Pakistan Journal of Applied Sciences 2(3): 288-294, 2002

Comparison of Neural Algorithms for Function Approximation

[1]Lale Özyilmaz, [1]Tülay Yildirim and [2]Kevser Köklü
[1]Electronics and Communication Engineering Department Yildiz Technical University, Turkey
[2]Mathematical Engineering Department Yildiz Technical University, Turkey

Abstract: In this work, various neural network algorithms have been compared for function approximation problems. Multilayer Perceptron (MLP) structure with standard back propagation, MLP with fast back propagation (adaptive learning and momentum term added), MLP with Levenberg-Marquardt learning algorithms, Radial Basis Function (RBF) network structure trained by OLS algorithm, and Conic Section Function Neural Network (CSFNN) with adaptive learning have been investigated for various functions. Results showed that the neural algorithms can be used for functional estimation as an alternative to classical methods.

Key Words: Function Approximation, Multilayer Perceptron, Radial Basis Function, Conic Section Function Neural Network, Adaptive Learning

Introduction

Functional estimation is an important problem in data analysis and pattern recognition problems. Studies on artificial neural networks as function approximators have been used in various areas. For example, the purpose in control systems is to find a suitable feedback function providing a relation from measured output to control inputs. Similarly, purpose in adaptive filtering is to obtain a function providing a relation from delayed values of an input signal to a suitable output signal. In the literature, Multilayer Perceptron and Radial Basis Function Network structures are used for function approximation. (Hagan *et al.*, 1996)

Multilayer Perceptron: Multilayer Perceptron (MLP) is the most common neural network model, consisting of successive linear transformations followed by processing with non-linear activation functions. MLP represents a generalisation of the single layer perceptron, which is only capable to construct linear decision boundaries and simple logic functions. However, by cascading perceptrons in layers complex decision boundaries and arbitrary Boolean expressions can be implemented. MLP is also capable to implement non-linear transformations for function approximations. (Geva and Sitte, 1992; Haykin, 1994; Hush and Horne, 1993 and Lippmann, 1987)

The network consists of a set of sensory units (source nodes) that constitute the input layer, one or more hidden layers of computation nodes, and an output layer. Each layer computes the activation function of a weighted sum of the layer's inputs. The input signal propagates through the network in a forward direction, on a layer-by-layer basis. The learning algorithm for multilayer perceptrons can be expressed using generalised Delta Rule and gradient descent since they have non-linear activation functions. (Jondarr, 1996; Riedmiller and Braun, 1997; Rumelhart *et al.*, 1987 and Zurada, 1995)

In general form of an MLP network, the x_i inputs are fed into the first layer of $x_{h,1}$ hidden units. The input units are simply 'fan-out' units: no processing takes place in these units. The activation of a hidden unit (neuron j) is a function f_j of the weighted inputs plus a bias, as given in Eq.1.

$$x_{pj} = f_j\left(\sum w_{ji} x_{pi} + \theta_j\right) = f_j(y_{pj}) \quad (1)$$

where w_{ji} is the weight of input i to neuron j, x_{pi} is input i, that is, output i from the previous layer, for input pattern p and θ_j is the threshold value. The output of the hidden units is distributed over the next layer of $x_{h,2}$ hidden units until the last layer of hidden units, of which the outputs are fed into a layer of x_o output units. (Krose *et al.*, 1993 and Vysniauskas *et al.*, 1993)

Back Propagation Training Algorithm (BP): Back-propagation algorithm is perhaps the most widely used training procedure for feedforward neural networks. It consists of an error function representing a measure of the performance of the network. The algorithm cycles through the training data are initialisation, presentation of training examples, forward computation, backward computation and iteration. (Hagan *et al.*, 1996; Jondarr, 1996 and Rumelhart *et al.*, 1987)

Fast Back Propagation Training Algorithm (FBP)

Learning Rate and Momentum: The learning procedure using back-propagation algorithm requires the change in weight made by gradient descent rule. This can be very slow if learning rate γ is small. Practically, γ is chosen as large as possible without leading to oscillation. One way to avoid oscillation at large γ is to make the change in weight dependent of the past weight change by adding a momentum term, α, as a constant which determines the effect of the previous weight change:

$$\Delta w_{ij}(t+1) = \gamma \delta_i^p a_j^p + \alpha \Delta w_{ij}(t) \quad (2)$$

where t indexes the presentation number.

When no momentum term is used, learning takes a long time before the minimum has been reached with a low learning rate, whereas for high learning rates the minimum is never reached because of the oscillations. When adding the momentum term, the minimum will be reached faster. (Hagan *et al.*, 1996; Krose and Smagt, 1993; Riedmiller and Braun, 1997 and Zurada, 1995)

Radial Basis Functions (RBF): The Radial Basis Function approach to approximating functions consists of modelling an input-output mapping as a linear combination of radially symmetric functions (Botros and Atkeson, 1991; Broomhead and Lowe, 1988 and Poggio and Girosi, 1990). The properties of RBF's are attracting a great deal of interest due to their rapid training, generality, and simplicity. In the context of an RBF neural network, the hidden units provide a set of "kernel functions" that constitute an arbitrary "basis" for the input patterns (vectors) when they are expanded into the hidden-unit space; these kernel

functions are called *radial basis functions*. These are generally non-linear functions that are built up into one function that can partition the pattern space successfully using hyperspheroids. Each kernel is associated with an activation region from the input space with respect to the data sample local densities and its output is fed to an output unit (Haykin, 1994)

An RBF network, which has a feedforward topology, can be considered as a two-layer fully interconnected network whose output nodes form a linear combination of the basis functions computed by the hidden layer nodes. The network is designed to perform a non-linear mapping from the input space to hidden space, followed by a linear mapping from the hidden space to the output space. An RBF is a multidimensional function which depends on the distance $r=\|x - c\|$ (where $\|\ .\ \|$ denotes a vector form) between the input vector x and the centre c. The first layer of an RBF network computes this distance of the input to the network to a set of stored memories. Each basis function is a non-linear function of a corresponding distance. The basis functions in the hidden layer produce a localised response to the input vector. The basis functions encode the inputs by computing how close the inputs are to the centres of the receptive field instead of just evaluating the weighted sum of the inputs.

The general form of an RBF is

$$F(x) = \sum_{i=1}^{N} w_i \phi(\|x - c_i\|) \tag{3}$$

where w_i are the coefficients or weights of c_i . N is the number of centres. The known data points c_i is taken to be the centres of radial basis functions. x is the input to the network.. $\{\varphi(\|x-c_i\|) \mid i = 1,2,..., N\}$ is a set of N arbitrary functions, known as *radial-basis functions*. This functional form is pre-selected with the centres c_i being some fixed points in N-dimensional space appropriately spanning the input domain.

Levenberg-Marquardt Algorithm (LM): The Levenberg-Marquardt algorithm is a variation of Newton's method that was designed for minimising functions that are sums of squares of other non-linear functions. This is very well suited to multilayer neural network training where the performance index is the mean squared error as given below

$$F(x)=E[e^T e]=E[(t-a)^T (t-a)]. \tag{4}$$

If each target occurs with equal probability, the mean squared error is proportional to the some of squared errors over the Q targets in the training set:

$$F(x)=\sum_{q=1}^{Q}(t_q-a_q)^T(t_q-a_q)=\sum_{q=1}^{Q}e_q^T e_q=\sum_{q=1}^{Q}\sum_{j=1}^{S^M}(e_{j,q})^2=\sum_{i=1}^{N}(v_i)^2, \tag{5}$$

where

$e_{j,q}$ is the j[th] element of the error for the q[th] input/target pair. Eq.5 is equivalent to the performance index,

$F(x)=\sum_{i=1}^{N}v_i^2(x)=V^T(x)V(x)$, for which Levenberg-Marquardt was designed. The key step in this algorithm is the computation of the Jacobian matrix. A variation of the back-propagation algorithm is used to perform this computation. To create the Jacobian matrix, the derivatives of the errors are computed, instead of the derivatives of the squared errors, with respect to the weights and biases of the network. (Hagan *et al.*, 1996)

Conic Section Function Neural Network (CSFNN): Conic Section Function Neural Network (Dorffner, 1994 and Yildirim and Marsland, 1997) is a unified framework based on hyperplanar and hyperspherical decision regions as special cases of conic sections. These are decision regions of MLP and RBF networks, respectively. The idea of CSFNN is to generalise unit function that contains all decision regions by providing a relationship between RBF and MLP units. Activation for CSFNN can be given in a general form as

$$y_j = \sum_{i=1}^{n-1}(x_i - c_{ij})w_{ij} - \cos\omega_j \sqrt{\sum_{i=1}^{n-1}(x_i - c_{ij})^2}$$

$$x_{n-1} \equiv 0 \tag{6}$$

where i and j are the indices referring to the units in the input and hidden layer, y is the output of network. This equation consists of two major parts analogous to the MLP and RBF networks. w_{ij} refers to the weights for MLP network and c_{ij} refers to the centre coordinates in an RBF network. ω_j is the opening angle of a cone to provide transition from radial basis function network to multilayer perceptron. The learning algorithm for CSFNN involves the determination of the centres and updating weights, centres, and the vertex angle of cone, ω, appropriately. The centres are determined using orthogonal least squares learning algorithm (Chen *et al.*, 1991). The vertex angle is chosen in such a way that the network would start as RBF. The weights, centres and angle values are updated using error back propagation so that the network would converge quickly. (Ozyilmaz and Yildirim, 2000 and Yildirm and Marsland, 1997)

Function Approximation Problem: In the application, various functions have been applied to the neural algorithms described above and the results were compared in terms of training time, the number of training epochs and the number of floating point operations.

First function applied to compare the neural algorithms was y=sin(4x) over the range (-1,1) changing with 0.1 step. x values in determined interval are given as input data to the network, and the function values corresponding to these are given as target data. Using this data, the network has been trained until it was reached to a determined error value, then, it was tested. In testing process, network output was calculated by using the values in interval (-1.05, 1.05) with 0.1 step. In Table 1, five different algorithms were compared for this function. These are back propagation, fast back propagation, Levenberg-Marquart, Radial Basis Function and Conic Section Function Neural Network. As can be seen from the Fig. 1, the convergence has changed depending on the number of hidden layer units for CSFNN, that is, it changes with the different number of determined centres.

Secondly, applying the same training and testing procedure, $y=\sin(4x^2)$ function has been investigated and the results was given in Table 2 and Fig. 2.

As another problem, the symmetric of the function in Matlab Neural Network Toolbox was used and the algorithms have been compared. Table 3 and Fig. 3 show the results.

Finally, y=200*(sin x)/x function has been investigated and the results was shown in Table 4 and Fig. 4. Multilayer Perceptron training algorithms except the one using Levenberg-Marquardt method did not provide any approximation for this function.

Results and Discussion

In this work, various function approximators using different neural algorithms (Multilayer Perceptron trained by back propagation, fast back propagation, and Levenberg-Marquart algorithms, Radial Basis Function and Conic Section Function Neural Network) have been investigated. Comparisons between the algorithms were given in terms of training time, training epochs and floating point operations. According to the results, it was seen that training time, training epochs and the floating point operations have been changed in every run of the program for the training algorithms of Multilayer Perceptron depending on the random weight initialization. For this reason, an average of the results of different runs was given for this type of network. This situation is not valid for RBF and CSFNN.

These approximators converge to the functions with the same number of compared parameters. The best results were obtained when Radial Basis Function was used. However, CSFNN also provides good results depending on the number of hidden layer, that is, the convergence changes with the number of determined centres. Even for the smaller number of centres in CSFNN, function curves follow the desired one in less training time and less training epoch doing less floating point operations. Hence, close results to RBFs can be obtained for CSFNN.

Consequently, function approximation problems can be solved with neural networks as an alternative to traditional methods by choosing appropriate algorithm and network structure.

Table I: Comparison of the Algorithms for Y=sin(4x) Function

Technique	Time~	Epochs*	Flops	
Backpropagation	66.1	10000	52442630	(average)
Fast Backpropagation	55.9	7920	41282507	(average)
Levenberg-Marquardt	0.8	6	521121	(average)
Radial Basis Network	0.6	6	37133	
Conic Section	0.8	4	24092	
Conic Section	0.8	5	31599	
Conic Section	0.9	7	48975	
Conic Section	1.0	8	58956	
Conic Section	1.1	10	81754	
Conic Section	1.2	12	108620	
Conic Section	1.2	13	123671	

~ Times are in seconds and may vary.
* Epochs refers to number of neurons for RBF and number of centers for CSFNN.

Table 2: Comparison of the Algorithms For Y=sin($4x^2$) Function

Technique	Time~	Epochs*	Flops	
Backpropagation	19.1	2180	114345	(average)
Fast Backprop	3.9	386	2017352	(average)
Levenberg-Marquardt	1.1	10	1046164	(average)
Radial Basis Network	0.8	8	54606	
Conic Section	1.4	6	39895	
Conic Section	1.2	10	81775	
Conic Section	1.2	12	108641	
Conic Section	1.2	14	139863	

~ Times are in seconds and may vary.
* Epochs refers to number of neurons for RBF and number of centers for CSFNN.

Table 3: Comparison of the Algorithms for Symmetric Matlab Function

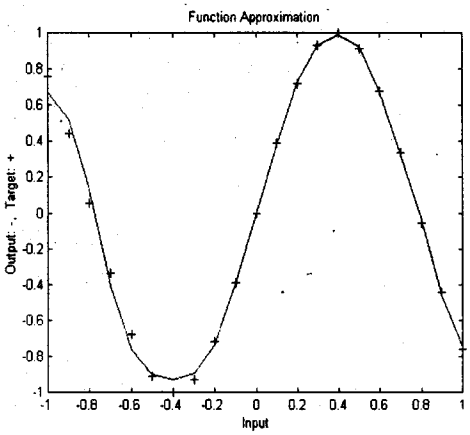
Technique	Time~	Epochs*	Flops	
Backpropagation	66.7	10000	52442630	(average)
Fast Backprop	25.0	3455	18019944	(average)
Levenberg-Marquardt	0.9	5	445103	(average)
Radial Basis Network	0.8	6	37133	
Conic Section	0.3	10	81712	
Conic Section	1.5	13	125232	

~ Times are in seconds and may vary.
* Epochs refers to number of neurons for RBF and number of centers for CSFNN.

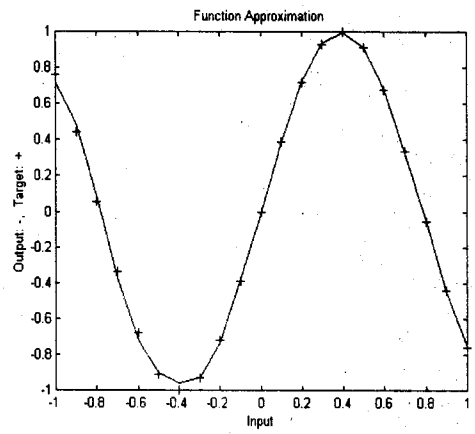
Table 4: Comparison of the Algorithms for Y = 200*(Sinx) / X Function

Technique	Time~	Epochs*	Flops
Levenberg-Marquardt	0.3	7	974306
Radial Basis Network	0.6	40	2526080
Conic Section	0.3	10	246079

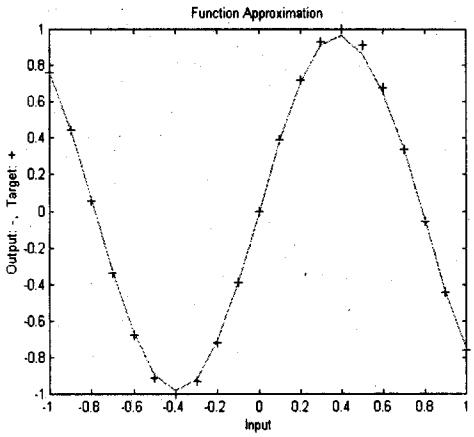
~ Times are in seconds and may vary.
* Epochs refers to number of neurons for RBF and number of centers for CSFNN.



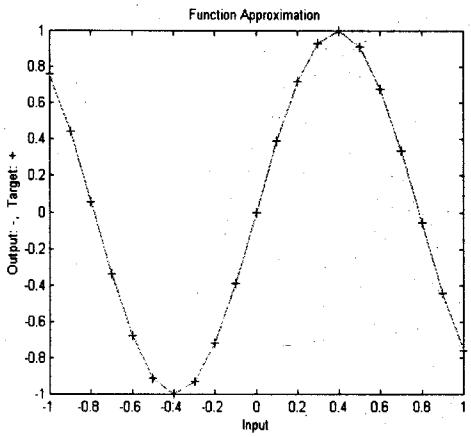
a)BP



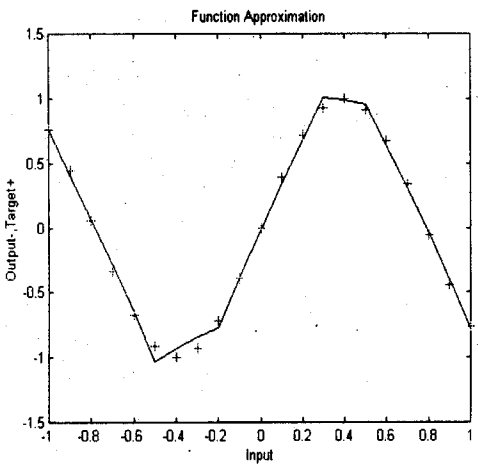
b)FBP



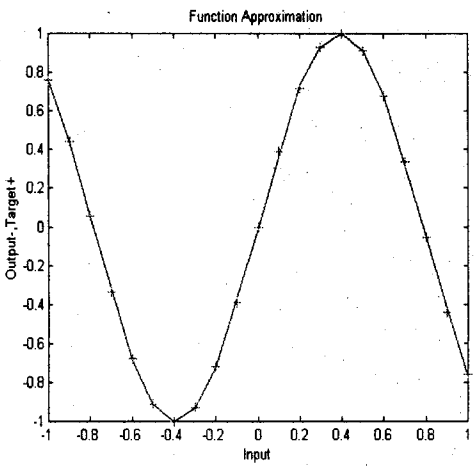
c)LM



d)RBF

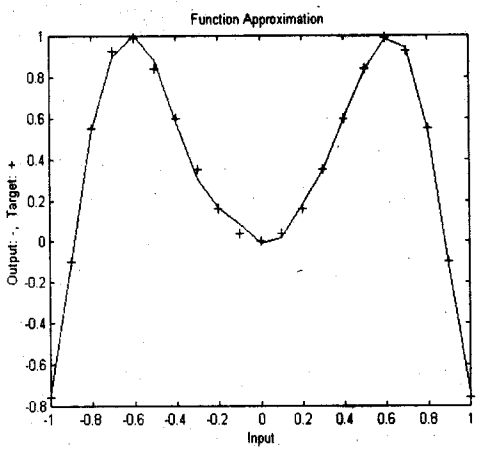


e)CSFNN with 7 centers

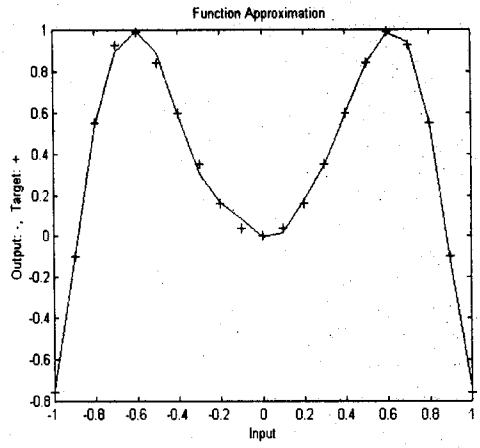


f) CSFNN with 13 centers

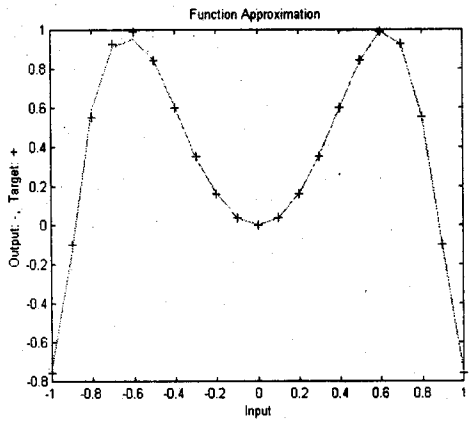
Fig. 1: Results of Test Values for Y=sin(4x) Function



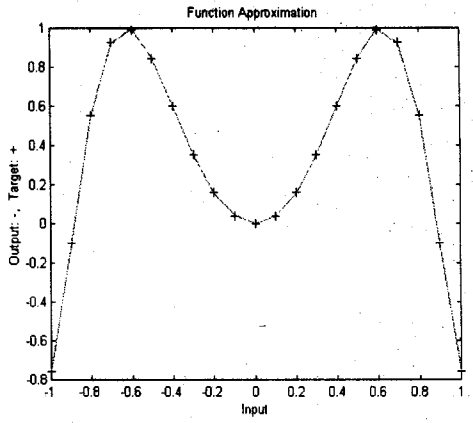
a) BP



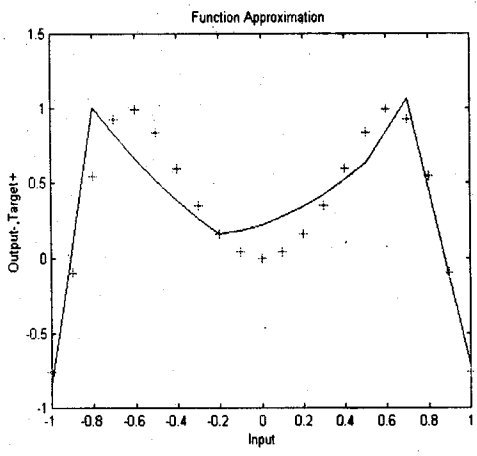
b) FBP



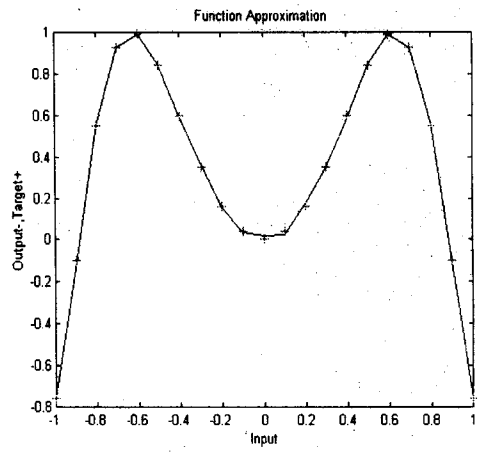
c)LM



d) RBF

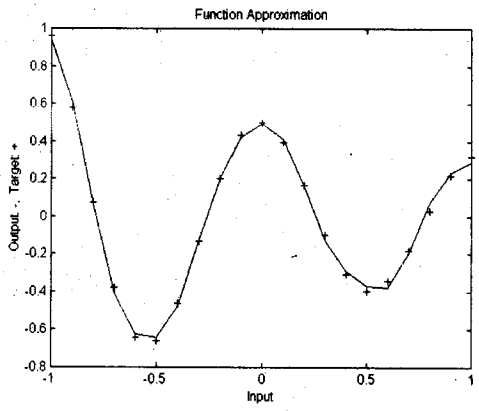


e) CSFNN with 6 centers

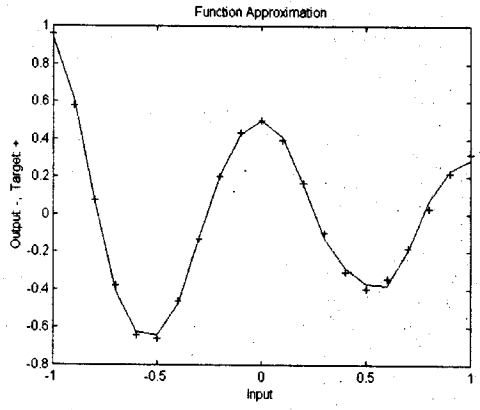


f) CSFNN with 14 centers

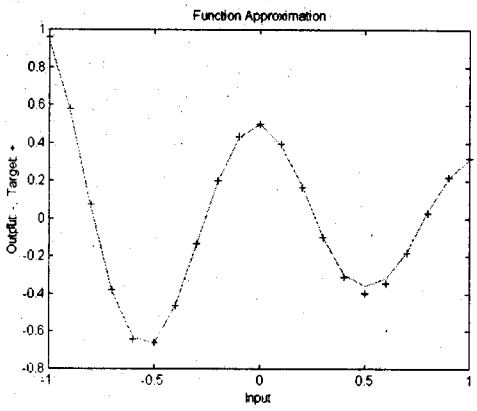
Fig. 2: Results of Test Values for Y=sin(4x^2) Function



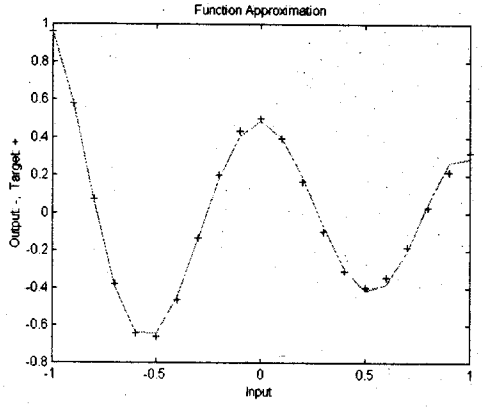
a) BP



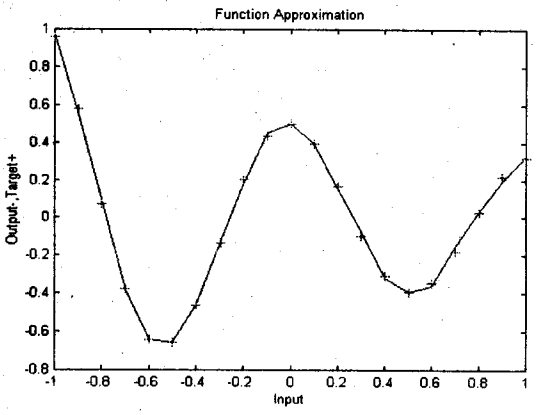
b) FBP



C) LM



d) RBF



e) CSFNN with 13 centers

Fig. 3: Results of Test Values for Symmetric of the Function in Matlab Neural Network Toolbox

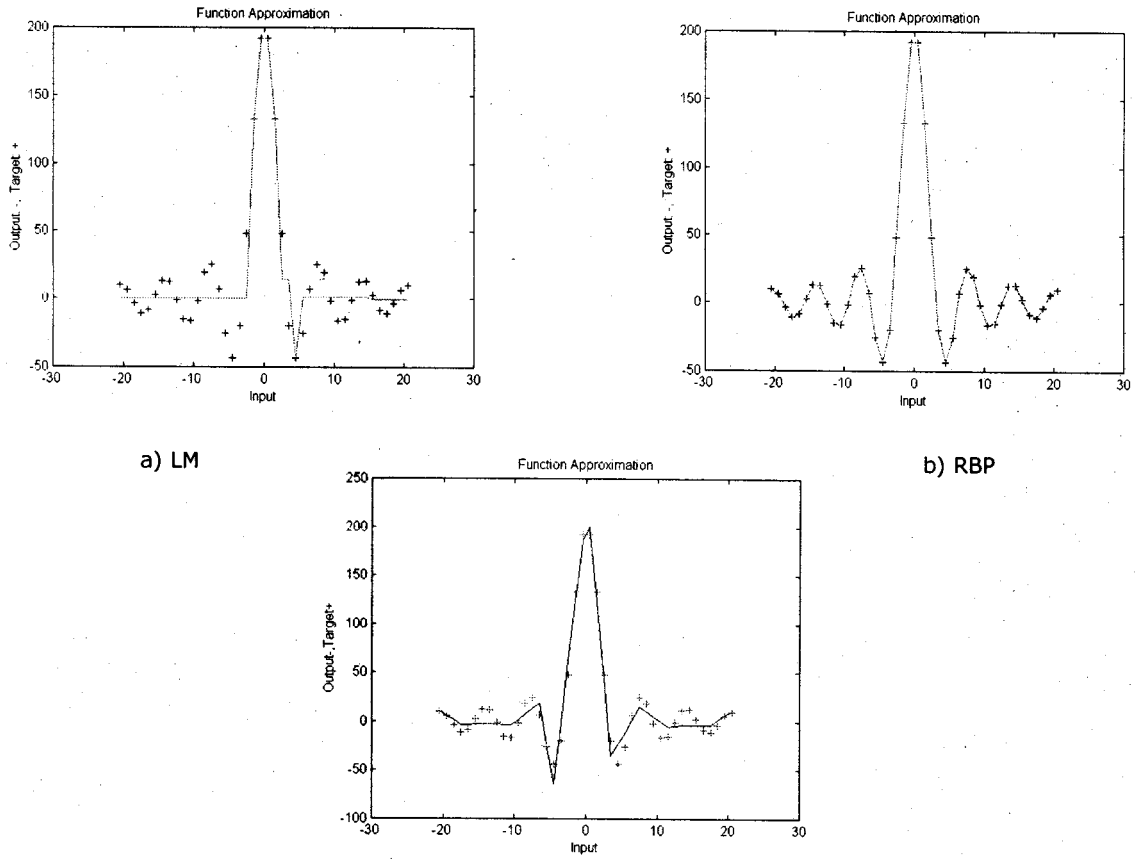


Fig. 4: Results of Test Values for Y=200*(Sin X)/x Function

References

Botros, S. M., Atkeson. C. G., 1991. Generalization properties of radial basis functions. In *Advances in Neural* Information Processing Systems 3 San Mateo, CA: Morgan Kaufmann, 707-713.

Broomhead, D.S., Lowe. D., 1988. Multivariable functional interpolating and adaptive networks. Complex Systems, 321-355.

Chen, S., Cowan, C. F. N., ve Grant, P.M., 1991. "Orthogonal Least Squares learning algorithm for Radial Basis Function Networks", IEEE Transactions on Neural Networks, 2: 302-309.

Dorffner, G. 1994. "A Unified Framework for MLPs and RBFNs: Introducing Conic Section Function Networks", Cybernetics and Systems, 25: 511-554.

Geva, S., ve Sitte, J., 1992. "A Constructive Method for Multivariate Function Approximation by Multilayer Perceptrons", IEEE Transactions on Neural Networks, 3 621-624.

Hagan, M. T., Demuth, H. B., ve Beale, M., 1996. Neural Network Design, PWS Publishing Company, Boston.

Haykin, S., 1994. Neural Networks: A Comprehensive Foundation, Macmillan College Publishing, New York.

Hush, D.R., and Horne, B.G., 1993. Progress in supervised neural networks. IEEE Signal Processing Magazine. 8-39,.

Jondarr, C. G. H., 1996. Back propagation family album. Technical Report C/TR96-05, Dept. of Computing, Macquarie Uni.

Kröse, B. J. A., Van der Smagt, P. P., 1993. An introduction to neural networks. Univ. of Amsterdam.

Lippmann, R.P., 1987. An introduction to computing with neural nets. IEEE ASSP Magazine, 4-22.

Özyılmaz, L., Yıldırım, T., 2000. "Using Conic Section Function Neural Network for Function Approximation", Proceedings of TAINN'2000 The 9th Turkish Symposium on Artificial Intelligence and Neural Networks, 21-23 409-412 (in Turkish).

Poggio, T., and Girosi, F., 1990. Networks for approximation and learning. Proc. IEEE, 78: 1481-1497,

Riedmiller, M., and Braun, H., 1997. RPROP - A fast adaptive learning algorithm. Proceedings of ISCIS *VII*.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., 1987. Learning internal representations by error propagation. in Parallel Distributed Processing, ed. D.E. Rumelhart and J. L. McClelland, 1: Cambridge, MA: MIT Press, 318-362.

Vysniauskas, V., Groen, F. C. A., Krose, B. J. A., 1993. The optimal number of learning samples and hidden units in function approximation with a feedforward network. Technical Report CS-93-15, Uni. of Amsterdam.

Yıldırım, T., ve Marsland, J.S., 1997. "A Unified Framework for Connectionist Models", in Perspectives in Neural Computing: Connectionist Representations, ed. J.A. Bullinaria, D. W. Glasspool, and G. Houghton, 26-39, Springer, London, UK.

Zurada, J. M., 1995. Introduction to Artificial Neural Systems, PWS Publishing, Boston.