Journal of
Applied & Computational Mathematics

Tawfiq and Al-Abrahemee, J Appl Computat Math 2014, 3:3
DOI: 10.4172/2168-9679.1000160

Research Article Open Access

Design Neural Network to Solve Singular Perturbation Problems

Tawfiq LNM* and Al-Abrahemee KMM

Department of Mathematics, College of Education for Pure Science/Ibn Al-Haitham, University of Baghdad, Baghdad, Iraq

Abstract

The aim of this paper is to design neural network to present a method to solve Singular perturbation problems (SPP) by using network having one hidden layer with 5 hidden units (neurons) and one linear output unit, the sigmoid activation of each hidden units is tansigmoid. The neural network trained by the back propagation with different 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: Singularly perturbed problems; Neural network; Quasi-Newton; Levenberg-Marquardt, Bayesian regulation

Introduction

Singularly perturbed problems (SPP) are common in applied sciences and engineering. They often occur in, for example, fluid dynamics, quantum mechanics, chemical reactions, electrical networks, etc. A well known fact is that the solution of such problems has a multi scale character, i.e., there are thin transition layers where the solution varies very rapidly, while away from the layers the solution behaves regularly and varies slowly. For a detailed discussion on the analytical and numerical treatment of such problems one may refer to the Malley [1], Doolan et al. [2], Roos et al. [3], and Miller et al. [4]. Numerically, the presence of the perturbation parameter leads to difficulties when classical numerical techniques are used to solve such problems, this is due to the presence of the boundary layers in these problems, see for example [1,5]. Even in the case when only the approximate solution is required.

Many methods have been developed so far solving Singularly perturbed boundary value problems (SPBVP) , 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.

Singularly Perturbed Problems

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 $\epsilon > 0$, which we denote by P_ϵ, when $\epsilon=0$, we have the reduced problem P_o.

We want to study the relationship between the solution of P_ϵ and the solution of P_o under appropriate assumptions. The perturbation problem (PP), may consist of an ordinary differential equation, or a system of differential equations, dong with some given conditions which illustrate the problem. Thus, the general form of the 2nd order singularly perturbed problems (SPP):

$$P_\in = \frac{d\ \ y}{dx} = \quad (x, y, y', \in) \tag{1}$$

where f are n-dimensional vector functions, x is a scalar variable in a given interval.

A perturbation problem (1) is called SPP if $\epsilon \to 0$, the solution $y_\epsilon(x)$ converges to $y_o(x)$ only in some interval of x, but not throughout the entire interval, thus giving rise to an "boundary layers" phenomena at both end-points [6].

Artificial Neural Network

Artificial neural network is a simplified mathematical model of the human brain. It can be implemented by both electric elements and computer software. It is a parallel distributed processor with large numbers of connections; it is an information processing system that has certain performance characters in common with biological neural networks [7].

The arriving signals, called inputs, multiplied by the connection weights (adjusted) are first summed (combined) and then passed through a transfer function to produce the output for that neuron. The transfer (activation) function acts on the weighted sum of the neuron's inputs and the most commonly used transfer function is the sigmoid function (tansig.) [8].

There are two main connection formulas (types): feedback (recurrent) and feed forward connections. Feedback is one type of connection where the output of one layer routes back to the input of a previous layer, or to the same layer. Feed forward neural network (FFNN) does not have a connection back from the output to the input neurons [9]. There are many different training algorithms, but the most often used training algorithm is the back propagation (BP) rule. A

***Corresponding author:** Tawfiq LNM, Department of Mathematics, College of Education for pure science/Ibn Al-Haitham, University of Baghdad, Baghdad, Iraq, E-mail: drluma_m@yahoo.com

Received February 04, 2014; **Accepted** March 25, 2014; **Published** April 14, 2014

Citation: Tawfiq LNM, Al-Abrahemee KMM (2014) Design Neural Network to Solve Singular Perturbation Problems. J Appl Computat Math 3: 160 doi:10.4172/2168-9679.1000160

NN is trained to map a set of input data by iterative adjustment of the weights. Information from inputs is fed forward through the network to optimize the weights between neurons. Optimization of the weights is made by backward propagation of the error during training phase. The ANN reads the input and output values in the training data set and changes the value of the weighted links to reduce the difference between the predicted and target (observed) values. The error in prediction is minimized across many training cycles (iteration or epoch) until network reaches specified level of accuracy. A complete round of forward backward passes and weight adjustments using all input output pairs in the data set is called an epoch or iteration. In order to perform a supervised training we need a way of evaluating the ANN output error between the actual and the expected outputs. A popular measure is the mean squared error (MSE) or root mean squared error (RMSE) [10].

Structure of Neural Network

In an ANN expressions structure, architecture or topology, express the way in which computational neurons are organized in the network [11].

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 levels or 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.

Description of the Method

This section illustrates how our approach can be used to the approximation solution of the singular perturbation problem of the form:

$$\frac{d^2y}{dx^2} = F\left(x, y, y', \in\right) \qquad (2)$$

where x ϵ D, D ⊂ R denoted the domain and y(x) is the solution to be computed.

If yt (x, p) denoted a trial solution with adjustable parameters p, the problem is transformed to a discredited from:

$$\min_{p} \sum_{x_i \in D} F\left(x_i, y_t(x_i, p), y_t'(x_i, p), \in\right) \qquad (3)$$

In our proposed approach, the trial solution y_t employs an ANN and the parameters *p* correspond to the weights and biases of the neural architecture.

We choose a form for the trial function (*x*) such that achieved by writing as a sum of two terms:

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

where *N*(*x, p*) is ANN with parameters *p* and *n* input units fed with the input vector *x*. The term (*x*) contains no adjustable parameters and satisfies the BCs. The second term *G* is constructed so as not to contribute to the BCs, since (*x*) satisfy them. This term can be formed by using ANN whose weights and biases are to be adjusted in order to deal with the minimization problem.

Illustration of the Method

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

$$\frac{d^2y}{dx^2} = F\left(x, y, y', \in\right) \qquad (5)$$

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

$$y_t(x, p) = \frac{(bA - aB)}{(b - a)} + \frac{B - A}{b - a} x + (x - a)(x - b)\, N(x, p) \qquad (6)$$

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

The error quantity to be minimized is given by:

$$E(p) = \sum_{i=1}^{n} \left\{ \frac{d^2 y_t(x_i p)}{dx^2} - f\left(x_i, y_t(x_i, p), \frac{dy_t(x_i p)}{dx}\right) \right\}^2 \qquad (7)$$

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

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

$$\frac{d^2 y_t(x, p)}{dx^2} =_2 N(x, p) + 2\left\{(x - a) + (x - b)\right\}\frac{dN(x, p)}{dx} + (x - a) + (x - b)\frac{d^2 N(x, p)}{dx^2} \qquad (8)$$

It is straight forward to compute the gradient of the error with respect to the parameters p.

Numerical Result

In this section, we report numerical result, using amulti-layer ANN having one hidden layer with 5 hidden units (neurons) and one linear output unit. The sigmoid activation of each hidden unit is tansig, the analytic solution (*x*) was known in advance. Therefore, we test the accuracy of obtained solutions computing the mean square error (MSE).

Example 1

Consider the following 2nd order singular perturbation problem:

$$\in y''+yy'-y=0,\quad x\in[0,1]$$

BC's (Dirishlit case): y(0)=-1, y(1)=3.9995, and the analytic solution [12]:

$$y=x+c_1\tanh\left(c_1\frac{\frac{x}{\in}+c_2}{2}\right)$$

where,

$$c_1=2.9995$$

$$c_2=\frac{1}{c_1}\log\log(\frac{c_1-1}{c_2+1})s.t\in=10^{-5}$$

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

$$y_t(x)=-1+4.9995x+x(x-1)N(x,p)$$

The ANN trained using a grid of ten equidistant points in 0, 1. Figure 1 displays the analytic and neural solutions with different training algorithm. The neural results with different types of training algorithm such as: Levenberg–Marquardt (trainlm), quasi–Newton (trainbfg), Bayesian Regulation (trainbr) introduced in Table 1 and its errors gave in Table 2, Table 3 gives the performance of the train with epoch and time and Table 4 gives the weight and bias of the designer network.

Example 2

Consider the following 2nd order singular perturbed problem:

$$\in y''+2y'+e^{y}=0$$

with BC: y(0)=0, y(1)=0 and x∈ (0, 1).

$$y=\log\left(\frac{2}{1+x}\right)-\log(2)e^{\frac{-2x}{\in}},\quad \in=10^{-7}$$

The analytic solution is [12]:

The ANN trained using a grid of ten equidistant points in 0, 1. Figure 2 display the analytic and neural solutions with different training algorithm. The neural results with different types of training algorithm such as: Levenberg–Marquardt (trainlm), quasi–Newton (trainbfg), Bayesian Regulation (trainbr) introduced in Table 5 and its errors gave in Tables 6 and 7 gives the performance of the train with epoch and time and Table 8 gives the weight and bias of the designer network.

Conclusion

In this paper, we design neural network to solve singular perturbed problem. A fast and efficient algorithm (LM) for ANN with one hidden layer has been presented and tested on two examples. Through the

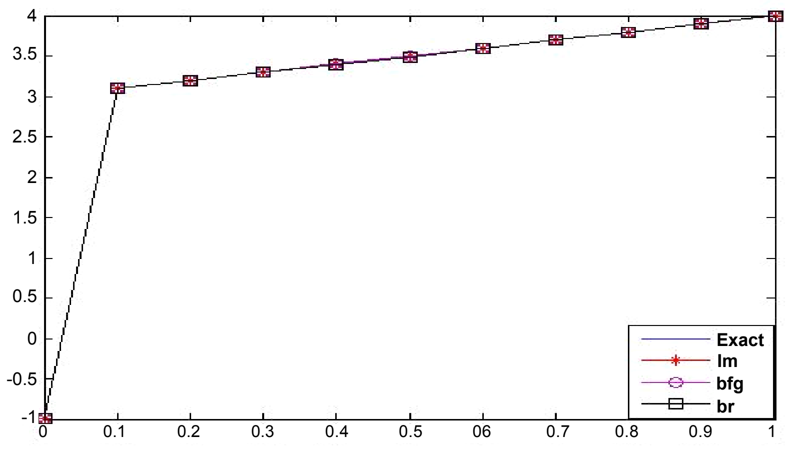

Figure 1: Analytic and neural solution of example1 using: trainlm, trainbfg,trainbr and traincgp training algorithm.

Input	Analytic solution	Out of suggested ANN $y_t(x)$ for different training algorithm			Numerical solution
x	**$y_a(x)$**	**Trainlm**	**Trainbfg**	**Trainbr**	
0.0	3.09950000000000	-0.999880745270845	-0.999880745270845	-0.999934351350428	-1.0000000
0.1	3.19950000000000	3.09952352215727	3.09952352215727	3.09928640827629	3.1035900
0.2	3.29950000000000	3.19982552466162	3.19982552466162	3.19982150310036	3.2011864
0.3	3.39950000000000	3.29890300366976	3.29890300366976	3.29912397768203	3.3003857
0.4	3.49950000000000	3.40001713095587	3.40001713095587	3.39987747119349	3.3999858
0.5	3.59950000000000	3.49974092352976	3.49974092352976	3.47742525061300	3.4997456
0.6	3.69950000000000	3.59934730598352	3.59934730598352	3.59853213880555	3.5995858
0.7	3.79950000000000	3.69942849140595	3.69942849140595	3.70080106004170	3.6994715
0.8	3.89950000000000	3.79968216405646	3.79968216405646	3.79794587586114	3.7993855
0.9	3.99950000000000	3.89954946888650	3.89954946888650	3.89853640620021	3.8993189
1.0		3.99860986942680	3.99860986942680	3.99996079784071	3.9992657

Table 1: Analytic and Neural solution of Example 1.

The error $E(x)=\|y_t(x)-y_a(x)\|$where $y_t(x)$ computed by the following training algorithm		
Trainlm	**Trainbfg**	**Trainbr**
6.43341191782776e-05	6.77710633278927e-07	6.56486495714814e-05
5.53198735442351e-05	1.59832806989613e-07	0.000213591723708273
0.000261518497015612	1.20135476944228e-06	0.000321503100361831
0.00110130535869502	0.000916274984845256	0.000376022317968516
0.00122930168483482	7.29152511702580e-08	0.000377471193488521
0.000969607591307931	5.38443689634960e-09	0.0220747493870004
0.00213983449812005	0.00190942600971189	0.000967861194448805
0.000263003413316199	0.00156884042056982	0.00130106004170516
0.00288916575896980	5.40250320035796e-08	0.00155412413885836
0.00368039653067687	0.000814108450935880	0.000963593799789830
0.00180864374061107	2.08825303715798e-08	0.000460797840708516

Table 2: Accuracy of solutions for Example 1.

Train Function	**Performance of train**	**Epoch**	**Time**	**Msereg.**
Trainlm	1.04e-30	18	0:00:01	1.508187974455366e-07
Trainbfg	3.45e-21	251	0:00:04	6.917729369371148e-07
Trainbr	4.68e-04	1386	0:00:16	4.490158138376238e-05

Table 3: The performance of the train with epoch and time.

Weights and bias for trainlm		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.3313	0.1347	0.5047
0.4316	0.1199	0.8945
0.7179	0.8935	0.3857
0.9162	0.6531	0.2921
0.8900	0.0403	0.2340
Weights and bias for trainbfg		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.1999	0.3585	0.2499
0.1630	0.1343	0.3648
0.0369	0.9986	0.3991
0.2727	0.5135	0.9263
0.2301	0.3878	0.4955
Weights and bias for trainbr		
Net.IW{1,1}	Net.LW{2,1}	Net.B{1}
0.9193	0.3865	0.6633
0.9889	0.6030	0.6023
0.9326	0.5603	0.6565
0.4615	0.8458	0.3099
0.9049	0.2848	0.3316

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

In-put	**Analytic solution**	**Out of suggested FFNN $y_t(x)$ for different training algorithm**			**Numerical Patching method**
x	**$y_a(x)$**	**Trainlm**	**Trainbfg**	**Trainbr**	
0.0	0	5.30985922569390e-16	3.88590164923235e-06	-0.000108152119378016	0.0000000
0.1	0.597837000755620	0.597837000755621	0.592918052256386	0.572073288709426	0.5978370
0.2	0.510825623765991	0.510825623765991	0.510779902455636	0.502977459576147	0.5108256
0.3	0.430782916092454	0.428258098617260	0.430881601043291	0.433818154990611	0.4307829
0.4	0.356674943938732	0.356228190041481	0.356684767145891	0.364595340116877	0.3566749
0.5	0.287682072451781	0.287682072451781	0.287601746123872	0.295308980153126	0.2876821
0.6	0.223143551314210	0.223143551314210	0.223043377389487	0.225959040328804	0.2231435
0.7	0.162518929497775	0.162518929497775	0.162467868943705	0.156545485907880	0.1625189
0.8	0.105360515657826	0.105379554764756	0.105383361925040	0.105425301654285	0.1053605
0.9	0.0512932943875505	0.0512932943875507	0.0513474403606819	0.0358844139613455	0.0512933
1.0	0	-6.99557875567738e-05	-3.48131331875563e-05	-0.0337201923390429	1.0000000

Table 1: Analytic and Neural solution of Example 1.

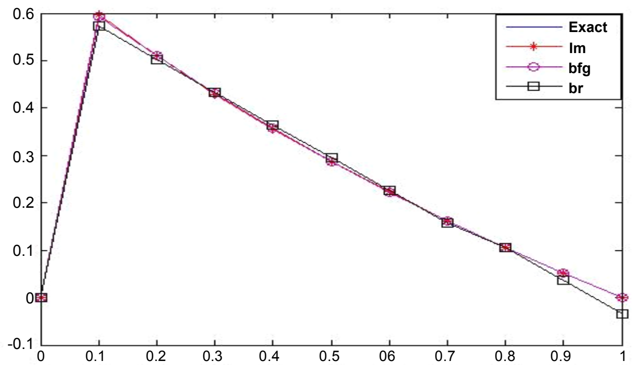

Figure 2: Analytic and neural solution of example 2 using: trainlm, trainbfg and trainbr training algorithm.

The error E(x) thit(x)-y,(x)	1 where yt(x) computed by the follming training algorithm	
Trait	**Trainbfg**	**Trainbr**
5.30985922569390e-16	3.88590164923235e-06	0.0001081)2119373016
5.55111512312578e-16	0.00491894349923447	0.0257637120461945
0	4.57213103546295e-05	0.00784816413984335
0.00252481747519467	9.86849503369248e-05	0.00303523889815655
0.000446753897250962	9.82320715375451e-06	0.00792039617814461
0.00762690770134561	3.03263279092747e-05	2.77555756156239e.16
0.00281548901459333	0.000100173924722308	1.94289029309402e-16
0.00597344358989510	5.10605540695319e-05	4.44039209850063e-16
6.47859964587327e-05	2.28462672137442e-05	1.90391069293111e-05
0.0154038304262050	5.41459731314))9e-05	1.73472347597631e-16
0.0337201923390429	3.48131331375563e-05	6.99557875567738e-05

Table 6: Accuracy of solutions for Example 2.

Msereg.	Time	Epoch	Performance of train	Train Function
5.383267058914925e-07	0:00:03	253	1.66e-32	Trainlm
1.982598144414512e-06	0:00:07	479	3.04e-09	Trainbfg
1.860207937865030e-04	0:00:22	1558	2.30e-13	Trainbr

Table 7: The performance of the train with epoch and time.

Weights and bias for trainhn		
Net.B [11]	Net.LW (2,1)	Net.IW [1,1]
0.6981	0.9831	0.2691
0.6665	0.3015	0.4228
0.1781	0.7011	0.5479
0.1280	0.6663	0.9427
0.9991	0.5391	0.4177
Weights and bias for trainhn		
Net.B[11]	Net.LW(2,1)	Net.IW[1,1]
0.6981	0.9831	0.2691
0.6665	0.3015	0.4228
0.1781	0.7011	0.5479
0.1280	0.6663	0.9427
0.9991	0.5391	0.4177
Weights and bias for trainhn		
Net. B[11]	Net. LW(2,1)	Net. IW[1,1]
0.6981	0.9831	0.2691
0.6665	0.3015	0.4228
0,1781	0.7011	0.5479
0,1280	0.6663	0.9427
0.9991	0.5391	0.4177

Table 8: Initial weight and bias of the network for different training algorithm.

comparison with exact solutions show that the ANN method has good accuracy and efficiency and results obtained using the ANN method is with low error.

References

1. O'Malley RE (1991) Singular Perturbation Methods for Ordinary Differential Euations, Springer Verlag, New York.
2. Doolan EP, Miller JJH, Schilders WHA (1980) Uniform Numerical Methods for Problems with Initial and Boundary Layers, Boole Press, Dublin, Ireland.
3. Roos HG, Stynes M, Tobiska L (1996) Numerical Methods for Singularly Perturbed Differential Equations, Springer, Berlin.
4. Miller JJH, O'Riordan E, Shishkin GI (1996) Fitted Numerical Methods for Singular Perturbation Problems, World Scientific, Singapore.
5. O'Malley RE (1974) Introduction to Singular Perturbations, Academic Press, New York.
6. Jianzhong M (1997) Some Singular Singularly Perturbed Problems, Calgary, Alberta, September.
7. Galushkin AI (2007) Neural Networks Theory, Springer, Berlin, Germany.
8. Ali MH (2012) Design fast feed forward neural networks to solve two point boundary value problems. MSc thesis, University of Baghdad, College of Education for science/Ibn Al-Haitham.
9. Mehrotra K, Mohan CK, and Ranka S (1996) Elements of Artificial Neural Networks, Springer, New York, USA.
10. Ghaffari A, Abdollahi H, Khoshayand MR, Bozchalooi IS, Dadgar A, et al, (2006) Performance comparison of neural network training algorithms in modeling of bimodal drug delivery. International Journal of Pharmaceutics, 327: 126-138.
11. Haykin S (1993) Neural networks: A comprehensive foundation.
12. Padmajaa P, Reddy YN (2013) A Numerical Patching Method for Solving Singular Perturbation Problems Via Padé Approximates. International Journal of Applied Science and Engineering 1: 51-67.