GUI System for Function Approximation using Neural Networks

Tarun Varshney, Student Member, IEEE  
Department of EE&I Engineering,  
Moradabad Institute of Technology, Moradabad  
Uttar Pradesh (India)  
email: t_varshney@yahoo.com

Satya Sheel, Senior Member, IEEE  
Department of Electrical Engineering,  
MNNIT, Allahabad, UP, INDIA  
e-mail: satyasheel@mnnit.ac.in

Abstract--This paper focuses the function approximation capability of feed forward neural network (FFNN). A Graphical user Interface (GUI) system has been developed and tested for function approximation. This GUI system can approximate any nonlinear/linear function which can have any number of input variable and six output variables. Configuration of neural network can be set from a single GUI window. A FFNN with a single hidden layer has been used to approximate the functions. To train the neural network various variant of Back propagation training algorithms has been configured in GUI system. Although this GUI system has been tested on various function but simulation results of two cases has been reported. Finally comparison has been made which shows that Levenberg-Marquardt (LM) back propagation with single hidden layer FFNN converges faster than other training algorithms.

Keywords—Function approximation, GUI, Levenberg-Marquardt Algorithm, Feed forward neural network, MISO.

I. INTRODUCTION

Function approximation is a significant task in numerous engineering applications ranging from dynamic system identification to control, fault detection, optimization and classification. It refers to curve fitting equations (i.e., function approximators) to observed data. The term function approximator is used to represent multivariable approximation models, mostly nonlinear, which possess a number of adjustable parameters known as weights. Common approximation methods include least squares fit, neural networks in the form of feed forward multilayer perceptrons (MLPs), radial basis function (RBF) networks [1] etc. Neural networks are a convenient means of representation because they are universal approximators that can learn data by example or reinforcement [2-4], either in batch or sequential mode. Typically, training involves the numerical optimization of the error between the data and the actual network’s performance with respect to its adjustable parameters or weights. Considerable effort has gone into developing techniques for accelerating the convergence of these optimization-based training algorithms [5-12]. An alternative to the multilayer perceptron is the Radial Basis Function (RBF) network. [13-14] proved theoretically that the RBF type neural networks are capable of universal approximations and learning without local minima, thereby guaranteeing convergence to globally optimum parameters. Generally, the traditional radial basis function neural network may sometimes be considered prohibitively expensive to implement in computational terms for large amounts of training data [13-14]. The performance of a trained RBF network depends on the number and locations of the radial basis functions, their shape and the method used for learning the input-output mapping. The most popular existing learning strategies for RBF neural networks can be classified as follows: (i) strategies selecting the RBF centers randomly from the training data, (ii) strategies employing unsupervised procedures for selecting the RBF centers, and (iii) strategies employing supervised procedures for selecting the RBF centers. Two difficulties are involved with traditional RBF networks: the initial configuration of an RBF network needs to be determined by a trial-and-error method, and the performance suffers degradation when the desired locations of the center of the RBF are not suitable. In this paper a GUI system for function approximation using neural network with single hidden layer has been modeled where user can configure the various parameters of neural network. This paper is organize as follows- Section II describes the various features of GUI system, the basics of LM algorithm are illustrated in section III, results of simulation and comparisons has been shown in section IV and finally concluding remarks are given in section V.

II. DEVELOPMENT OF GUI SYSTEM

To simplify the simulation procedure a GUI system has been developed and shown in Fig. 1. The various simulation parameters can be set from a single panel. User may choose any number of hidden neurons for hidden layer. Network
training can be stopped by examining either maximum number of iteration/epochs or permissible error whichever is met first. One can choose learning rate and momentum rate between 0 to 1. This GUI system handles any numbers of input variable and maximum six (06) output variables. The efficiency of neural network training improves by normalizing the inputs and outputs data. Normalization of input & output data pair can be done between -1 to 1 as well as for 0 and standard mean of 1. Input and output pairs has to be store in an excel file and can be fetch by typing the name of file in the field of filename.xls. Activation function for hidden layer and output layer can be chosen e.g sigmoid, log sigmoid, linear. To train the neural network a variant of backpropagation algorithm can be opt from following list

- Gradient descent backpropagation
- Gradient descent with momentum backpropagation
- Gradient descent with adaptive learning rate backpropagation
- Gradient descent with momentum and adaptive learning rate backpropagation
- Levenberg-Marquardt backpropagation
- BFGS Quasi-Newton backpropagation
- RPROP backpropagation
- Bayesian Regulation backpropagation
- Conjugate gradient backpropagation with Polak-Ribiere updates
- Conjugate gradient backpropagation with Fletcher-Reeves updates
- Conjugate gradient backpropagation with Powell-Beale restarts
- Scaled conjugate gradient backpropagation
- Conjugate gradient backpropagation with Powell-Beale restarts
- One step secant backpropagation.

Although this GUI system has been tested for various nonlinear functions but in this paper only the results of two nonlinear functions has been reported. It has been found that LM algorithm converges faster than other training algorithms, a brief overview of LM algorithm is described in next section.

III. LM TRAINING ALGORITHM

LM algorithm is the blend of steepest descent and the Gauss-Newton searching methods and illustrated in [15-16]. It consists from the following equation:

\[ (J^T J + \lambda I) \delta = J^T E \]  

(1)

Where J is the Jacobian matrix for the system, \( \lambda \) is the Levenberg's damping factor, \( \delta \) is the weight update vector and E is the error vector containing the output errors for each input vector used on training the network. The \( \lambda \) damping factor is adjusted at each iteration, and guides the minimization process. This adjustment for \( \lambda \) is done by using an adjustment factor v, usually defined as 10. If \( \lambda \) needs to increase, it is multiplied by \( v \). If it needs to decrease, then it is divided by \( v \). The process is repeated until the error decreases. When this happens, the current iteration ends.

The following steps are involved in LM algorithm;
1. Compute the Jacobian of the system
2. Compute the error gradient \( g = J^T E \)
3. Solve \((H + \lambda I) \delta = g \) to find \( \delta \)
4. Update the network weights \( w \) using \( \delta \)
5. Recalculate the sum of squared errors using the updated weights
6. If the sum of squared errors has not decreased, Discard the new weights, increase \( \lambda \) using \( v \) and go to step 3.
7. Else decrease \( \lambda \) using \( v \) and stop.

IV. SIMULATION RESULTS AND DISCUSSION

Single hidden layer neural network has been considered because it has been well proven that single layer neural networks are universal approximators. Two benchmark problems are tested with this GUI system. For both cases 1000 pairs of input and output has been generated. This set of data has been divided into training, testing and validation data sets. 60 % data has been used to train the neural network and 20 % data has been used for testing and rest data set has been used for validation purpose.

Once input output pairs are generated, it must be preprocessed otherwise, the neural network will not produce correct output. For preprocessing normalization has been done between -1 to +1. Neural network has been trained for 0.001 performance goal. In each problem the neural network is trained from any arbitrarily random weight initialization.

A. Two dimensional function-I

The 2-D Gabor function is given by following equation

\[ g(x_1, x_2) = \frac{1}{2\pi(0.5)^2} e^{-(x_1^2 + x_2^2)^2/(2(0.5)^2)} \cos(2\pi(x_1 + x_2)) \]  

(2)

where x1 and x2 are input variables as in [17]. For this case 1000 data are generated in the range of [-0.5 0.5]. The neural network has been trained with LM algorithm. Learning rate and momentum coefficient has been chosen 0.5 and 0.9 respectively. Simulation has been carried out with the consideration of 15 hidden neurons in hidden layer. By the observation Fig. 2 it can be seen that feed forward neural network with single hidden layer can approximate the 2-D Gabor function which is highly nonlinear. The data set is divided into three distinct sets called training, testing and validation sets. Training set is largest set and used by neural network to learn patterns presents in data. The testing set is used to evaluate the generalization ability of a trained network. A final check on a performance of the trained network is made using validation set. The comparison of square error for all three sets with respect to epochs has been
shown in Fig. 3 and it has been noticed that only in 199 epochs neural network is able to get performance goal. The RBF neural network has been used to approximate the Gabor function [17]. The simulation results of present work have been compared in Table 1. From the observation it is clear that Feed forward neural network with single hidden layer can better approximate the 2-D Gabor function.

<table>
<thead>
<tr>
<th>SN</th>
<th>Algo</th>
<th>Type of Network</th>
<th>No. of centers</th>
<th>No. of hidden neurons</th>
<th>rms error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BP</td>
<td>RBF</td>
<td>40</td>
<td>---</td>
<td>0.0847241</td>
</tr>
<tr>
<td>2</td>
<td>BP</td>
<td>RBF</td>
<td>80</td>
<td>---</td>
<td>0.0314169</td>
</tr>
<tr>
<td>3</td>
<td>LF-I</td>
<td>RBF</td>
<td>40</td>
<td>---</td>
<td>0.0192033</td>
</tr>
<tr>
<td>4</td>
<td>LF-II</td>
<td>RBF</td>
<td>40</td>
<td>---</td>
<td>0.0186757</td>
</tr>
<tr>
<td>5</td>
<td>LM</td>
<td>FFNN</td>
<td>---</td>
<td>15</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Fig. 1. Overlook of GUI system

TABLE 1
PERFORMANCE COMPARISON FOR GABOR FUNCTION

B. Two dimensional function-II

In this case a nonlinear function is approximated which is given as

\[ Z(x, y) = 2 \sin(\pi e^{-x^2-y^2}) \]  

where \( x \) and \( y \) are input variables as in [18]. Simulation has been carried out with BP with momentum constant and LM training algorithm. Learning rate and momentum coefficient has been chosen 0.3 and 0.9 respectively. Fig. 4 shows that neural network with the architecture of 2-15-1 can approximate the function which is given by Eq. 3. From the comparison graph Fig. 5 it is observed that only in 10 epochs neural network is able to get performance goal of 0.001.

Table 2 concludes that the LM algorithm can approximate the non linear function much faster than the BP with momentum constant. It takes 98.07% lesser epochs to approximate the function with the performance goal of 0.001. It is worth to mention that rather than 40 neurons only 15 neurons are sufficient to approximate the function with LM algorithm. With the lesser number of hidden neurons computation time can be saved.
VI. REFERENCES


V. CONCLUSION

This paper presents GUI system to approximate the functions using neural network. This GUI provides computing without the deepest theoretical knowledge which is the main advantage of this system. User can change configuration, and the most of the important parameters of the neural network. This contribution shows the function approximation capability of feed forward neural network with single hidden layer. Two 2-D highly non linear functions are tested where LM algorithm has been chosen for training. Simulation results confirmed that FFNN with single hidden layer having 15 neurons have better function approximation capability than the radial basis function and LM algorithm converges faster than back propagation algorithm with momentum constant.