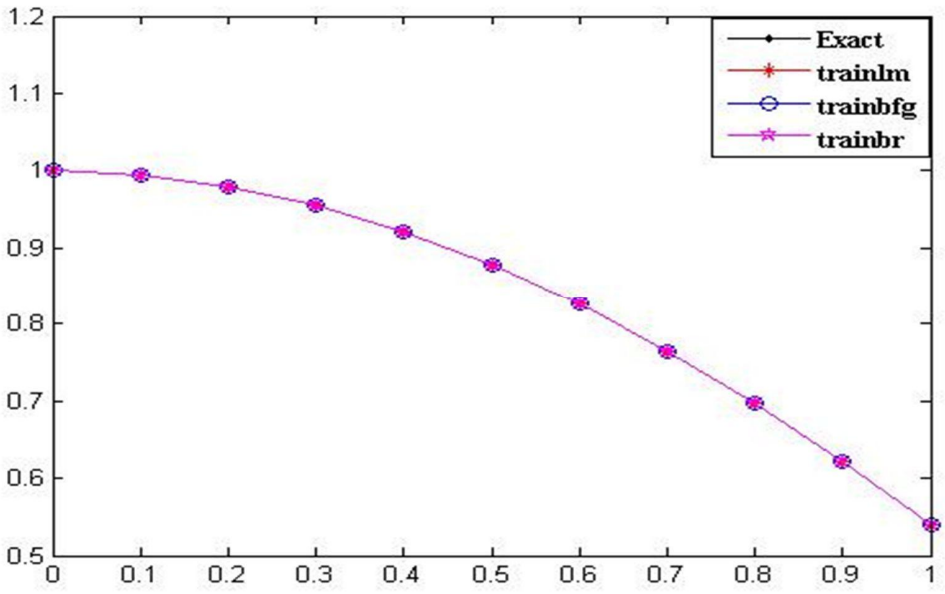

Figure 8.4: analytic and neural solution of example using : trainlm , trainbfg and trainbr training algorithm

تصميم شبكة عصبية لحل مسائل الاضطراب المنتظمة ذات الشروط الابتدائية والحدودية

د. خالد منديل محمد

جامعة القادسية- كلية التربيه -قسم الرياضيات

تاريخ الاستلام 2016/1/11 **تاريخ القبول 2016/3/28**

الاميل :Khalid_math98@yahoo.com

المستخلص

الهدف من هذا البحث هو حل مسائل الاضطراب المنتظمة باستخدام الشبكات العصبية. استخدمنا شبكة متعددة الطبقات ذات طبقة خفية واحدة ذو خمس وحدات (عصبونات) خفية و وحدة أخراج خطية دالة الاستثارة لكل وحدة خفية هي دالة الأساس الصلبة حيث أن الشبكة دربت بواسطة الانتشار المرتد مع خوارزميات تدريب مختلفة مثل شبه-نيوتن ، لفنبرك-ماركوادت و بايسن. أخيرا النتائج للاختبارات العددية قورنت مع الحل المضبوط في أمثلة توضيحية لتعزيز و تأكيد الدقة و كفاءة للتقنية المقترحة .