Artificial Neural Network Based Numerical Solution of Ordinary Differential Equations

A THESIS

Submitted in partial fulfillment of the

requirement of the award of the degree of

Master of Science

In

Mathematics

By

Pramod Kumar Parida

Under the supervision of

Prof. S. Chakraverty

May, 2012

DEPARTMENT OF MATHEMATICS

NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA-769008

ODISHA, INDIA

DECLARATION

I declare that the thesis entitled **"Artificial Neural Network Based Numerical Solution of Ordinary Differential Equations"** for the requirement of the award of the degree of Master of Science, submitted in the Department of Mathematics, National Institute of Technology, Rourkela is an authentic record of my own work carried out under the supervision of Dr. S. Chakraverty.

The matter embodied in this thesis has not been submitted by my in any other institution or university for award of any degree.

Date: (PRAMOD KUMAR PARIDA)

NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled **“Artificial Neural Network Based Numerical Solution of Ordinary Differential Equations”** in partial fulfillment of the requirement for the award of the degree of Master of Science, submitted in the Department of Mathematics, National Institute of Technology, Rourkela by **Pramod Kumar Parida** is an authentic record of his own work carried out under my supervision. The content of this has not been submitted for the award of any other degree.

Dr. S. Chakraverty

Professor, Department of Mathematics

National Institute of Technology

Rourkela-769008

Odisha, India

Acknowledgements

I am deeply grateful to all the persons who helped me directly or indirectly with my work.

Special thanks to my supervisor Prof. S. Chakraverty who helped and supported me. Thanks to all the faculties and staff of Department of Mathematics, NIT Rourkela.

Thanks to all the research scholars, my classmates and friends for their support.

Thanks to my family members who always support and encourage me on my journey.

Finally, thanks to God for the kind blessings.

Abstract

In this investigation we introduced the method for solving Ordinary Differential Equations (ODEs) using artificial neural network. The feed forward neural network of the unsupervised type has been used to get the approximation of the given ODEs up to the required accuracy without direct use of the optimization techniques. The problem is formulated in such a manner that it satisfies the initial/boundary conditions by its construction. The trail solution of the ODE is the sum of two terms. The first term satisfies the initial or boundary conditions, while the second one is the feed forward neural output produced by n number of inputs and h number of hidden sigmoid units. The error gradient has been reduced by applying general learning method to get the desired output.

The results have been verified for different problems and the convergence of Artificial Neural Network (ANN) output has been checked for arbitrary points. It may be noted that the interpolation is also possible through this process. The advantage of neuron processor is that the output can be produced to any arbitrary accuracy, while the targets or exact results are unknown or hard to find out.

Table of Contents

	Page
Declaration	(i)
Certificate	(ii)
Acknowledgements	(iii)
Abstract	(iv)
Table of Contents	(v)

Chapters

1. Introduction	1
2. Literature review	3
3. ANN Details	5
4. General method for ODEs using ANN	12
5. Examples and Results	16
6. Discussion	20
7. Conclusion and Future Research	21
References	22

Chapter -1

Introduction

An Artificial Neural network (ANN), usually called "neural network" (NN), is a mathematical model or computational model that simulates the computational model like the biological neural networks. It consists of an interconnected artificial neurons and processes information using a connectionist approach. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning process.

Another aspect of the artificial neural network is that there are different architectures, which requires different types of algorithms, but compare to other complex system, a neural network is relatively simple if handled intelligently.

The advantage of the ANN is the feed forward networking and back propagation of error, by which the network can be trained to minimize the error up to an acceptable accuracy. The training procedure of the network can selected to fit the purpose, for supervised and unsupervised training types.

The benefits of ANN is that the output of the network for selective number of points can be used to find out the outcome for any other new point using the same parameters for interpolation and extrapolation.

Here we present a method for solving the ordinary differential equations which depends on the function approximation capacity of the feed forward neural network and returns the solution of differential equation in a closed analytic and differentiable form.

The feed forward network has the adjustable parameters (weights and biases) that can minimize the error function. To train the network we used unsupervised training technique which requires the computation of the error gradient with respect to the network parameters.

The solution of the differential equation is expressed as the sum of the two terms; the first term satisfies the initial or boundary conditions containing no adjustable parameters. The second term is the feed forward network that is trained to satisfy the differential equation. The neural network method can approximate the solution to an acceptable accuracy. So it is suitable to choose a particular neural model for solving the differential equations.

The training of the network is done in such a way that for each input point to the function, the parameters can be updated in the manner that reduces the error i.e. the network output converges for each given input. Also we can check for interpolation for any new point using the previously trained network parameters.

Chapter -2

Literature Review

ANN is a field which is growing from the last few decades. An enormous amount of literature has been written on the topic of neural networks. Because neural networks are applied to such a wide variety of subjects, it is very difficult to mention here all of available material. A brief history of neural networks has been written to give an understanding of the subject. Papers on various topics related to this study are detailed to establish the need for the proposed work in this study. As such, following paragraphs give a brief literature review for the ANN in general and related to the present problem in particular.

Networks of linear units are the simplest kind of networks, where the basic questions related to learning, generalization, and self-organization can sometimes be answered analytically. Demongeot et al. (2008) presented some relevant theoretical results on the asymptotic behavior of finite neural networks, when they are subjected to fixed boundary conditions. In order to prove that boundaries have no significant impact on one-dimensional neural network, they presented a new general mathematical approach based on the use of a projectivity matrix of the boundary influence in neural networks. Here authors also introduce the numerical tools generalizing the method in order to study phase transitions in more complex cases.[2] Mizutani and Demmel (2003) briefly introduces numerical linear algebra approaches for solving structured nonlinear least squares problems arising from multiple-output neural-network (NN) model. An interesting method proposed by Murao and Kitamura (1997) [3], to evolve adaptive behavior of learning in an artificial neural network (ANN). The adaptive behavior of learning emerges from the coordination of learning rules. Each learning rule is defined as a function of local information of a corresponding neuron only and modifies the connective strength between the neuron and its neighbors.

Chakraverty and his co-authors (2006a), (2007a), (2010), investigated various application of ANN in different practical problems. In particular, the papers include regression based weight generation algorithm in neural network for estimation of frequencies of vibrating plate, neural network based simulation for response identification of two storey shear building subject to earthquake motion and response prediction of single storey building structures subject to earthquake motions. [4] Chakraverty and Gupta (2008) also studied the comparison of neural network configurations in the long range forecast of southwest monsoon rainfall over India. The same author also developed iterative training of neural networks (Chakraverty (2007b) [5]) and identified the structural parameters of two-storey shear building. Prediction of response of structural systems subject to earthquake motions has also been investigated by Chakraverty et al. (2006b) [6] using ANN. For fault classification in structure with incomplete measured data by using auto associative neural networks was studied by Marwala and Chakraverty (2006) [7].

In the finite difference and finite element methods we approximate the solution by using the numerical operators of the function's derivatives and finding the solution at specific preassigned grids. A few works have been done for solving ODE's and PDE's using ANN, which are refereed to produce this paper. Lee and Kang (1990) presented neural algorithms for solving differential equations [9]. Meade Jr and Fernandez (1994) presented the nonlinear differential equations solved by using feed forward neural networks [10], [11]. Lagaris, Likas and Fotiadis (1998) presented the optimization for multidimensional neural network training and simulation [12].

Chapter -3

ANN Details

3.1 The Biological Model [Zurada (1992)]

Artificial neural networks emerged after the introduction of simplified neurons by McCulloch and Pitts in 1943 (McCulloch and Pitts, 1943). These neurons were presented as models of biological neurons and as conceptual components for circuits that could perform computational tasks. The basic model of the neuron is founded upon the functionality of a biological neuron. "Neurons are the basic signaling units of the nervous system" and "each neuron is a discrete cell whose several processes arise from its cell body". Figure 3.1 gives the structure of a neuron in human body.

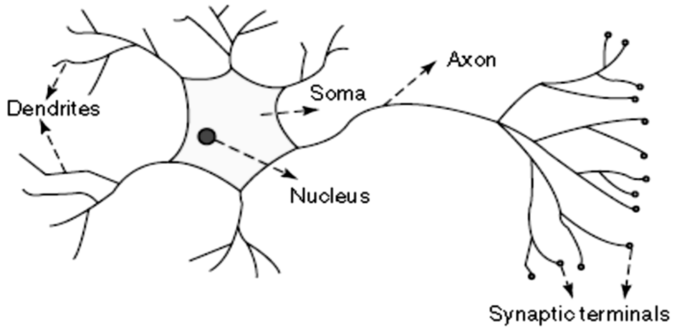

Fig. 3.1 Structure of biological neural system

Human brain has more than 10 billion interconnected neurons. Each neuron is a cell that uses biochemical reactions to receive, process, and transmit information.

The networks of nerve fibers called *dendrites* are connected to the cell body or soma, where nucleus of the cell is located. The body of the cell is a single long fiber called the *axon*, which is branched in to strands and sub strands, are connected to other neurons through the synaptic terminals or synapses.

The basic processing elements of neural networks are called *artificial neurons*, or *simply neurons* or *nodes*. In the mathematical model of neuron, the synaptic effects are represented by connection weights, that modulate the effect of the input signals and the nonlinear characteristic of neurons is represented by a activation function.

The neuron impulse is computed as the weighted sum of the input signals, transformed by the activation function. The learning capability of an artificial neuron can be achieved by adjusting the weights in accordance to the chosen learning algorithm.

3.2 The Mathematical Model [Zurada (1992)]

A functional model of a biological neuron has three basic components of importance. First, the synapses of the neuron which are modeled as weights. The strength of the connection between an input and a neuron is noted by the value of the weight. An activation function controls the amplitude of the output of the neuron. An acceptable range of output is usually between 0 and 1, or -1 and 1.

A typical artificial neuron and the modeling of a multilayered neural network are illustrated in figure 3.2. The signal flow from inputs $x_1, \ldots ,x_n$ are considered to be unidirectional, indicated by arrows to the neuron's output signal flow (O). The neuron output signal O is given as:

$$O = f(net) = f\left(\sum_{j=1}^{n} w_j x_j\right)$$

Where w_j is the weight vector, and the function *f(net)* is an activation function.

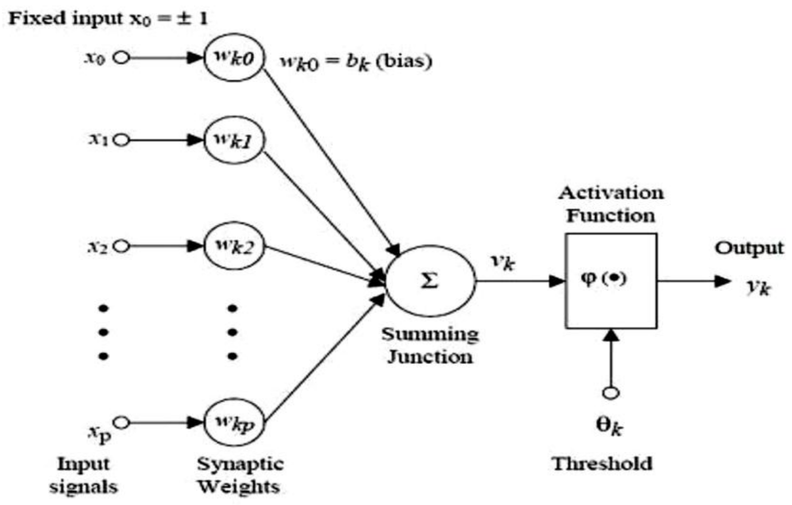

Fig. 3.2 Neural Model

The variable *net* is defined as a scalar product of the weight and input vectors by

$$net = w^T x = w_1 x_1 + \cdots + w_n x_n$$

where T is the transpose of a matrix.

The output value O is computed as

$$O = f(net) = \begin{cases} 1 & if w^T x \geq \theta \\ 0 & otherwise \end{cases}$$

Where θ is called the threshold level; and this type of node is called a *linear threshold unit*. The internal activity of the model for the neurons is given by

$$v_k = \sum_{j=1}^{p} w_{kj} x_j$$

Then the output of the neuron y_k would be the outcome of some activation function on the value of v_k.

3.3 Neural Network Architecture

The architecture contains three neuron layers: input, hidden, and output layers. In feed-forward networks, the signal flows from input to output unit, strictly in a forward direction. The data processing of neurons can be extended over multiple units. The changes of the activation values of the output neurons are significant, that the dynamical behavior constitutes the output of the network. There are several other neural network architectures available, depending on the properties and requirement of the application. A neural network has to be configured such that the implement of a set of inputs must produce the desired set of outputs. One way is to set the weights explicitly, using the previous knowledge. Another way is to train the neural network by feeding it teaching patterns and letting it to change its weights according to some learning rule.

The learning situations in neural networks may be classified into three distinct types. These are supervised, unsupervised, and reinforcement learning.

In supervised learning, an input vector is presented at the inputs together with a set of desired outputs, one for each node, at the output layer. A forward pass is done, and the errors between the desired and actual response for each node in the output layer are computed. These error values are then used to determine weight changes in the net according to the learning rule. The term *supervised* is originated from the fact that the desired signals at each output nodes are provided by an external teacher. The examples of this technique are the back propagation algorithm, the delta rule, and the perceptron rule.

In unsupervised learning, a (output) unit is trained to respond to a set of patterns with in the input. The system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no known set of categories into which the patterns are to be classified; rather, the system must develop its own representation of the inputs.
The examples of this technique are Hebbian, Winner-take-al.

Reinforcement learning is the type of learning, where what to do – how to map situations to actions –so as to maximize a numerical reward signal. The learner is not directed which actions to perform, as in most forms of learning, but instead must generate which actions yield the most reward by trying them. In cases the most interesting and challenging factor is that actions may affect not only the immediate reward, but also the next situation and through that all subsequent rewards. The trial and error search and delayed in the reward are the two most important distinguishing features of reinforcement learning.

3.4 Feed Forward Network

The data goes from input to output units in strictly feed-forward direction. The data processing can extend over multiple units, but no feedback connections are present, that is, there is no connections extending from outputs of units to inputs of units in the same layer or previous layers. This is shown in figure 3.4 [Zurada (1992)] for single-layer feed-forward network along with the block diagram.

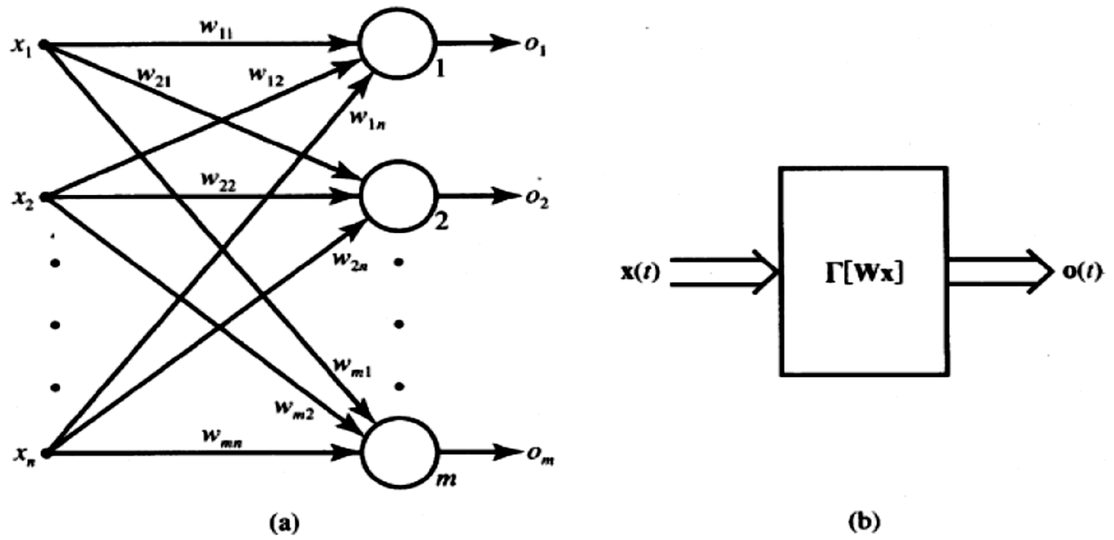

Fig. 3.4 Single-layer discrete-time feed-forward network: (a) interconnection scheme and (b) block diagram.

3.5 Back Propagation Learning

The simple perceptron can handle linearly separable or linearly independent problems. Taking the partial derivative of the error of the network with respect to each of its weights, we can know the flow of error direction in the network. If we take the negative derivative and then proceed to add it to the weights, the error will decrease until it approaches local minima. If the derivative is positive, the error is increasing when the weight is increasing. Then we have to add a negative value to the weight or the reverse if the derivative is negative. Because of these partial derivatives and then applying them to each of the weights, starting from the output layer to hidden layer weights, then the hidden layer to input layer weights, this algorithm is called the *back propagation algorithm*.

A neural network can be trained in two different ways: online and batch modes. The weight updates of the two methods for the same data presentations are different.

The online method weight updates are computed for each input data sample, and the weights are modified after each sample. The other way is to get the weight updates for each input sample is to store values during one pass through the training set which is called an *epoch*. At the end of the epoch, all the values are added, and the weights will be updated with the calculated values. This method generates the weights with a cumulative weight update, so it will follow the gradient more closely. It is called the *batch-training mode*.

The training of the network involves feeding samples as input vectors, calculation of the error of the output layer, and then adjusting the weights of the network to minimize the error.

The average of all the squared errors (E) for the outputs is computed to make the derivative simpler. After the error is computed, the weights can be updated one by one. In the batched mode the descent depends on the gradient ∇E for the training of the network [Zurada (1992)]

$$\Delta w_{ij}\,(n) = -\eta * \frac{\delta E}{\delta w_{ij}} + \alpha * \Delta w_{ij}\,(n-1)$$

Where η and α are the learning rate and momentum respectively. The momentum term returns the effect of previous weight changes on the current direction of movement in the weight space.

It has been proven that back propagation learning with sufficient hidden layers can approximate any nonlinear function to arbitrary accuracy. This makes back propagation learning the right model for signal prediction and system modeling.

Chapter 4

General method for ODEs using ANN

4.1 Description of the Method

This paper gives a method to solve the ordinary differential equations. We introduce a trial function which can be used to train input data at arbitrary nodes. This trial function is based on two facts. First, satisfies the boundary conditions (BC's) of the differential equation. Second, is the sum of two terms, involving perceptron parameters. This technique is not only applicable to ordinary differential equations, but also can used to solve partial differential equations, that can be considered in the future works.

Consider the general differential equation [Lagaris et al. (1998)]:

$$G\big(x, \Psi(x), D\Psi(x), \dots, D^m\Psi(x)\big) = 0, \qquad x \epsilon D \tag{1}$$

Subject to some boundary conditions, where $x = (x_1, x_2, \dots, x_n) \,\epsilon\, R^n$, $D \subset R^n$ denotes the domain and $\Psi(x)$ is the solution to be computed.

If $\Psi_t(x,p)$ is a trail solution with parameters p, the problem formulated to [Lagaris et al. (1998)]

$$\min_p \sum_{x_i \in D} \Big(G\big(x_i, \Psi_t(x_i,p), D\Psi_t(x_i,p), \dots, D^m\Psi_t(x_i,p)\big)\Big)^2 \tag{2}$$

The trail solution of Ψ_t employs a feed forward neural network with the parameters p related to the weights and biases of the neural architecture. The trail function $\Psi_t(x)$ is formulated to satisfy the boundary conditions. The trail solution can be written as the sum of two terms

$$\Psi_t(x) = A(x) + F(x, N(x,p)) \tag{3}$$

Where $N(x,p)$ is the single output of feed forward neural network with the parameters p and n input units with the input x.

The term $A(x)$ contains no adjustable parameters which satisfies the boundary conditions. The second term F makes no contribution to BC's but this employs a neural network whose weights and biases are adjusted to minimize the error function.

4.2 Computation for Gradient [Lagaris et al. (1998)]

We need to minimize (2) by the neural network training. The error corresponding to each input x_i is the value $G(x_i)$ which has to be zero. The error computation not only involves the outputs but also the derivatives of the network output to the respective inputs. So it requires finding out the gradient of the network derivatives to that of its inputs.

Now consider a multilayered perceptron with n inputs, a hidden layer with h units and one output. For the given inputs $x = (x_1, x_2, \dots, x_n)$ the output is given by

$$N(x) = \sum_{i=1}^{h} v_i \sigma(z_i) \tag{4}$$

Where $z_k = \sum_{j=1}^{n} w_{jk} x_j + u_k$, w_{jk} denotes the weight from input to hidden unit, v_i denotes weight from hidden to output unit, u_k denotes the biases and $\sigma(z_i)$ is the sigmoid activation function.

The derivatives of N with respect to input x_i is

$$\frac{\partial^k N}{\partial x_j^k} = \sum_{i=1}^{h} v_i w_{ij}^k \sigma_i^{(k)} \tag{5}$$

Where σ^k denotes the k-th order derivative of sigmoid.

Now N_g is the derivative of the network to any of its inputs. Let $P_i = \prod_{k=1,2,\dots,n} w_{ij}^k$ then we have the relation $N_g = D^n N = \sum_{i=1}^{n} v_i P_i \, \sigma_i^{(\Lambda)}$ (6)

The derivative of N_g with respect to other parameters is given as:

$$\frac{\partial N_g}{\partial v_i} = P_i \sigma_i^{(\Lambda)}, \frac{\partial N_g}{\partial u_i} = v_i P_i \sigma_i^{(\Lambda+1)} \text{ and } \frac{\partial N_g}{\partial w_{ij}} = x_j v_i P_i \sigma_i^{(n+1)} + v_i \Lambda_j w_{ij}^{\Lambda_i - 1}. \quad (7)$$

Now after getting all the derivatives we can find out the gradient of error. Using general learning method for unsupervised training we can minimize the error to the desired accuracy.

4.3 Method for solving ODEs

Take the first order ODE: $\frac{d\Psi(x)}{dx} = f(x, \Psi)$ with $x \in [0,1]$ and initial condition $\Psi(0) = A$.

The trail solution is given by $\Psi_t(x) = A + xN(x,p)$ [Lagaris et al. (1998)] where $N(x,p)$ is the neural output of the feed forward network with input x and weights p.

The error to be minimized is given by

$E(p) = \sum_i \left\{\frac{d\Psi_t(x_i)}{dx} - f\big(x_i, \Psi_t(x_i)\big)\right\}^2$, Where $x_i \in [0,1]$.

So we can compute the gradient of error with respect to the parameter p using (4) to (7).

For the second order ODE $\frac{d^2\Psi(x)}{dx^2} = f(x, \Psi, \frac{d\Psi}{dx})$. The initial value is $\Psi(0) = A$ and $\frac{d}{dx}\,\Psi(0) = A'$.

The trail solution is $\Psi_t(x) = A + A'x + x^2 N(x,p)$ and the error function is given by

$E(p) = \sum_i \left\{\frac{d^2\Psi_t(x_i)}{dx^2} - f\left(x_i, \Psi_t(x_i), \frac{d\Psi_t(x_i)}{dx}\right)\right\}^2.$

This can be extended to k-th order ODE $\frac{d^k\Psi(x)}{dx^k} = f(x, \Psi_1, \Psi_2, \dots, \Psi_k)$ with $\Psi_i(0) = A_i$ and consider the trail solution $\Psi_{t_i}(x) = A_i + xN_i(x, p_i)$.

The error function to minimize is given by

$$E(p) = \sum_{k=1}^{K} \sum_i \left\{\frac{d\Psi_{t_k}(x_i)}{dx^k} - f_k\left(x_i, \Psi_{t_1}, \dots, \Psi_{t_k}\right)\right\}^2.$$

Now the activation function is $\sigma = \frac{1}{1+e^{-x}}$ and the derivatives of the sigmoid are

$\sigma' = \sigma(1 - \sigma)$,

$\sigma'' = 2\sigma^3 - 3\sigma^2 + \sigma$,

$\sigma''' = -6\sigma^4 + 12\sigma^3 - 7\sigma^2 + \sigma$ and so on.

The network is trained using unsupervised method in general way without applying any optimization technique. For every input the parameters are updated and the sum of the errors for each input was minimized by feed forward neural network training.

Chapter 5

Examples and Results

1. Problem-1:

Consider the following first order ODE:

$\frac{d\Psi(x)}{dx} = 4x^3 - 3x^2 + 2$ Subject to $\Psi(0) = 0$ and $x \in [0,1]$.

The exact solution of the ODE is: $\Psi(x) = x^4 - x^3 + 2x$.

The trail solution is given as $\Psi_t(x) = xN(x,p)$.

The network is trained using 10 equidistance points in [0,1]and with 5 sigmoid hidden units.

The following Table 5.1 shows the error minimization and the convergence of the neural output up to the given accuracy. The weights are first selected randomly.

Table 5.1: Convergence of the neural output for different error

Testing Points (x)	Exact Values	Error							
		0.5	0.1	0.05	0.01	0.005	0.001	0.0005	0.0001
		Neural Output							
0	0	0	0	0	0	0	0	0	0
0.1111	0.2210	0.2045	0.2142	0.2178	0.2205	0.2207	0.2209	0.2209	0.2209
0.2222	0.4359	0.4422	0.4411	0.4407	0.4400	0.4397	0.4391	0.4384	0.4380
0.3333	0.6420	0.7137	0.6831	0.6720	0.6632	0.6609	0.6571	0.6536	0.6526
0.4444	0.8410	0.9823	0.9242	0.9026	0.8879	0.8835	0.8758	0.8687	0.8675
0.5555	1.0349	1.1988	1.1407	1.1184	1.1086	1.1042	1.0952	1.0867	1.0861
0.6666	1.2346	1.3354	1.3228	1.3164	1.3259	1.3247	1.3190	1.3131	1.3127
0.7777	1.4510	1.4175	1.4920	1.5168	1.5531	1.5560	1.5558	1.5538	1.5518
0.8888	1.6997	1.5251	1.6987	1.7542	1.8109	1.8142	1.8140	1.8101	1.8062
0.9999	2.0000	1.7576	1.9887	2.0708	2.1062	2.1025	2.0918	2.0773	2.0758

The following Fig. 5.2 shows the difference in exact result and the neural output with the accuracy of 0.0001. The error can be minimized more but it requires more time to converge.

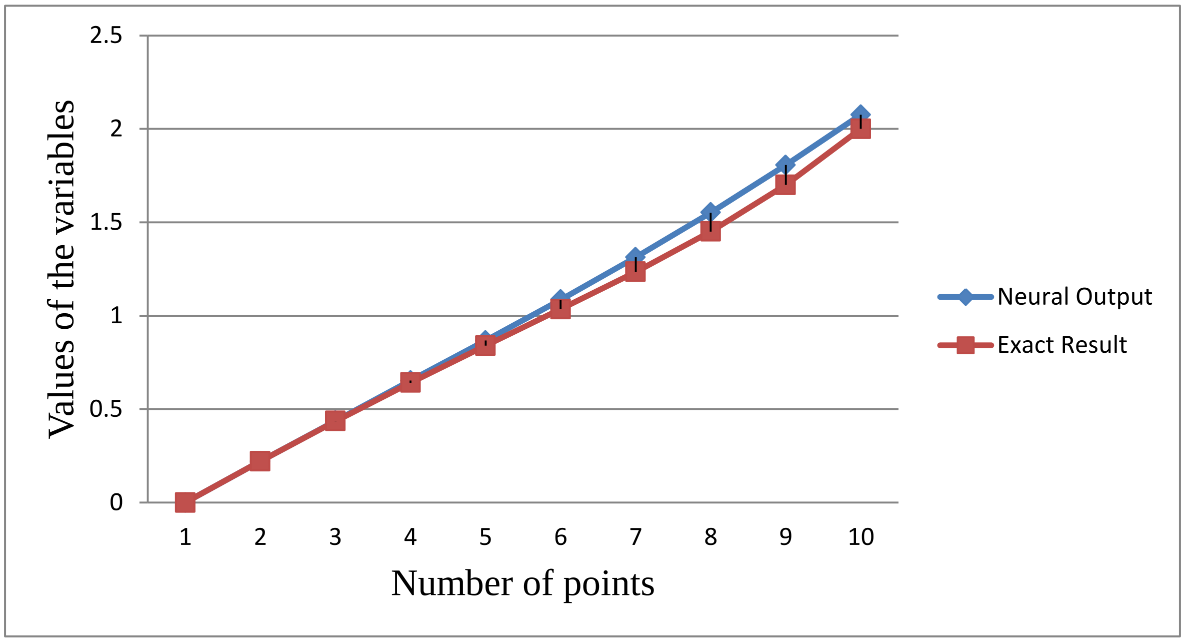

Fig: 5.2 Comparison between exact results to the approximated result

The interpolation with the accuracy of 0.0001 is shown in the following Table 5.3 at the testing points as shown.

Table 5.3 Neural interpolation

Testing points	0.5308	0.7792	0.2630	0.1524	0.9340	0.4694
Exact Results	0.9914	1.4539	0.5126	0.3018	1.8142	0.8839
ANN Results	1.0373	1.5347	0.5169	0.3020	1.9131	0.9163

2. Problem-2

Take another first order ordinary differential equation:

$$\frac{d\Psi(x)}{dx} = 2x + 1 \qquad \text{at} \qquad \Psi(0) = 0 \text{ and } x \in [0,1].$$

The exact solution is: $\Psi(x) = x^2 + x$ and

The trail solution can be written as $\Psi_t(x) = xN(x, p)$.

The network is trained using 6 equidistance points in [0,1] and with 5 sigmoid hidden units. The following Table 5.4 shows the error minimization and the convergence of the neural output up to successive accuracy.

Table 5.4: Convergence of the neural output for different error

Testing Points (x)	Exact results	Error							
		0.5	0.1	0.05	0.01	0.005	0.001	0.0005	0.0001
		Neural Output							
0	0	0	0	0	0	0	0	0	0
0.2	0.2400	0.2098	0.2250	0.2283	0.2312	0.2381	0.2401	0.2407	0.2418
0.4	0.5600	0.4856	0.4778	0.4836	0.4971	0.5395	0.5410	0.5487	0.5503
0.6	0.9600	0.9818	0.7889	0.7952	0.8102	0.8672	0.9135	0.9418	0.9562
0.8	1.4400	1.7915	1.1390	1.1846	1.2308	1.2700	1.3341	1.3722	1.4092
1.0	2.0000	2.8339	1.4397	1.5401	1.6700	1.7431	1.8019	1.8157	1.8601

The following Fig. 5.5 shows the difference in exact result and the neural output with the accuracy of 0.0001.

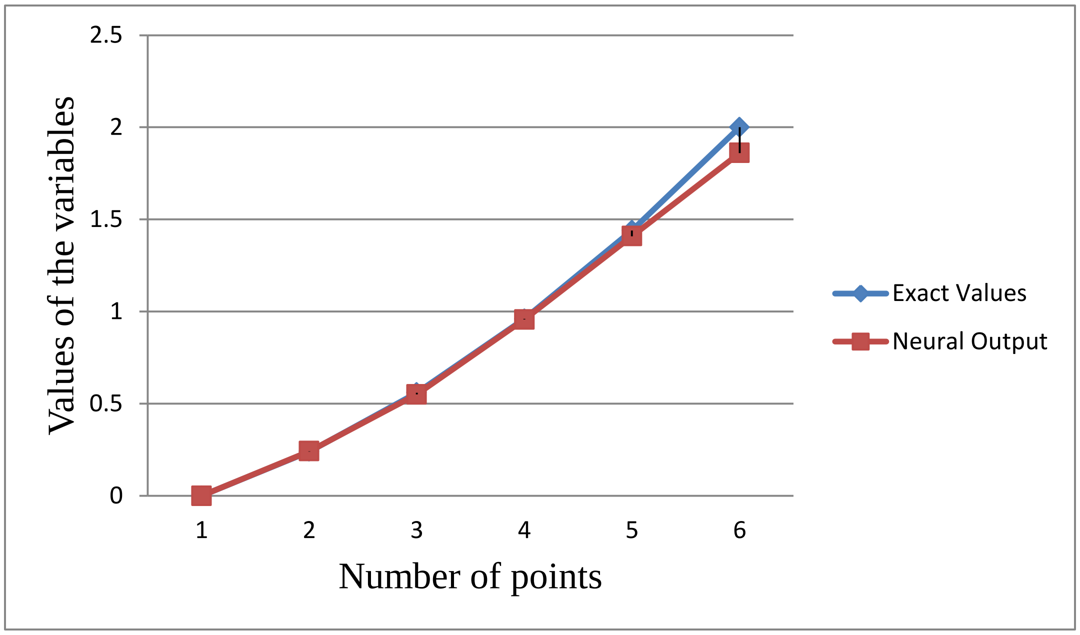

Fig. 5.5 Comparison between exact results to the approximated result

The interpolation with the accuracy of 0.0001 is shown in the following Table 5.6 at the selected testing points.

Table 5.6 Neural interpolation

Testing points	0.8235	0.6787	0.1712	0.3922	0.0318	0.9502
Exact Results	1.5017	1.1393	0.2005	0.5460	0.0328	1.8531
ANN Results	1.6035	1.3385	0.2388	0.5701	0.0336	1.9526

Chapter 6

Discussion

The unsupervised training method is useful for formulating the differential equation when the target is unknown. It has been observed that neural output varies for the number of inputs that are to be trained as well as the number of hidden nodes. The learning constant used has the effect for the convergence of gradient error. Small values of the learning constant make the slow convergence of the error and the large values make converge rapid. But the error approximation differs for both the cases.

The updated parameters of the neuron process works for each new arbitrary point from the domain. The approximation is more effective when the selection of the parameters is made such that the error at first epoch is small. By this type of selection the time for error convergence can be reduced to some extent. The error accuracy is not same for each testing point .i.e. each testing point cannot converge with the same accuracy. It was observed that at some cases the convergence of neuron output get stopped while the accuracy was increased further and further.

Chapter 7

Conclusion and Future Research

The function approximation capability of neural network provides accurate and differentiable solutions in a closed analytic form. It gives an excellent function approximation and satisfies the initial/boundary conditions. The method provides excellent generalization as the derivation at test points never deviates more to that of exact one.

The neural network architecture has been considered as fixed for all the simulation experiments. Certainly the result may be good if we take more number of hidden nodes. As such it will be a challenge to see how far the result becomes better by taking different number of hidden layers and nodes.

The other factor is the variation of test points with respect to the error minimization, because we only considered equidistance points. So better result could be expected by taking more number of test points where the error values are more.

It was observed that the parameters of ANN remains fixed while the dimension can be increased by taking more test points. But in this process the time required for the ANN training will be more for getting better result.

References

[1] Demongeot J., Jézéquel C. and Sylvain Sené S. (2008), Boundary Conditions and Phase transitions in Neural Networks. Theoretical results, Neural Networks, 21, 971–979.

[2] Mizutani E. and Demmel J.W. (2003), On Structure-Exploiting Trust-Region Regularized Nonlinear Least Squares Algorithms for Neural-Network Learning, Neural Networks, 16, 745–753.

[3] Murao H.and Kitamura S. (1997), Evolution of Locally Defined Learning Rules and their Coordination in Feedforward Neural Networks, Artif Life Robotics , 1, 89-94.

[4] Chakraverty S. and Gupta P. (2008), Comparison of Neural Network Configurations in the Long Range Forecast of Southwest Monsoon Rainfall Over India, J.neural Computing and Applications, U.K., 17, 2, 187-192.

[5] Chakraverty S. (2007b), Identification of Structural Parameters of Two-storey Shear Building by the Iterative Training of Neural Networks, Architectural Science Review Australia, 50, 4, 380-384.

[6] Chakraverty S., Marwala T. and Gupta P. (2006b), Response Prediction of Structural System Subject to Earthquake Motions Using Artificial Neural Network, Asian Journal of Civil Engineering, 7, 3, 301-308.

[7] Marwala T. and Chakraverty S. (2006), Fault Classification in Structures with Incomplete Measured Data Using Autoassosiative Neural Networks and Genetic Algorithm, Current Science, 90, 4, 542-548.

[8] Lambert J.D. (1983), Computational Methods in Ordinary Differential Equations, John Wiley and Sons, New York,.

[9] Lee H., Kang I.S. (1990), Neural algorithms for solving differential equations, Journal of Computational Physics 91 110–131.

[10] Meade Jr A.J., Fernandez A.A. (1994), The numerical solution of linear ordinary differential equations by feed forward neural networks, Mathematical and Computer Modelling 19 (12) 1–25.

[11] Meade Jr A.J., Fernandez A.A. (1994), Solution of nonlinear ordinary differential equations by feed forward neural networks, Mathematical and Computer Modelling 20 (9) 19–44.

[12] Lagaris I.E., Likas A., Fotiadis D.I. (1998), Artificial neural networks for solving ordinary and partial differential equations, IEEE Transactions on Neural Networks 9 (5) 987–1000.

[13] Likas A., Karras D.A., Lagaris I.E. (1998), Neural-network training and simulation using a multidimensional optimization system, International Journal of Computer Mathematics 67 33–46.

[14] Malek A., Beidokhti R. Shekari (2006), Numerical solution for high order differential equations, using a hybrid neural network—Optimization method, Applied Mathematics and Computation 183,260–271.

[15] Zurada J. M. (1992), Introduction to artificial neural systems, West Publishing Company, St. Paul, United States of America.

[16] Abraham A. (2004) Meta-Learning Evolutionary Artificial Neural Networks, Neuro computing Journal, Vol. 56c, Elsevier Science, Netherlands, (1–38).