Republic of Iraq

Ministry of higher education and scientific research

University of Al- Qadisiyah

College of education

Dept. of Mathematics

Use Neural Networks for Solve Two Point Boundary Value Problems

A research

Submitted to the department of education Al– Qadisiyah University as a partial fulfillment o requirement for certificate o the Bachelor of science Mathematics

By

Ali Hamza Jabbar

Supervisor by

Dr. Khaled Mindeel

2019 **1440**

بسم الله
الرحمن
الرحيم

(يؤتي الحكمة من يشاء ومن يؤت الحكمة

فقد أوتي خيراً كثيراً وما يذكر الا أولو الالباب)

صدق الله العلي العظيم

سورة البقرة الاية 269

DEDICATION

I present this research to my dearest people "My family "for their support and help

presented to me in all my educationally life. I never forget their advices, kindness, and compassion …

The person believed me while I was not able to My mother

The individual who made me believed that there is nothing as real as a dream My father

Thanks for their support hoping to be the devoted daughter and this simple effort as gift wishing to accept it

Ali

ACKNOWLEDGEMENT

I should like expressed my sincere

great for my supervisor Khalid

Mendel for here unfailing guidance and remarkable efforts.

Also I would like to express my gratitude to my parents and my sister for all support and help they have given to me.

Ali

CONTENTS

CHAPTER ONE

MATEHMATICAL BACKGROUND

CHAPTER TWO

DESIGN FAST FEED FORWARD NETWORKS TO SOLVE TWO POINT BOVNDARY UALUE PROBLEMS

CHAPTER

ONE

MATHEMATICAL BACKGROUND

1.1.Introduction

This chapter contains a brief overview of artificial neural networks has been to clarify the most important features of artificial neural networks and the basic elements that make up the network. The work of each element in the network. The training was to explain the network and the types of training for the networks and the algorithm that was used in training (feedback) and how they work. As was stopped training conditions And how is the appropriate choice of parameters Walt.

1.2. APPROXIMATION THEORY [1]

The primary aim of a general approximation is to represent non-arithmetic quantities by arithmetic quantities so that the accuracy can

be ascertained to a desired degree. Secondly, we are also concerned with the amount of computation required to achieve this accuracy. A complicated function f(x) usually is approximated by an easier function of the form $\phi(x; a_0, \ldots, a_n)$ where $a_0, \ldots, a_n$ are parameters to be determined so as to characterize the best approximation of f. Depending on the sense in which the approximation is realized, there are three types of approaches:

1. Interpolatory approximation: The parameters a_i are chosen so that on a fixed prescribed set of points x_i, i = 0, 1, . . ., n, we have

$\phi(x_i; a_0, \ldots, a_n) = f(x_i) = f_i$.

Sometimes, we even further require that, for each i, the first r_i derivatives of ϕ agree with those of f at x_i .

2. Least-square approximation: The parameters a_i are chosen so as to

Minimize $\|f(x) - \psi(x; a_0, \ldots, a_n)\|_2$.

3. Min-Max approximation: the parameters a_i are chosen so as to

Minimize $\|f(x) - \phi(x; a_0, \ldots, a_n)\|_\infty$.

1.3. Artificial Neural Networks

A neural network is a parallel, distributed information processing structure consisting of processing elements (which can possess a local memory and can carry out localized information processing operations) interconnected via unidirectional signal channels called connections. Each processing element has a single output connection that branches ("fans out") into as many collateral connections as desired; each carries the same signal , the processing element output signal. The processing element output signal can be of any mathematical type desired. The information processing that goes on within each processing element can be defined arbitrarily with the restriction that it must be completely local; that is, it must depend only on the current values of the input signals arriving at the processing element via impinging connections and on the values stored in the processing element's local memory[2] .

ANN can be most adequately characterized as 'computational models'with particular properties such as the ability to adapt or learn, to generalize, or to cluster ororganize data, and which operation is based on parallel processing. However, many of the

abovementionedproperties can be attributed to existing (non-neural) models; the intriguing questionis to which extent the neural approach proves to be better suited for certain applications thanexisting models. To date an equivocal answer to this question is not found.

1.4.Using ANN for Solving Differential Equation

Application of ANNto the solution of differential equations is stipulated also by two reasons. The first reason is that logical sequence scheme corresponds to the logical scheme in the solution of differential equations. In other words, the solution scheme of differential equations enables to create the appropriate structure of ANN. The second reason is that ANN can accurately approximate the function [3] .

1.4.1. Connections Between Units[4]

In most cases it is assumed that each unit provides an additive contribution to the input of theunit with which it is connected. The total input to unit k is simply the weighted sum of theseparate outputs from each of the connected units plus a bias b_k:

$$S_k(x) = \sum_j w_{jk}(x)\, y_j(x) + b_k(x) \quad (1.1)$$

The contribution for positive w_{jk} is considered as an excitation and for negative w_{jk} as inhibition.

In some cases more complex rules for combining inputs are used, in which a distinction is madebetween excitatory and inhibitory inputs. We call units with a propagation rule (1.1) sigmaunits.

A different propagation rule, introduced byKröseet al[5].

1.4.2. Activation Function [4]

The activation function (sometimes called a transfer function) can be a linear or nonlinear function. There are many different types of activation functions. Selection of one type over another depends on the particular problem that the neuron (or ANN) is to solve.

The activation function, denoted by $\phi : R \rightarrow R$ defines the output of a neuron, whichis bounded monotonically increasing, differentiableand satisfies :$Lim_{x \rightarrow +\infty} \phi(x) = 1$ and$Lim_{x \rightarrow -\infty} \phi(x) = 0$.

1.4.3.The Bias[4]

Some ANN employ a bias unit(offset , threshold) as part of every layer except the output layer. This units have a constant activation value of 1 or -1, it's weight might be adjusted during training. The bias unit provides a constant term in the weighted sum which results in an improvement on the convergence properties of the ANN .

A bias acts exactly as a weight on a connection from a unit whose activation is always 1. Increasing the bias increases the net input to the unit .

In the previous section we discussed the properties of the basic processing unit in an ANN. The next section focuses on the pattern of connections between the units and thepropagation of data.

1.5. Multi-layer Feed Forward Neural Networks

A feedforward neural network(FFNN) has a layered structure. Each layer consists of units which receive theirinput from units from a layer directly below and send their output to units in a layer directlyabove the unit. There are no connections within a layer. The N_i inputs are fed into the firsthidden layer of $N_{k,1}$ hidden units. The input units are merely 'fan-out' units; no processing takes placein these units. The activation of a hidden unit is a function $\mathbf{f}_i$ of the weighted inputs plus abias. The output of the

hidden units is distributed over the next layer of$N_{k,2}$ hidden units, until the last layer of hidden units, of which the outputs are fed into a layerof N_o output units (see figure 1.1)[5]

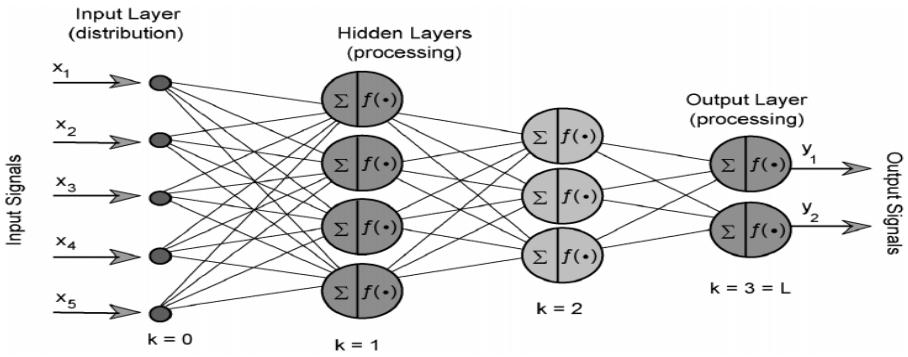

Figure 1.1: Multilayer feed forward neural network.

1.6. Training ANN[5]

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 (The number of iterations of training process) until satisfied output y_k (corresponding to the values of the input variables x) is obtained and the error is acceptably small.

CHAPTER

TWO

DESIGN FAST FEED FORWARD NEURAL NETWORKS TO SOLVE TWO POINT BOUNDARY VALUE PROBLEMS

2.1. 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.

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. In this chapter we design fast FFNN as a method to solve SBVP.

2.2. Description the Design of Feed Forward Neural Network

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 were called collocation neural network. Since it is known that a multilayer FFNN with one hidden layer can approximate any function to arbitrary accuracy 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 of a second order singular differential equation:

$$y^{(2)}(x) = F(x, y(x), y'(x)), \qquad (2.1)$$

which a subject to certain BC's and $x = (x_1, x_2,\ldots, x_n) \in R^n$, $D \subset R^n$ denotes the domain and y(x) is the solution to be computed.

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

$$\text{Min}_p \sum_{\vec{x}_i \in \hat{D}} F(x_i , y_t(x_i , p), y_t'(x_i , p)), \qquad (2.2)$$

subject to the constraints imposed by the BC's.

In our proposed approach, the trial solution y_t 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 $y_t(x)$ such that it satisfies the BC's. This is achieved by writing it as a sum of two terms:

$$y_t(x_i , p) = A(x) + G(x, N(x, p)), \qquad (2.3)$$

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 $y_t(x)$ 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.

2.3. Computation of the Gradient

An efficient minimization of equation (2.2) can be considered as a procedure of training the FFNN, where the error corresponding to each input vector x_i is the value $E(x_i)$ which has to forced near zero. Computation of this error value involves not only the FFNN output but also the derivatives of the output with respect to any of its inputs. Therefore, in computing the gradient of the error with respect to the network weights consider a 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 $x = (x_1, x_2, \ldots, x_n)$ the output of the FFNN is :

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

w_{ij} denotes the weight connecting the input unit j to the hidden unit i

v_i denotes the weight connecting the hidden unit i to the output unit ,

b_i denotes the bias of hidden unit i, and

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:

$$\frac{\partial N}{\partial \nu_i} = \sigma(z_i), \qquad (2.4)$$

$$\frac{\partial N}{\partial b_i} = v_i \sigma'(z_i), \qquad (2.5)$$

$$\frac{\partial N}{\partial w_{ij}} = v_i \sigma'(z_i)\, x_j , \qquad (2.6)$$

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.

2.4. Illustration of the Design Feed Forward Neural Network

In this section we describe solution of TPSBVP using FFNN. Now, consider the 2nd order SBVP :

$$\frac{d^2y}{dx^2} = f(x, y, \frac{dy}{dx}), \qquad (2.7)$$

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

$$\mathbf{y_t(x,p) = \frac{(bA - aB)}{b-a} + \frac{(B-A)}{b-a}x + (x-a)(x-b)N(x,p)}, \qquad (2.8)$$

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

Note that

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

$$E[p] = \sum_{i=1}^{n} \left\{ \frac{d^2 y_t(x_i,p)}{dx^2} - f(x_i, y_t(x_i, p), \frac{dy_t(x_i p)}{dx} \right\}^2, \qquad (2.9)$$

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

$$\mathbf{\frac{dy_t(x,p)}{dx} = \frac{A-B}{b-a} + \{(x-a) + (x-b)\} N(x,p) + (x-a)(x-b)\frac{dN(x,p)}{dx}}, \qquad (2.10)$$

and

$$\mathbf{\frac{d^2 y_t(x,p)}{dx^2} = 2N(x,p) + 2\{(x-a) + (x-b)\}\frac{dN(x,p)}{dx} + (x-a)(x-b)\frac{d^2 N(x,p)}{dx^2}}, \qquad (2.11)$$

it is straight forward to compute the gradient of the error with respect to the parameters p using (3.4) – (3.6). The same holds for all subsequent model problems.

Now, we describe the solution of higher order SBVP using FFNN. To illustrate the design, we will consider the 4^{th} order TPSBVP:

$$y^{(4)}(x) = f(x, y(x), y'(x), y''(x), y^{(3)}(x)), \qquad (2.12a)$$

with BC : $y^{(i)}(a) = A_i$ and $y^{(i)}(b) = B_i$, $i = 0,1$ (2.12b)

Let us in our approach assume that the trial function is in the form:

$$y_t(x, P) = Z(x) + M(x)\, N(x, P), \qquad (2.13)$$

Now one may ask whether, for a general given boundary conditions in

For the nth order TPSBVP equation (3.1) with mixed BC: $y(a) = A$, $y'(a) = A'$, $y''(a) = A''$,…,$y^{(k)}(a) = A^{(k)}$, $y(b) = B$, $y'(b) = B'$, $y''(b) = B''$,…,$y^{(n-k)}(b) = B^{(k)}$, $k = 1,2, \ldots, n-1$ and the trial solution can be have the form:

$$y_t(x) = A + A'x + A'' x^2 + \ldots + A^{(n-1)} x^{n-1} + (x-a)^n (x-b)^n N(x, p), \qquad (2.14)$$

For the above nth order TPSBVP the error function, which should be minimized, is given by the following equation:

$$E(P) = \sum_{i=1}^{n} \left\{ \frac{d^n y_t(x_i, p)}{dx^n} - f(x_i, y_t(x_i, P), y_t'(x_i, P), \ldots, y_t^{(n-1)}(x_i, P)) \right\}^2, \qquad (2.15)$$

Note that

We take the same manner of solution if the problems have Neumann or Mixed BC. That is the type of BC is not effect to the design of FFNN.

2.5. Examples

To illustrate the technique proposed in the preceding sections we give examples to demonstrate the behavior and properties of proposed design, the programs are written with MATLAB 7.11.

We used 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.

For the test problems the analytic solution $y_a(x)$ was known in advance. Therefore we test the accuracy of the obtained solution by computing the error: $E(x) = | y_t(x) - y_a(x) |$.

In order to illustrate the characteristics of the solution provided by the suggested design, we provide figures displaying the corresponding error E(x) both at the few points (training points) that were used for training and at many other points (test points) of the domain of each 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 2.1

Consider the following 2nd order regular TPBVP :

$$\frac{\mathbf{d^2y}}{\mathbf{dx^2}} - \mathbf{9x}\frac{\mathbf{dy}}{\mathbf{dx}} + \mathbf{25y(x) = 0}$$

with Dirishlit BC: $y(0) = 0$, $y(1) = 1$, $x \in [0, 1]$.

The analytic solution is: $y_a(x) = x^5$, according to the equation (2.8) 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 (2.1) 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 (2.1) and its errors gave in table (2.2), table (2.3) gave the performance of the train

for epoch and time, table (2.4) gave the initial weight and bias of the design network.

Table2.1: Analytic and Neural solution of example 2.1

In-put	Analytic solution	Output of suggested FFNN $y_t(x)$ for different training algorithm		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	0	3.18275672804846e-05	4.35445013380331e-11	8.31618490448882e-06
0.1	e-05 1.00000000000000	1.00000000001765e-05	0.000186674070754123	e-052.85819893857120
0.2	000320000000000000.	0.000256457971210211	0.000391087630215781	0.000232913430622261
0.3	0.00243000000000000	0.00243000000000004	0.00243000000377580	0.00240520621257612
0.4	0.0102400000000000	0.0103018040271908	0.0102399999671248	0.0103474110758739
0.5	0.0312500000000000	0.0312499999999999	0.0312499999220488	0.0313038651973674
0.6	0.0777600000000000	0.0777600000000002	0.0777600000012999	0.0776815784037241

0.7	0.168070000000000	0.168242776025208	0.168042712286393	0.168133137710980
0.8	0.327680000000000	0.327680000000000	0.327680000004937	0.327646281291338
0.9	0.590490000000000	0.590490000000000	0.591391796585037	0.590501796346579
1.0	1	1	0.999999999974204	0.999997845646038

Table 2.2: Accuracy of solutions for example 2.1

The error $E(x) = \| y_t(x) - y_a(x) \|$ where $y_t(x)$ computed by the following training algorithm		
trainlm	trainbfg	Trainbr
3.18275672804846e-05	4.35445013380331e-11	8.31618490448882e-06
1.76531830603163e-16	0.000176674070754123	1.85819893857120e-05
6.35420287897890e-05	7.10876302157806e-05	8.70865693777388e-05
4.16333634234434e-17	3.77579911781112e-12	2.47937874238852e-05
6.18040271908281e-05	3.28752337852567e-11	0.000107411075873889
1.11022302462516e-16	7.79512010495864e-11	5.38651973673865e-05
1.80411241501588e-16	1.29991850617017e-12	7.84215962758433e-05
0.000172776025207599	2.72877136067162e-05	6.31377109800990e-05
0	4.93727281281053e-12	3.37187086617563e-05
1.11022302462516e-16	0.000901796585036974	1.17963465786630e-05
0	2.57959209548631e-11	2.15435396155872e-06

Table 2.3: The performance of the train for epoch and time

Train Function	Performance of train	Epoch	Time	MSE
Trainlm	1.29e-32	83	0:00:01	3.16e-009
Trainbfg	1.39e-21	822	0:00:18	7.7295e-008
Trainbr	6.81e-27	462	0:00:11	1.8260e-009

Table 2.4: 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}
0.3012	0.5822	0.3619
0.9506	0.1531	0.7248
0.4606	0.0731	0.8583
0.2876	0.5806	0.3479
0.0846	0.2870	0.9617

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}
0.9541	0.1820	0.6427
0.5428	0.0930	0.0014
0.5401	0.4635	0.0304
0.3111	0.0093	0.2085
0.0712	0.9150	0.4550

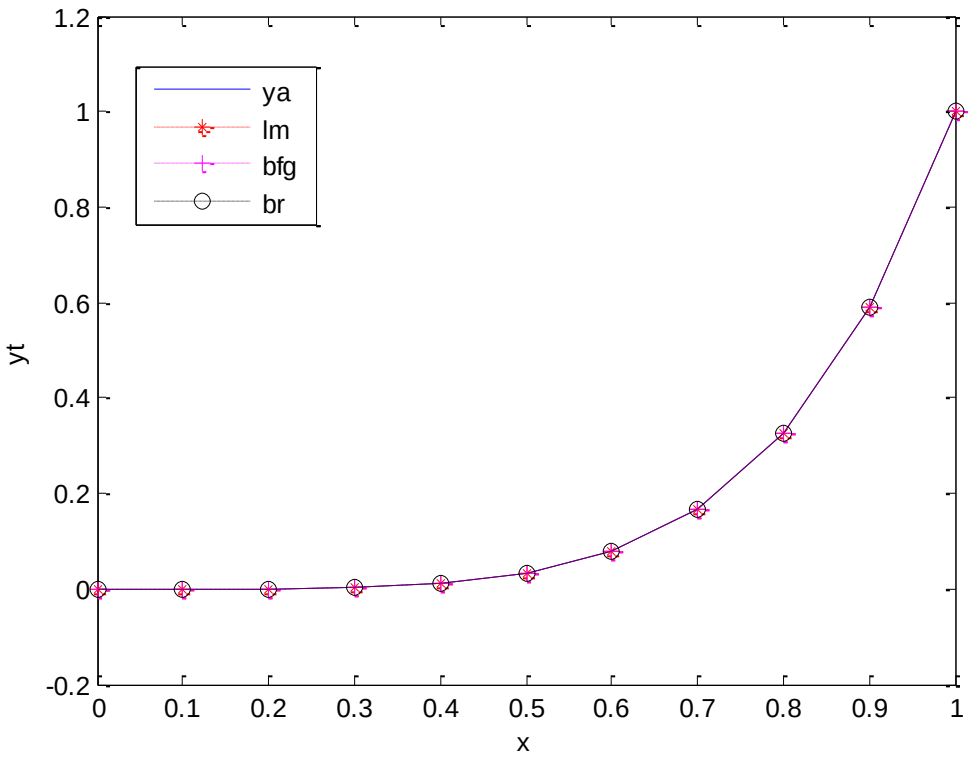

Figure 2.1: analytic and neural solution of example 2..1 using : trainbfg, trainbr, and trainlm training algorithm

Example 2.2

Consider the following periodic 2nd order regular TPSBVP:

$$\frac{d^2y}{d^2x} + \left(\frac{1}{x}\right)\frac{dy}{dx} + \cos(x) + \frac{\sin(x)}{x} = 0$$

with mixed BC: y'(0) = 0 , y(1) = cos1 and x ∈ [0, 1]. The analytic solution is: $y_a(x) = \cos(x)$ (see [62]), according to the equation (2.8) the trial neural form of the solution is taken to be: $y_t(x) = (\cos 1)\, x + x\,(x - 1)\, N(x, p)$.

The FFNN trained using a grid of ten equidistant points in [0, 1]. Figure (2.2) display the analytic and neural solutions with different training algorithm. The neural results with different types of training algorithm such as: trainlm, trainbfg, trainbr, introduced in Table (3.6) and its errors gave in Table (2.7), Table (2.8) gave the weight and bias of the design network, Table (2.9) gave the performance of the train with epoch and time.

Table 2.6: Analytic and neural solution of Example 2.2

In-put	Analytic solution	Out of suggested FFNN $y_t(x)$ for different training algorithm		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	**1**	**1**	**1.00026931058869**	**1.00000028044679**
0.1	**0.995004165278026**	**0.994992000703617**	**0.995004165151447**	**0.995003016942414**
0.2	**0.980066577841242**	**0.980066577841242**	**0.980038331595444**	**0.980068202697384**
0.3	**0.955336489125606**	**0.955336489125606**	**0.955326539544390**	**0.955327719921018**
0.4	**0.921060994002885**	**0.921060994002885**	**0.921060993873107**	**0.921057767471849**

0.5	**0.877582561890373**	**0.877582561890373**	**0.877583333411869**	**0.877587507015204**
0.6	**0.825335614909678**	**0.825335614909678**	**0.825335614867241**	**0.825332390929038**
0.7	**0.764842187284489**	**0.764838404395133**	**0.764842187227892**	**0.764831348535762**
0.8	**0.696706709347165**	**0.696656420761032**	**0.696706709263878**	**0.696708274506624**
0.9	**0.621609968270664**	**0.621454965504409**	**0.621609968278774**	**0.621608898908218**
1.0	**0.540302305868140**	**0.540302305868140**	**0.540302305877365**	**0.540302558954826**

Table 2.7: Accuracy of solution for Example 2.2

The error $E(x) = \| y_t(x) - y_a(x) \|$ where $y_t(x)$ computed by the following training algorithm		
t r a i n l m	t r a i n b f g	trainbr
0	0.000269310588686622	**2.80446790679179e-07**
1.21645744084464e-05	**1.26578747483563e-10**	**1.14833561148942e-06**
0	**2.82462457977806e-05**	**1.62485614274566e-06**
0	**9.94958121636191e-06**	**8.76920458825481e-06**
0	**1.29778077173626e-10**	**3.22653103579373e-06**
0	**7.71521496467642e-07**	**4.94512483140142e-06**
0	**4.24377200047843e-11**	**3.22398064012130e-06**
3.78288935598548e-06	**5.65963942378289e-11**	**1.08387487263162e-05**
5.02885861338731e-05	**8.32875990397497e-11**	**1.56515945903823e-06**
0.000155002766255130	**8.10940203876953e-12**	**1.06936244681499e-06**

0	**9.22539822312274e-12**	**2.53086685830795e-07**

Table 2.8: Initial weights 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}
0.2858	0.0759	0.1299
0.7572	0.0540	0.5688
0.7537	0.5308	0.4694
0.3804	0.7792	0.0119
0.5678	0.9340	0.3371

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

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

0.1951	0.1792	0.3766
0.3268	0.9689	0.8772

Table 2.9 :The performance of the train with epoch and time

Train Function	Performance of train	Epoch	Time	MSE
Trainlm	**0:00**	**75**	**0:00:01**	**2.1859e-009**
Trainbfg	**6.42e-21**	**6187**	**0:04:23**	**6.6751e-009**
Trainbr	**5.88e-12**	**1861**	**0:0030**	**2.0236e-011**

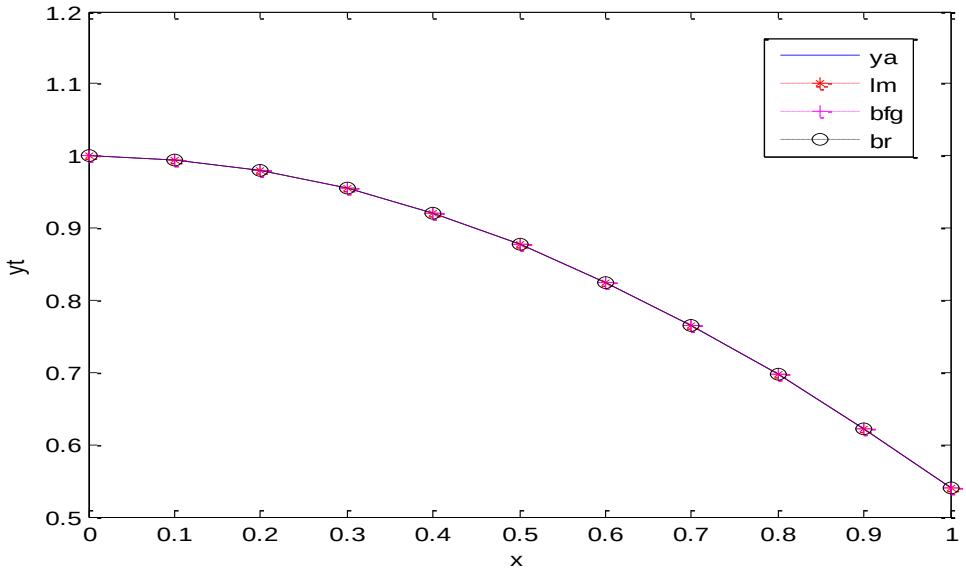

Figure 2.2: Analytic and neural solution of example 2.2 using: trainbfg, trainbr, trainlm.

2.6. Discussion

Based on our examples, we see that the application of Levenberg – Marquardt appears to be the fastest method for training moderate-sized FFNN (up to several hundred weights). It also has a very efficient MATLAB implementation, since the solution of the matrix equation is a built–in function, so its attributes become even more pronounced in a MATLAB setting.

The disadvantage of the Bayesian regularization method is that it generally take longer to converge than early stopping.

Networks are also sensitive to the number of neurons in their hidden layers, too few neurons can lead to under-fitting and too many neurons can contribute to over-fitting, in which all training points are well fit, but the fitting curve take swirled oscillations between these points.

References

[1] A. Blum and R. Rivest, "Training a 3-node neural network is NP-complete", Proc. of the 1988Workshop on Computational Learning, pp. 9-18, 1988.

[2] A. Junaid, M. A. Z. Raja, and I. M. Qureshi," Evolutionary Computing Approach for The Solution of Initial value Problems in Ordinary Differential Equations", World Academy of Science, Engineering and Technology , No.55 , PP:578 – 581, 2009.

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

[4] A. Malek and R. S. Beidokhti, "Numerical solution for high order differential equations using a hybrid neural network-optimization method ", Applied Mathematics and Computation ,Vol.183, pp: 260–271, 2006.

[5] B. Kröse and P. van der Smagt , " An introduction to Neural Networks ", Eighth edition , Amsterdam, November 1996 .