

Republic of Iraq  
Ministry of Higher Education  
And Scientific Research  
Al-Qadisiya University  
Collage of Education  
Department Of mathematics



# Solve Perturbation Problems By Neural Network

A reasarch

Submitted to the department of mathematics college of education  
University AL-Qadisiya as a partial fulfillment of the requirement for  
the degree of bachelor of science in mathematics

By

Dhuha Nazar Obeid

Supervised By

Dr Khalid Mindeel Mohammed

# Chapter One

## **Mathematical Background**

### **1. Introduction**

This chapter introduces a brief presentation of artificial neural networks (ANN's), to choose suitable network, that is suitable designer and architecture to implement the problem of this thesis, also introduce a brief definition and presentation of perturbation problems, singular perturbation problems, which helps us to design a suitable ANN's to solve suggested problem.

## **Section One: Artificial Neural Networks**

### **1.1.1.Introduction**

A Neural Network is a powerful modeling tool that is able to capture and represent complex input/output relationships. The motivation for the development of neural network technology stemmed from the desire to develop an artificial system that could perform "intelligent" tasks similar to those performed by the

human brain. Neural networks resemble the human brain in the following two ways:

- A neural network acquires knowledge through learning.
- A neural network's knowledge is stored within inter-neuron connection strengths known as synaptic weights. [21]

### **1.1.2.What is Artificial Neural Networks?**

Artificial neural network is designed to perform assignments in the similar manner that human brain works. An ANNs consists of a number of simple and processor units, called neurons, which are identical to the biological neurons in the brain. The neurons are connected by links passing signals from one to another neuron these links called the weights. Each neuron receives input signals during its connections; but, it never produces more than a single output signal. The output signal is sent during the neuron's outgoing connection which is corresponding to the biological axon. The outgoing connection, in turn, rifting into a number of laterals that send the same signal. The outgoing laterals terminate at the incoming connections of other neurons in the ANN. Figure (1.1)illustrates the connections of ANN

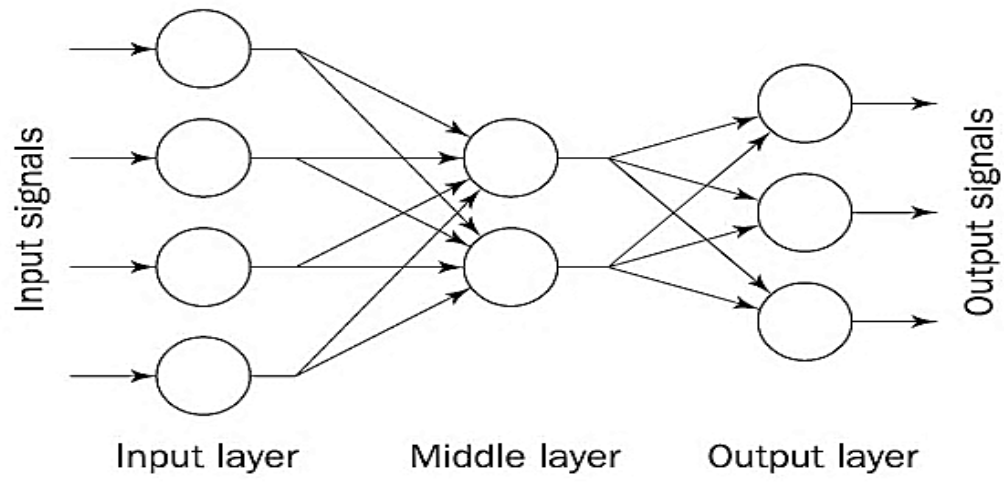


Figure 1.1: Architecture of ANN

In the same way of the human brain which have been designed the component of computing unit, ANN could perform machine learning, which involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by symmetry and thus improve the performance of the intelligent system over time.

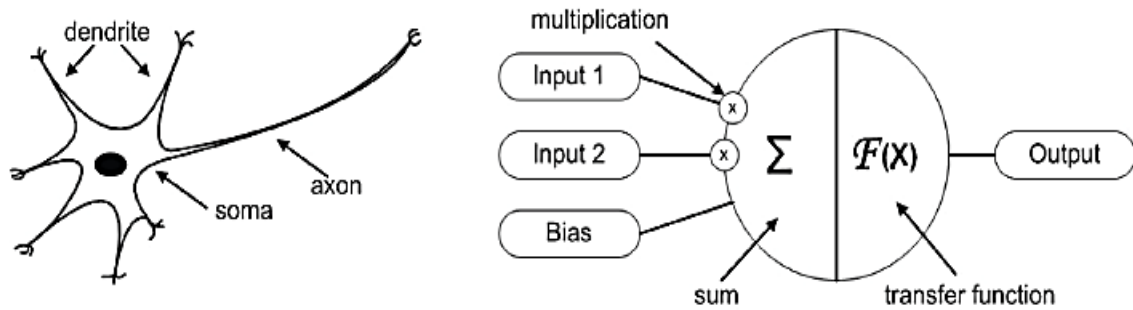
It is worth noting that in order to perform machine learning, three essential features are:

- Neurons, as mentioned above, the basic computing elements;
- Network architecture that describes the connections between computing units, and;
- Training (learning) algorithm that is used to find the values of the network parameters for performing particular task.

### **1.1.3. Artificial Neuron**

Artificial neuron is a basic building unit of every ANN. Its design and functionalities are derived from observation of a biological neuron that is basic

building unit of biological systems which includes the brain, spinal cord and peripheral ganglia [20]. Figure (1.2) illustrates the similarities in design and functionalities which can be seen where the left side of a figure represents a biological neuron with its soma, dendrites and axon and where the right side of a figure represents an artificial neuron with its inputs, weights, transfer function, bias and outputs.



**Figure1.2:Similarity between biological and artificial neuron.**

In case of biological neuron information comes into the neuron via dendrite, soma processes the information and passes it on via axon. In case of artificial neuron the information comes into artificial neuron via inputs that are weighted, each input can be individually multiplied with a weight.

Then the artificial neuron sums the weighted inputs and bias then processes the sum with a transfer function. At the end an artificial neuron passes the processed

information via output(s). The mathematical description of artificial neuron model can be seen as[5]:

$$y = F \left( \sum_{i=0}^m w_i x_i + b \right) \quad (1.1)$$

where:

$x_i$ : input value,  $w_i$ : weight value,  $b$ : bias,

$F$ : transfer function,  $y$ : output value.

As seen from a model of an artificial neuron and its equation (1.1) the major unknown variable of our model is its transfer function.

### 1.1.6. Feed Forward Neural Networks

An ANN with feed forward topology is called feed forward neural network (FFNN) and has only one worked formula: information must flow from input to output in only one direction with no back loops , that is, the output from one layer of neurons feeds forward into the next layer of neurons. There not any backward connections, and connections never skip a layer(see Figure 1.4) [43]. There are no limitations on number of layers, type of transfer function used in individual



artificial neuron or number of connections between individual artificial neurons [56].

## **Section Two: Perturbation Problems**

### **1.2.1.Introduction**

The term "perturbation problem" is generally used in mathematics when one deals with the following situation: There is a family of problems depending on a small parameter  $0 < \epsilon \ll 1$ , perturbation problems governing mathematical models appear in many interesting applications in sciences and engineering fields such: mechanical engineering, hydraulics and chemical engineering. Similarly, studies of heat transfer, diffusion, and evaporation in moving fluids were greatly aided by the

knowledge of the boundary layer flow [42]. The above information indicates that there has been some successful interaction between scientists and engineers which has resulted in an almost exponential growth of research on boundary layer flow in recent years. To this end, it will be paramount to highlight an account of some recent developments (both theoretical and applied) on non-linear perturbation models. We include some of the works that use the perturbation techniques to solve some other problems in recent years. There are two types of Perturbation Problems, regular perturbation problems and singular perturbation problems, in this thesis, will focus on singular perturbation problems (SPP).

### **1.2.2. Singular Perturbation Problems**

Singular perturbation problems (SPP) are common in applied sciences and engineering, for example, fluid dynamics, quantum mechanics, chemical reactions, electrical networks, chemical kinetics, elasticity, aerodynamics, plasma dynamics, magneto-hydrodynamics and other domains of the world of fluid motion [27].

Perturbed problems are known to be the problems consist a small parameter  $\varepsilon$  multiplied by one or more parts of the problem.

Singularly Perturbed problems are known to be the problems in which the highest order in derivative term is multiplied by a small parameter  $\varepsilon$  and this parameter is known as the perturbation parameter. A singularly perturbed differential equation is a differential equation in which the highest derivative is multiplied by a small parameter  $\varepsilon$ . The general form of singular perturbation problems for ODE which containing a small positive parameter  $\varepsilon$ ,  $0 < \varepsilon \ll 1$ , has the form (in case 2<sup>nd</sup> order):

$$\varepsilon y''(x) = f(x, y, y', \varepsilon), \quad x \in [a, b], \quad (1.7)$$

Where  $f$  is in general nonlinear functions of their arguments, and

$$f(x, y, y', \varepsilon) \in C^3([a, b] \times \mathbb{R}^2 \times [0, 1])$$

$$\frac{\partial f}{\partial \varepsilon}(x, y, y', \varepsilon) \neq 0, \quad (x, y, y', \varepsilon) \in ([a, b] \times \mathbb{R}^2 \times [0, 1]),$$

This problem has been treated by several authors in the last years for more details see [42], [2], [3], [13], [41], [55], [30], [7], [12] and their references.

### 1.2.3. Regular Perturbation Problems

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

The 2<sup>nd</sup> order regular perturbation problem has the form:

$$y''(x) = f(x, y, y', \varepsilon), x \in [a, b], 0 < \varepsilon \ll 1; \quad (1.8)$$

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

$$f(x, y, y', \varepsilon) \in C^3([a, b] \times \mathbb{R}^2 \times [0, 1])$$

$$\frac{\partial f}{\partial \varepsilon}(x, y, y', \varepsilon) \neq 0, (x, y, y', \varepsilon) \in ([a, b] \times \mathbb{R}^2 \times [0, 1]),$$

**Note**, assume that our problem contains only one small, positive parameter  $\varepsilon$  ( $0 < \varepsilon \ll 1$ ), denote the problem by  $P_\varepsilon$ . What happens if  $\varepsilon \rightarrow 0$ ? we have the reduced problem  $P_0$ . We want to study the relationship between the solution of  $P_\varepsilon$  and the solution of  $P_0$  under appropriate assumptions. A perturbation problem (1.7) is called SPP, if  $\varepsilon \rightarrow 0$ , the solution  $y_\varepsilon(x)$  converges to  $y_0(x)$  only in some interval of  $x$ , but not throughout the entire interval, thus giving rise to the "boundary layers" phenomena at both end-points [33].

#### 1.2.4. Differential Equations and Perturbation Problems

An ordinary differential equation(ODE) is an equation containing a function of one independent variable and its derivatives.

There are many general forms of an ODE's can take, and these are classified in practice. The derivatives are ordinary because partial derivatives only apply to functions of many independent variables and said to be partial differential equation (PDE).

The problems of ODE are classified into initial value problems(IVP) and boundary value problems (BVPs), depending on how the conditions at the endpoints of the domain are specified. All the conditions of an initial value problem are specified at the initial point of the domain. On the other hand, the problems becomes a boundary value problem if the conditions are needed for both initial and final points of the domain [48].

There are many interesting regions for which to specify boundary conditions. One deal in this thesis is to solve perturbation problems as the form:

$$y'' = f(x, y, y', \epsilon); \quad a \leq x \leq b, \quad (1.9)$$

with specific boundary condition.

There are many types of the boundary conditions [49]:

- If the boundary gives a value to the normal derivative of the problem then it is a Neumann boundary condition.

- If the boundary gives a value to the problem then it is a Dirichlet boundary condition.
- If the boundary has the form of a curve or surface that gives a value to the normal derivative and the problem itself then it is a Cauchy boundary condition or mixed condition.

## Chapter Two

### **Suitable Architecture Of Networks To Solve Singular Perturbation Problems**

#### **2.1. Introduction**

Many methods have been developed to solve this problem. Some of them give the solution at a selected group of points, others give the solution in analytic

form by using basis functions and transform the original problem usually to a system of algebraic equations.

Most of the previous studies in solving SPPs using ANNs was restricted to the case of solving the systems of algebraic equations which we resulted from the discretization of the domain. In this chapter we suggest suitable FFNNs as a method to solve SPPs with initial or boundary conditions. To demonstrate the applicability of the suggested design, we solved several model examples and presented the computational results.

The computed results have been compared with the solution to show the accuracy and efficiency of the suggested networks.

## 2.2. Description the Design of the Neural Networks

In this section, we will illustrate how can be design suitable neural network to find the solution of the SPPs, the general form of a non-linear second order SPPs is:

$$\epsilon \frac{d^2 y}{dx^2} = F(x, y, y'), \quad x \in R, \quad (2.1)$$

subject to IC's or BC's and  $y(x)$  is the solution to be computed ,where  $\epsilon \ll 1$ , and the function  $f(x, y, y')$  is a non-linear function of  $y$  satisfies:

$$f(x, y, y') \in C^3([a, b] \times \mathbb{R}^2)$$

$$\frac{\partial f}{\partial \varepsilon}(x, y, y') \neq 0, (x, y, y') \in ([a, b] \times \mathbb{R}^2), \quad a, b \in \mathbb{R},$$

In the suggested network the design expressed as summing of two parts: the first part satisfies the IC or BC. The second part can be found by using suggested FFNN which is trained so as to satisfy the suggested problem and such technique called collocation neural network. That is:

$$y_t(x, p) = y_a(x) + G(x, \varepsilon, N(x, p)) \quad (2.2)$$

where  $N(x, p)$  is output of the FFNN with coefficients  $p$  and  $n$  input units fed with the input data  $x$ , the second part  $G$  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.

### 3.3. Architect Suggested Networks

We suggest multilayer feed forward neural network with one hidden layer which can approximate any function to arbitrary accuracy to solve SPPs.

### 3.5. Choosing the Initial Weights



If we treat all the weights in the same way in the training algorithm, that is if we start all the weights as the same, the hidden units at the end the same value and the suggested network will not learned well. For this reason, we generally start all the weights and biases with small random values. Usually we take the weights around zero  $[-1, 1]$ , many researchers choose the weights by using Gaussian distribution around zero [14].

### 3.6. Illustration of the Architect FFNN for Solving SPPs With ICs

Consider the 2<sup>nd</sup> order non-linear SPPs:

$$\varepsilon \left[ \begin{array}{c} \text{[Red X icon]} \\ \text{[Empty box]} \end{array} \right], \varepsilon), \quad x \in [a, b], \quad (3.7)$$

With initial condition (IC):  $y(a) = A, y'(a) = A'$ ; where  $A, A' \in \mathbb{R}$ .

A trial solution  $y_t$  can be written as:

$$y_t(x, p) = A + A'(x - a) + (x - a)^2 N(x, p) \quad (3.8)$$

Where  $N(x, p)$  is the output of the FFNN with one input unit for  $x$  and weight  $p$ .

The amount of error which must be minimized is given as:

$$E[p] = \sum_{i=1}^n \left\{ \frac{d^2 y_t(x_i, p)}{dx^2} - f(x_i, y_t(x_i, p), \frac{dy_t(x_i, p)}{dx}, \varepsilon) \right\}^2 \quad (3.9)$$

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

$$\begin{aligned} \frac{dy_t(x, p)}{dx} &= A' + (x - a)^2 \frac{dN(x, p)}{dx} + 2(x - a)N(x, p) \\ \frac{d^2 y_t(x, p)}{dx^2} &= (x - a)^2 \frac{d^2 N(x, p)}{dx^2} + 2N(x, p) + 4(x - a) \frac{dN(x, p)}{dx} \end{aligned} \quad (3.10)$$

It is easy to evaluate the gradient of the performance with respect to the coefficient of the network.

### 3.7. Illustration of the Architect FFNN for Solving SPPs with BCs

Consider the 2<sup>nd</sup> order non-linear SPBVPs:

$$\varepsilon \left[ \begin{array}{c} \text{ } \\ \text{ } \end{array} \right], \varepsilon), x \in [a, b], \quad (3.11a)$$

With boundary conditions (BC):

$$\text{In the case of Dirichlet BC: } y(a) = A, y(b) = B; \quad (3.11b)$$

$$\text{In the case of Neumann BC: } y'(a) = A, y'(b) = B; \quad (3.11c)$$

$$\text{In the case of Cauchy or mixed BC: } y(a) = A, y'(b) = B, \quad (3.11d)$$

$$\text{Or } y'(a) = A, y(b) = B; \text{ where } A, B \in \mathbb{R} \quad (3.11e)$$

a trialist solution  $y_t$  can be written as:

$$y_t(x,p) = \frac{(bA-aB)}{b-a} + \frac{(B-A)}{b-a}x + (x-a)(x-b)N(x,p,\varepsilon), \quad (3.12)$$

where  $N(x, p, \varepsilon)$  is the output of the suggested FFNN with one input unit for  $x$  and weight  $p$ .

The amount of error which must be minimized is given as:

$$E[p] = \sum_{i=1}^n \left\{ \frac{d^2 y_t(x_i, p)}{dx^2} - f(x_i, y_t(x_i, p), \frac{dy_t(x_i, p)}{dx}, \varepsilon) \right\}^2, \quad (3.13)$$

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

$$\frac{dy_t(x,p)}{dx} = \frac{A-B}{b-a} + \{(x-a) + (x-b)\} N(x, p, \varepsilon) + (x-a)(x-b) \frac{dN(x,p,\varepsilon)}{dx}, \quad (3.14)$$

and

$$\frac{d^2 y_t(x,p)}{dx^2} = 2N(x, p, \varepsilon) + 2\{(x-a) + (x-b)\} \frac{dN(x,p,\varepsilon)}{dx} + (x-a)(x-b) \frac{d^2 N(x,p,\varepsilon)}{dx^2}, \quad (3.15)$$

It is easy to evaluate the gradient of the performance with respect to the coefficient.

### 3.9. Examples

To illustrate the suggested FFNN, we give many examples to demonstrate the behavior and efficiency of the suggested design; the programs are written with MATLAB version 7.12.

We suggest FFNN having 5 hidden units in one hidden layer and linear output unit. The sigmoid function of the hidden units is tansig.

We test the performance of the solutions by computing the absolute error:

$$E(x) = | y_t(x) - y_a(x) |.$$

We begin with linear problems.

### 3.9.1. Linear Examples

We beginning with SPPs consist perturbed IC, i.e., the IC contain perturbed number

#### Example 3.1

Consider the following 2<sup>nd</sup> order non homogeneous SPPs:

$$y'' + \varepsilon^3 e^{x^2} y' + 4(2 - x) = x^2 e^x, \quad x \in [0, 1]$$

with IC:  $y(0) = 1$  ,  $y'(0) = \frac{1}{\varepsilon}$  and the analytic solution is [6]:

$$y_a(x) = \frac{x^2 e^x}{2-x} + 3e^{\frac{-x}{\varepsilon}} - 2e^{\frac{-2x}{\varepsilon}},$$

depending on the equation (3.8), the suggested FFNN has the solution is:

$$y_t(x) = 1 + \frac{x}{\varepsilon} + x^2 N(x, p).$$

“ The FFNN trained using a grid of equidistant points in  $[0,1]$ , we get  $\varepsilon = 10^{-4}$ , Figure (3.1) illustrates the analytic and neural solution in the training set. Then oral result with training algorithms: Bayesian Regulation (trainbr), Levenberg – Marquardt (trainlm), quasi – Newton (trainbfg), and modified Levenberg – Marquardt (trainnml) are gave in Table (3.1) and its errors in Table (3.2), Table (3.3) gave the accuracy of the train for epoch and time compared with the mean square error of numerical method used in [6] to solve this example, Table (3.4) gave the initial weight and bias of the FFNN.

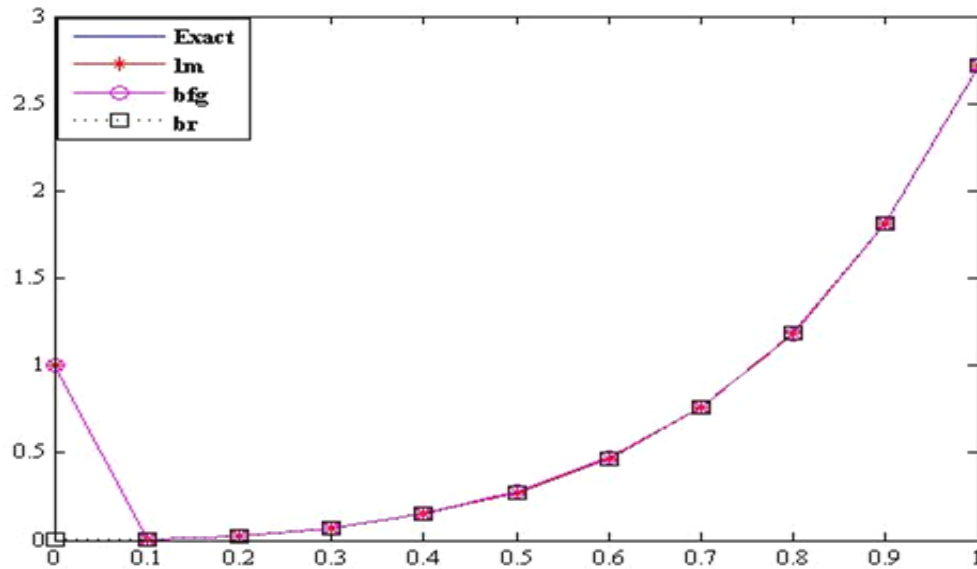


Figure3.1: analytic and neural solution of example 3.1, with  $\varepsilon = 10^{-4}$ .

input	Analytic solution	Solution of FFNN $y_t(x)$ for training algorithm
-------	-------------------	--

Table 3.1: Analytic and Neural solution of example 3.1,  $\varepsilon = 10^{-4}$

x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	1	0.999640976175370	1.00000060781947	0.000490306650163041
0.1	0.00581668904250341	0.00584858735571990	0.00581748398780408	0.00581682218438316
0.2	0.0271422835146705	0.0262235919926366	0.0260890754780599	0.0270766508893611
0.3	0.0714631133422590	0.0720379285133504	0.0714627704412003	0.0714583209579391
0.4	0.149182469764127	0.148860377994119	0.149182398207817	0.149199034176640
0.5	0.274786878450021	0.273944920310229	0.284179642578833	0.274774551828534
0.6	0.468544834386131	0.469737313059561	0.478528518159394	0.468532552107991
0.7	0.759029866661949	0.762296279887764	0.759030222208309	0.759056497736270
0.8	1.18695516186265	1.18678423417625	1.17927557251113	1.18693667442537
0.9	1.81116229094284	1.81124271936535	1.81116317468505	1.81116773829719
1.0	2.71828182845905	2.72006218179475	2.71828275379187	2.71828042684631

Table 3.2: Accuracy of solutions of example 3.1,  $\varepsilon = 10^{-4}$ 

The error $E(x) =  y_t(x) - y_a(x) $ where $y_t(x)$ computed by the following training algorithm		
Trainlm	Trainbfg	Trainbr
0.000359023824629823	6.07819470532789e-07	0.999509693349837
3.18983132164890e-05	7.94945300671215e-07	1.33141879750387e-07
0.000918691522033870	0.00105320803661058	6.56326253093281e-05
0.000574815171091411	3.42901058700273e-07	4.79238431989881e-06
0.000322091770007971	7.15563100717187e-08	1.65644125127151e-05
0.000841958139792765	0.00939276412881179	1.23266214873685e-05
0.00119247867343003	0.00998368377326325	1.22822781396525e-05
0.00326641322581489	3.55546359687153e-07	2.66310743208820e-05
0.000170927686401257	0.00767958935151758	1.84874372832766e-05
8.04284225008889e-05	8.83742201640558e-07	5.44735434981902e-06
0.0017803533569948	9.25332822809821e-07	1.40161273609607e-06

Table 3.3: The accuracy of the train of suggested FFNN,  $\varepsilon = 10^{-4}$ ,

Train function	Performance of train	Epoch	Time	Msereg.	MSE of Numerical Method in [6]
Trainlm	4.21e-31	121	0:00:01	1.583069324864220e-06	8.241e-2
Trainbfg	4.38e-19	140	0:00:02	2.254393657527956e-05	
Trainbr	1.87e-10	404	0:00:04	0.090819966644612	

**Table 3.4: Initial weight and bias of FFNN,  $\varepsilon = 10^{-4}$**

Initial weights and bias for trainlm		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.6393	0.4332	0.6240
0.9173	0.8842	0.3279
0.1615	0.3930	0.802
0.7156	0.1789	0.9994
0.5777	0.6333	0.9809

Initial weights and bias for trainbfg		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.4429	0.0323	0.8110
0.0530	0.5570	0.1386
0.0878	0.7198	0.8818
0.7979	0.1104	0.9235
0.6555	0.2166	0.0127

Initial weights and bias for trainbr		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.5502	0.2865	0.1692
0.1805	0.0773	0.4304
0.6784	0.9005	0.4162
0.0556	0.8466	0.7287
0.0340	0.3956	0.4064

### 3.9.2. Nonlinear Examples

#### Example 3.4

Consider the following 2<sup>nd</sup> order nonlinear SPP:

$$\varepsilon y'' + y' + y^2 = 0$$



with Dirishlit BC's:  $y(0) = 0, y(1) = 1/2$  and the analytic solution is[27]:

$$y_a(x) = \frac{1}{1+x} - \frac{e^{-x/\varepsilon}}{(1+x)^2},$$

Depending on equation (3.12) the neural solution is:

$$y_t(x) = \frac{1}{2} x + (x - 1)N(x, p)$$

The FFNN training using a grid of ten equidistance points in  $[0,1]$ , we get  $\varepsilon=10^{-5}$  Figure (3.8) illustrates the analytic and neural solutions with different training algorithm. The neural results with different types of training algorithm such as: trainlm, trainbfg, trainbr, are introduced in Table (3.32) and its errors are given in Table (3.33), Table (3.34) gives the accuracy of the training for epoch and time, finally Table (3.35) gives the initial weight and bias of the FFNN.

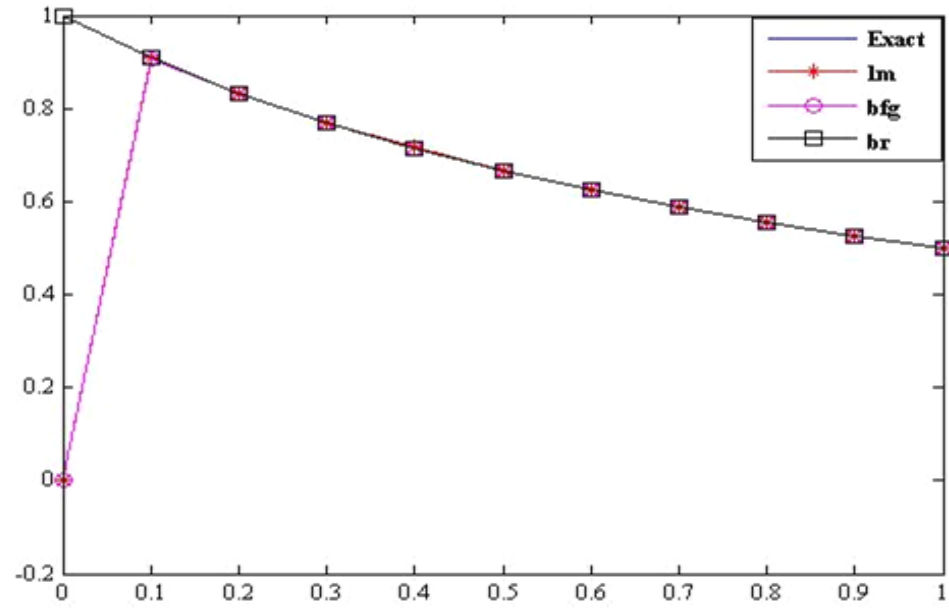


Figure3.8: analytic and neural solution of example 3.4 ,with  $\varepsilon = 10^{-5}$ .

Table 3.32: Analytic and Neural solution of example 3.4 ,  $\varepsilon = 10^{-5}$

input	Analytic solution	Solution of FFNN $y_t(x)$ for different training algorithm		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	0	2.01858731750028e-16	-7.46042284111519e-07	0.999394605934724
0.1	0.909090909090909	0.909090909090908	0.907012437275230	0.909090899405227
0.2	0.833333333333333	0.833333333333334	0.833316323016675	0.833333390984059
0.3	0.769230769230769	0.769230769230769	0.769224640766005	0.769222734844288
0.4	0.714285714285714	0.715802392679189	0.714266006331668	0.714285333857831
0.5	0.666666666666667	0.667307331279249	0.666704092932498	0.666667572440800
0.6	0.625000000000000	0.625000000000000	0.625022350272257	0.624999233072860
0.7	0.588235294117647	0.588235294117647	0.588206083448708	0.588235168678295
0.8	0.555555555555556	0.555555555555556	0.555530665155201	0.555556321619597
0.9	0.526315789473684	0.525705170771698	0.526340751988947	0.526315243648641
1.0	0.500000000000000	0.497859222777642	0.499993460412034	0.500000140628989

Table3.33: Accuracy of solutions for example 3.4 ,  $\varepsilon = 10^{-5}$ 

The error $E(x) =  y_t(x) - y_a(x) $ where $y_t(x)$ computed by the		
Trainlm	Trainbfg	Trainbr
2.01858731750028e-16	7.46042284111519e-07	0.999394605934724
7.77156117237610e-16	0.00207847181567955	9.68568181214380e-09
2.22044604925031e-16	1.70103166584612e-05	5.76507254157477e-08
1.11022302462516e-16	6.12846476422124e-06	8.03438648167010e-06
0.00151667839347458	1.97079540465994e-05	3.80427883417411e-07
0.000640664612582054	3.74262658315860e-05	9.05774133586057e-07
2.22044604925031e-16	2.23502722571656e-05	7.66927140127827e-07
3.33066907387547e-16	2.92106689391280e-05	1.25439352571810e-07
2.22044604925031e-16	2.48904003549155e-05	7.66064040980119e-07
0.000610618701986310	2.49625152629607e-05	5.45825043607451e-07
0.00214077722235811	6.53958796625886e-06	1.40628989231395e-07

**Table3.34: The accuracy of the train, with  $\varepsilon = 10^{-5}$**

TrainFunction	Performance of train	Epoch	Time	Msereg.
<b>Trainlm</b>	<b>3.58e-32</b>	<b>237</b>	<b>0:00:03</b>	<b>6.272629208206621e-07</b>
<b>Trainbfg</b>	<b>1.96e-23</b>	<b>478</b>	<b>0:00:07</b>	<b>3.538472650520739e-07</b>
<b>Trainbr</b>	<b>2.75e-13</b>	<b>576</b>	<b>0:00:06</b>	<b>0.081719147326782</b>

**Table 3.35: Initial weight and bias of the FFNN, with  $\varepsilon = 10^{-5}$**

Weights and bias for trainlm		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
<b>0.5238</b>	<b>0.0261</b>	<b>0.0073</b>
<b>0.2649</b>	<b>0.9547</b>	<b>0.6800</b>
<b>0.0684</b>	<b>0.4306</b>	<b>0.7060</b>
<b>0.4363</b>	<b>0.9616</b>	<b>0.6451</b>
<b>0.1739</b>	<b>0.7624</b>	<b>0.5523</b>

Weights and bias for trainbfg		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
<b>0.5406</b>	<b>0.8009</b>	<b>0.6618</b>
<b>0.4320</b>	<b>0.1425</b>	<b>0.1696</b>
<b>0.5427</b>	<b>0.4785</b>	<b>0.2788</b>
<b>0.7124</b>	<b>0.2568</b>	<b>0.1982</b>
<b>0.0167</b>	<b>0.3691</b>	<b>0.1951</b>

Weights and bias for trainbr		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
<b>0.4132</b>	<b>0.6154</b>	<b>0.1626</b>
<b>0.2178</b>	<b>0.7795</b>	<b>0.7968</b>
<b>0.8586</b>	<b>0.9548</b>	<b>0.1138</b>
<b>0.8610</b>	<b>0.9196</b>	<b>0.1588</b>
<b>0.2839</b>	<b>0.3848</b>	<b>0.3558</b>

