Journal of Computer Science 3 (8): 592-599, 2007
ISSN 1549-3636

Rule Extraction from Radial Basis Functional Neural Networks by Using Particle Swarm Optimization

[1]M. R. Senapati ,[2]I.Vijaya and [3]P.K.Dash (SMIEEE)
[1]Konark Institute of Science and Technology, Jatni, Khurda, Orissa, India
[2]M.Tech. Computer Science, Utkal University, Bhubaneswar, Orissa, India
[3]College of Engineering, Bhubaneswar, Orissa, India

Abstract: Radial basis functional neural networks (RBFNN) provide an outstanding possibility for generating rules for solving pattern classification problems. One of the most important factors in RBFNN is finding out the center and spread. This paper examines rules extracted from RBF networks trained by Particle swarm Optimization (PSO). The selection of the RBFNN centers, spreads and the network weights can be viewed as a system identification problem. Our Simulation results using Radial Basis Functional Neural Networks (RBFNN) was applied to the PAT, WBC and IRIS data sets as a classification problem to illustrate the new knowledge extraction technique. The results indicate that training RBFNN with PSO can provide comparable generalization of rules with less training time.

Keywords: Radial Basis Functional Neural Networks (RBFNN); Particle Swarm Optimization (PSO), Wisconsin Breast Cancer (WBC), Pattern classification, Gradient descent method, Genetic Algorithm.

INTRODUCTION

A Radial Basis Functional neural network (RBFNN) is trained to perform a mapping from an m-dimensional input space to an n-dimensional output space. RBFNN's can be used for discrete pattern classification, function approximation, signal processing, control, or any other application, which requires a mapping from an input space to an output space. Many recent developments of RBFNN and its applications can be found in *Neuro computing* special issues on RBFNN[1, 2].

An RBFNN consists of the m-dimensional input x being passed directly to a hidden layer. Suppose there are c neurons in the hidden layer. Each of the c neurons in the hidden layer applies an activation function, which is a function of the Euclidean distance between the input and an m-dimensional prototype vector. Each hidden neuron contains its own prototype vector as a parameter. The output of each hidden neuron is then weighted and passed to the output layer. The outputs of the network consist of sums of the weighted hidden layer neurons. Figure 1 shows a schematic form of an RBFNN network. It can be seen from the basic architecture, that the design of an RBFNN requires several decisions, including the following:

1. How many neurons will reside in the hidden layer? (i.e., what is the value of the integer c);
2. What are the values of the prototypes (i.e., what are the values of the v vectors)?
3. What function will be used at the hidden units (i.e., what is the function g (·))?
4. What weights will be applied between the hidden layer and the output layer?

The performance of an RBFNN network depends on the number and location (in the input space) of the centers, the shape of the RBFNN functions at the hidden neurons, and the method used for determining the network weights. Some researchers have trained RBFNN networks by selecting the centers randomly from the training data[3]. Some have used unsupervised procedures (such as the k-means algorithm) for selecting the RBFNN centers[2], while others have used supervised procedures for selecting the RBFNN centers[4].

This study will be divided into four parts: We begin with motivation and detailed description of Radial basis functional Neural Network (RBFNN) in first part and second part. The third part will review the most recent optimization technique namely Particle Swarm Optimization (PSO). We will end with the rule extraction for different data types for pattern recognition and its future avenue.

Several training methods separate the tasks of prototype determination and weight optimization for

Corresponding Author: M.R.Senapati, Konark Institute of Science & Technology, Jatni, Khurda, Orissa, India

classification and rule generation. This trend probably arose because of the quick training that could result from the separation of the two tasks. In fact, one of the primary contributors to the popularity of RBFNN networks was probably their fast training times as compared to gradient descent training (including back propagation) shown in Figure 1, it can be seen that once the prototypes are fixed and the hidden layer function $g(\cdot)$ is known, the network is linear in the weight parameters w. At that point training the network becomes a quick and easy task that can be solved via linear least squares. (This is similar to the popularity of the optimal interpolative net that is due in large part to the efficient non- iterative learning algorithms that are available[5,6])

Training methods that separate the tasks of prototype determination and weight optimization often do not use the input—output data from the training set for the selection of the prototypes. For instance, the random selection method and the k-means algorithm result in prototypes that are completely independent of the input—output data from the training set. Although this results in fast training, it clearly does not take full advantage of the information contained in the training set.

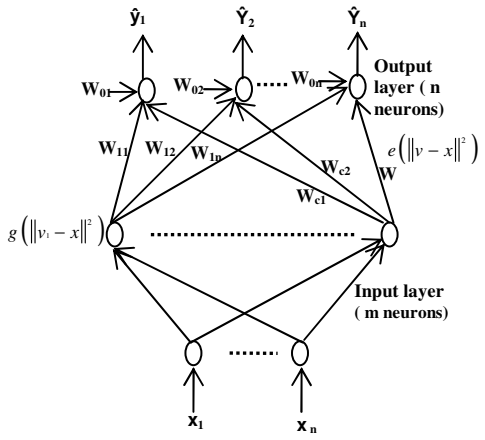

Fig. 1: Radial Basis Functional Network

Gradient descent training of RBFNN networks has proven to be much more effective than more conventional methods[4]. However gradient descent training can be computationally expensive. This paper extends the results of[4] and formulates a training method for RBFNN's based on Particle Swarm Optimization. This new method proves to be quicker than gradient descent while still providing performance at the same level of effectiveness.

Training a neural network is, in general, a challenging nonlinear optimization problem. Various derivative-based methods have been used to train neural networks, including gradient descent[4], Kalman Filtering[7,8], and the well-known back-propagation[9]. Derivative-free methods, including genetic programming[10] , learning automata[11] , and simulated annealing[12] have also been used to train neural networks.

Derivative-free methods have the advantage that they do not require the derivative of the objective function with respect to the neural network parameters. They are more robust than derivative-based methods with respect to finding a global minimum and with respect to their applicability to a wide range of objective functions and neural network architectures. However, they typically tend to converge more slowly than derivative-based methods. Derivative-based methods have the advantage of fast convergence, but they tend to converge to local minima. In addition, due to their dependence on analytical derivatives, they are limited to specific objective functions and specific types of neural network architectures.

INTERPRETATION OF RADIAL BASIS FUNCTION NEURAL NETWORKS

The multi layered feed forward network (MFN) is the most widely used neural network model for pattern classification applications. This is because the topology of the MFN allows it to generate internal representations tailored to classify the input regions that may be either disjointed or intersecting. The hidden layer nodes in the MFN can form hyper planes to partition the input space into various regions and the output nodes can select and combine the regions that belong to the same class. Back propagation (BP) is the most widely used training algorithm for the MFN's.

Recently researchers have begun to examine the use of Radial Basis Function neural networks (RBFNN) for pattern classification problems due to a number of drawbacks of BP-trained networks. Although a BP network produces decision surfaces that effectively separate training examples of different classes, this does not necessarily result in the most plausible or robust classifier. The decision surfaces of BP networks may not take on any intuitive shapes because regions of the input space not occupied by training data are classified arbitrarily, not according to proximity to training data. In addition, BP networks have no mechanism to detect that a case to be classified has fallen into a region with no training data. This is a serious drawback since the power system operates within a wide range of system and fault conditions.

The RBFNN consists of an input layer made up of source nodes and a hidden layer of a sufficiently high dimension. The output layer supplies the response of the network to the activation patterns applied to the input layer. The nodes within each layer are fully connected to the previous layer as shown in the Figure 1. The input variables are each assigned to a node in the input layer and pass directly to the hidden layer without weights. The hidden nodes, or units, contain the radial basis functions (RBFNN's) and are represented by the bell-shaped curve in the hidden nodes as shown in the Fig 1.

RBFNN Algorithm: This section describes how we used an RBFNN network to classify the data sets. RBFNN used here has an input layer, a hidden layer consisting of Gaussian node function, an output layer, and a set of weights, W to connect the hidden layer and output layer. We denote x to be the input vector to the network, where x = (x_1, x_2, x_3, ……x_D), and D is the embedding dimension. We call o the ANN output vector, where o = (o_1, o_2, o_3 , …. o_n) T is the number of out put nodes. We have P training patterns. The RBFNN classification problem is to approximate the mapping from the set of inputs,

$$x = \{x(1), x(2), ….., x(P)\}, \quad (1)$$

to the set of outputs,

$$o=\{o(1), o(2), o(3) , ……., o(P)\} \quad (2)$$

For an input vector x(t) , the output of j^{th} output node produced by an RBFNN is given by

$$o_i(t) = \sum_{i=1}^{m\,tot} W_{ij} \varnothing_i(t) = \sum_{i=1}^{m\,tot} W_{ij} e^{-\left\| \frac{x_i(t) - c_i}{2\sigma_i^2} \right\|} \quad \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

Where C_i is the center of the i^{th} hidden node, σ_i is the width of the ith center, and m_{tot} is the total number of hidden nodes. Using vector notation, let
$\varphi = (\phi_1(t),\phi_2(t),…..,\phi_{mtot}(t))$ and $w_j = (w_{1j}, w_{2j} , ….., w_{mto}(t_j))$ and RBFNN output can be written as $o_j = w_j\varphi^{T(}t)$.

The cost function of the network for the jth output is then calculated as e = (d – o_j) where d = desired output. The RBFNN classifier contains four sets of parameters that have to be learned form the examples. They are the centers, $c_i(t)$, number of centers m_{tot}, variances σ_I, and weights w_{ij},. We denote all the RBFNN's centers by C_{whole}. In our implementation of RBFNN, classes do not share centers. Each of these sets of centers is trained with a separate PSO clustering run. In each PSO run (corresponding to a different class), only the training vectors for that class would be used for clustering as described in the next section.

Once the RBFNN centers are initialized by PSO then the weights are updated according to the following:

$$w_{ij}(t+1) = w_{ij}(t)+2e \varnothing_i \ (t)$$

The centers are then updated according to the following:

$$c(t+1) = c(t) + \frac{2 e w_{ij} \varnothing_i (x_i - c_{ij})}{\sigma_i^2} \quad \ldots\ldots\ldots\ldots (4)$$

The variances are not updated in this experiment to minimize the time and thus $\sigma_{ij} = \sigma_{ij}$
There are several reasons for using an RBFNN in our classification problem. First many neural networks require nonlinear optimization for training.

The second reason for employing a RBFNN classifier is that the internal representation of training data of an RBFNN is intuitive. Each RBFNN center approximates a cluster of training of data vectors that are close each other in Euclidean space. When a vector is input to the RBFNN, the center near to that vector becomes strongly activated, in turn activating certain output nodes.

The hypothesis space implanted by these learning machines is constituted by functions of the form

$$f(x, w, v) = \sum_{i=1}^{m} w_k \varnothing_k (x, v_k) + w_0 \quad \ldots\ldots\ldots\ldots (5)$$

The nonlinear activation function Øk expresses the similarity between any input pattern x and the center v_k by means of a distance measure. Each function $Ø_k$ defines a region in the input space (receptive field) on which the neuron produces a appreciable activation value. If the common case when the Gaussian function is used, the center C_k of the function σ_k defines the prototype of input cluster k and the variance $Ø_k$ the size of the covered region in the input space.

The local nature of RBFNN networks makes them an interesting platform for performing rule extraction. However, the basis functions overlap to some grade in order to give a relatively smooth representation of the distribution of training data[13,14]. This overlapping is a shortcoming for rule extraction. Few rule extraction methods directed to RBFNN have been developed[15,16,17].

The rule extraction method for RBFNN derives descriptions in the form of ellipsoid. Initially, assigning each input pattern to their closest center of RBFNN node according to the Euclidean distance function a partition of the input space is made. When

assigning a pattern to its closest center, this one will be assigned to the RBFNN node that will give the maximum activation value for that pattern. From these partitions the ellipsoid are constructed. Next, a class label is assigned for each center of RBFNN units. Output value of the RBFNN network for each center is used in order to determine this class label. Then, for each node an ellipsoid with the associated partition data is constructed. Once determined the ellipsoid, they are transferred to rules. This procedure will generate a rule by each node.

PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) is a population based stochastic search process, modeled after the social behavior of a bird flock[18,19,20]. The algorithm maintains a population of particles, where each particle represents a potential solution to an optimization problem.

In the context of PSO, a swarm refers to a number of potential solutions to the optimization problem, where each potential solution is referred to as a particle. The aim of the PSO is to find the particle position that results in the best evaluation of a given fitness (objective) function.

Each particle represents a position in N_d dimensional space, and is "flown" through this multi-dimensional search space, adjusting its position towards both

- The particle's best position found thus far, and
- The best position in the neighborhood of that particle.

Each particle I maintains the following information :

- x_i : The *current position* of the particle.
- v_i : The *current velocity* of the particle.
- y_i : The *personal best position* of the particle.

Using the above notation, a particle's position is adjusted according to

$$v_{i,k}(t+1)=wv_{i,k}(t)+c_1r_{1,k}(t)\,(y_{i,k}(t)-x_{i,k}(t))+c_2r_{2,k}(t)(\hat{y}_k(t)-x_{i,k}(t)).....(6)$$

$$x_i(t+1)=x_i(t)+v_i(t+1).....(7)$$

where w is the inertia weight c_1 and c_2 are the acceleration constants, $r_{1,j}(t)$, $r_{1,j}(t) \sim U(0,1)$, and k=1, … ., N_d. The velocity is thus calculated based on three contributions: 1) a fraction of the previous velocity, 2) the cognitive component which is a function of the distance of the particle from its personal best position, and 3) the social component which is a function of the distance of the particle from the best particle found thus far (i.e; the best of the personal bests).

The personal best position of the particle is calculated as

$$y_i(t+1) = \begin{cases} y_i & \text{if } f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1) & \text{if } f(x_i(t+1)) < f(y_i(t)) \end{cases} \text{.....(8)}$$

Two basic approaches to PSO exists based on the interpretation of the neighborhood of particles. Equation (6) reflects the *gbest* version of PSO where, for each particle, the neighborhood is simply the entire swarm. The social component then causes particles to be drawn toward the best particle in the swarm. In the *lbest* PSO model, the swarm is divided into overlapping neighborhoods, and the best particle of each neighborhood is determined. For the *lbest* PSO model, the social component of equation(6) changes to

$$c_2r_{2,k}(t)(\hat{y}_k(t)-x_{i,k}(t)).....(9)$$

where $\hat{y}_j$ is the best particle in the neighborhood of the i-th particle.

The PSO is usually executed with repeated application of equations (6) and (7) until a specified number of iterations has been exceeded. Alternatively, the algorithm can be terminated when the velocity updates are close to zero over a number of iterations.

PSO Clustering : In the context of clustering, a single particle represents the N_c cluster centroid vectors. That is, each particle xi is constructed as follows: $x_i = (m_{i1}, \ldots, m_{ij}, \ldots, m_iN_c)$ ……..(10) where m_{ij} refers to the j-th cluster centroid vector of the i-th particle in the cluster C_{ij}. Therefore, a swarm represents a number of candidate clustering for the current data vectors. The fitness of particles is easily measured as the quantization error,

$$J_c = \frac{\sum_{i=1}^{N_c}[\sum \forall z_p \in C_{ij} d(Z_p, m_j)/|C_{ij}|]}{Nc} \text{.....(11)}$$

Where d is defined in equation,

$$d(Z_p, m_j) = \sqrt{\sum_{k=1}^{N_d}(Z_{pk} - m_{jk})^2} \text{.....(12)}$$

where k subscripts the dimension.

$$m_j = \frac{1}{n_j} \sum_{\forall Zp \in Cj} Z_p \text{.....(13)}$$

and $|C_{ij}|$ is the number of data vectors belonging to the cluster C_{ij}, ie; the frequency of that cluster.

This section first presents the standard gbest PSO for clustering data into a given number of clusters and

then shows the PSO algorithm can be used to improve the performance of Radial basis functional Neural Network (RBFNN) for classification.

gbest PSO clustering Algorithm

Using the standard gbest PSO, data vectors can be clustered as follows :

1. Initialize each particle to contain N_c randomly selected cluster centroids.
2. For t = 1 to t_{max} do
   a) For each particle i do
   b) For each data vector Z_p
      i) calculate the Euclidean distance $d(Z_p, m_{ij})$ to all cluster centroids C_{ij}
      ii) Assign Z_p to cluster C_{ij}, such that $d(Z_p, m_{ij}) = \min_{\forall c=1, ..., Nc} \{d(Z_p, m_{ic})\}$
      iii) calculate the fitness function using (6)
   c) Update the global best and local best positions
   d) Update the cluster centroids using equations (6) and (7). where t_{max} is the maximum number of iterations.

DISCUSSION

In order to evaluate the performance of the rule extraction algorithm, we carried out a two fold experiment with PAT[21], WBC, and IRIS data sets. The time for the error to converge with the center optimized by PSO was compared with the center optimized by genetic algorithm are presented in Table.7. The result shows if the RBFNN centers are optimized by PSO then the network takes less time for getting trained Table.7. In case of IRIS data set the overlap in the rule extracted is better than training RBFNN with genetic algorithm[22]. In case of WBC the convergence of the genetic algorithm takes more time than PSO[23] . The PAT[21] data set was included to show that this methodology can handle any type of classification task. The algorithms associated to the extraction method were simulated using MATLAB v6.5.

Simulation environment: In this section we describe and illustrate the use of Particle Swarm Optimization training for the centers of an RBFNN network. We tested the algorithms of the previous sections on the classical PAT dataset[21], and Wisconsin breast cancer (WBC) data set, and IRIS dataset.

PAT database: The PAT data set contains training set consisting of 450 exemplars and the test set containing 430 exemplars for a total of 880 exemplars.

- **WBC database:** The WBC data contains 400 exemplars and the test set containing 299 exemplars for a total of 699 exemplars.
- **IRIS plants database:** The IRIS data contains 50 exemplars from each category for a total of 150 patterns. We randomly divided the patterns into training and test sets, containing 34 exemplars from each category. The rest from each category were used for testing purpose.

The input data were normalized by replacing each feature value x by $x = (x - \mu x) / \sigma x$ where μx and σx denote the sample mean and standard deviation of this feature over the entire data set. The networks are trained to respond with the target value $y_{ik} = 1$, and $y_{jk} = 0 \ \forall \ j \neq i$, when presented with an input vector x_k from the i^{th} category.

The MATLAB m-files were used to generate the simulation results presented in this section. The training algorithms were initialized with prototype vectors randomly selected from the input data on a two fold basis and with the weight matrix W set to 1 and σ initialized to 1.

SIMULATION RESULTS

Fitness Convergence: Fig.2, Fig.3, Fig.4 shows the fitness convergence of PAT, WBC and IRIS datasets.

Error: Indication of error is a key attribute for any simulation results. Our observation through simulation for errors of PAT[21], WBC and IRIS datasets respectively are shown in Fig.5, Fig.6 and Fig.7,

Tabular Data: The results of centers obtained by RBFNN from our simulation studies are shown in tables. Table-1, Table-3, and Table-5 shows the centers of PAT, WBC and IRIS datasets respectively. Table-2, Table-4, and Table-6, shows the weights obtained for PAT, WBC and IRIS datasets respectively. Table-7 shows the time required to train the data sets with PSO and Genetic Algorithm.

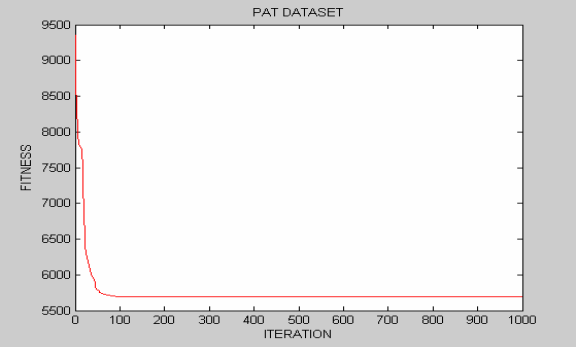

Fig.2: Fitness Convergence of Pat Dataset using PSO

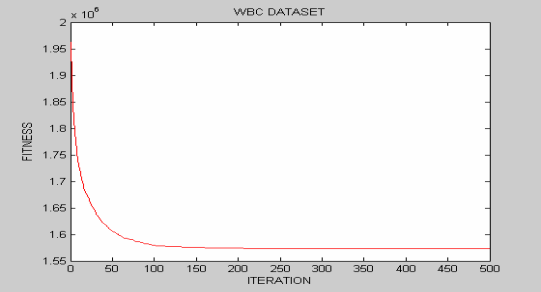

Fig.3: Fitness Convergence of WBC Dataset using PSO

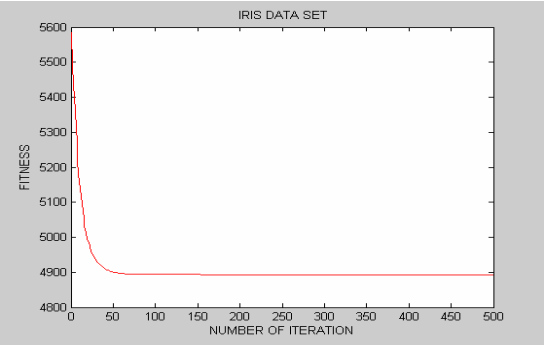

Fig.4: Fitness Convergence of IRIS Dataset using PSO

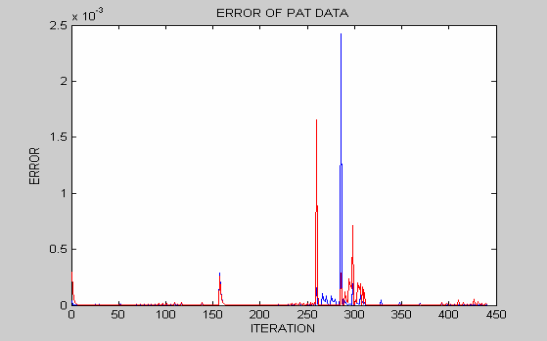

Fig.5: Error minimization of PAT Dataset using RBFNN

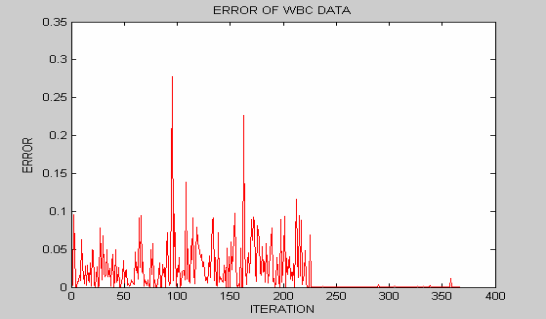

Fig.6: Error minimization of WBC Dataset using RBFNN

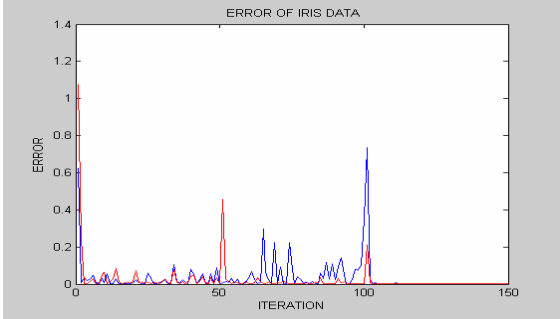

Fig.7: Error minimization of IRIS Dataset using RBFNN

Table 1: Centers obtained from RBFNN for PAT Data

P	11.8460	7.0010
A	2.0048	2.6812
T	11.5228	6.6686

Table 2: Weights obtained from RBFNN for PAT Data

-2.9305	4.1148
0.0405	-1.5288
-1.5702	3.7191

Table 3: Centers obtained from RBFNN for WBC Data

W	0.1845	2.5741	4.7897	3.2428	1.7303
B	0.1038	1.3345	2.3379	0.9890	
C	4.8878	6.0007	6.2246	12.3465	16.6876
	11.4862	10.6686	2.0459	13.2855	

Table 4: Weights obtained from RBFNN for WBC Data

-3.1954	-3.1954
0.5514	0.5514

Table 5: Centers obtained from RBFNN for IRIS Data

I	2.0790	-2.9273	7.0868	8.5308
R	2.0488	-2.8025	6.8230	8.1882
I	-1.1022	1.7729	-21356	-2.0742
S				

Table 6: Weights obtained from RBFNN for IRIS Data

-0.2179	-0.0954
-0.5244	-0.3841
0.1900	-0.0424

Table 7: Time taken for RBFNN to get trained with center optimized by PSO and Genetic Algorithm.

	PSO Time in Sec	Genetic Algorithm Time in Sec
PAT	5.0630	5.0940
WBC	4.0790	14.8130
IRIS	2.0160	2.0320

Rule for classification of PAT data sets

If (o(r, 1) >= 1.0000 or o(r, 1) <= 1.0122) and (o(r, 2) >= 0.5936 and o(r, 2) <= 1.0000) then Class= 1

If (o(r, 1) >= 1.0050and o(r, 1) <= 1.0161) and (o(r, 2) >= 0.3937 and o(r, 2) <= 0.8124) then Class=2

If (o(r, 1) >= 1.0003 and o(r, 1) <= 1.0374) and (o(r, 2) >= -0.4115 and o(r, 2) <= 0.9869) then Class=3

Rule for classification of WBC data sets

If (o(r, 1) >= -0.6914 or o(r, 1) <= 1) and (o(r, 2) >= -0.6914 or o(r, 2) <= 1) then Class=1

If (o(r, 1) >= 0.9894 or o(r, 1) <= 1) and (o(r, 2) >= 0.9894 or o(r, 2) <= 1) then Class=2

Rule for classification of IRIS data set

If (o(r, 1) >= 0.3060 or o(r, 1) <=0.9987) and (o(r, 2) >= 0.3060 or o(r, 2) <= 0.9987) then Class=1

If (o(r, 1) >= 0.9989 orl o(r, 1) <= 1.0000) and (o(r, 2) >= 0.9989 or o(r, 2) <= 0.9998) then Class=2;

If (o(r, 1) >= 1.0000 or o(r, 1) <= 1.0000) and (o(r, 2) >= 1.0000 or o(r, 2) <= 1.0000) then Class=3

CONCLUSION

The success of neural network architecture depends heavily on the availability of effective learning algorithms. The theoretical strength of the Particle Swarm Optimization (PSO) is yet to be used in hundreds of technologies, and this paper demonstrates that RBFNN network training is yet another fruitful application of Particle Swarm Optimization (PSO). Our simulation using MATLAB v6.5 verifies that initialization of the centers through Particle Swarm Optimization provides better performance. Further research could focus on the application of Particle Swarm Optimization (PSO) training to RBFNN networks with alternative forms of the generator function. (Recall that in this paper the deviations and weight matrix was initialized to ones.) Applying these techniques to large problems to obtain experimental verification of the computational savings of training the centers by PSO can be included as a future work

REFERENCES

1. V. D. Suanchez, A. (Ed.), 1998. Special Issue on RBFNN Networks, Part I, Neuro computing.
2. V. D. Suanchez, A. (Ed.), 2004. Special Issue on RBFNN Networks, Part II, Neuro computing.
3. D. Broomhead, D. Lowe, 1998.Multivariable functional interpolation and adaptive networks, Complex Systems v2, pp. 321-355
4. N. Karayiannis, 1999. Reformulated radial basis neural networks trained by gradient descent, IEEE Transactions on Neural Networks, Volume:10, Issue: 3, pp. 657-671.
5. D. Simon, Distributed fault tolerance in optimal interpolative nets, 2001. IEEE Transaction on Neural Networks, Volume: 12, Issue: 6, pp. 1348-1357.
6. S. Sin, R. DeFigueiredo, 1993. Efficient learning procedures for optimal interpolative nets, Neural Networks, Volume:6 , Issue:1, pp. 6 99-113.
7. John Sum, Chi-sing Leung, Gilbert H. Young, and Wing-kay Kan, 1999. Kalman Filtering Method in Neural-Network, Training and Pruning, IEEE Transactions On Neural Networks,. Volume:10, Issue:1, pp. 161-166.
8. Y. Zhang, X. Li, 1999. A fast U-D factorization-based learning algorithm with applications to nonlinear system modeling and identification, IEEE Transaction on Neural Networks, Volume:10, Issue: 4, pp. 930 - 938 .
9. Duro, R.J.; Reyes, J.S, 1999. Discrete-time back propagation for training synaptic delay-based artificial neural networks, IEEE Transaction on Neural Networks, Volume: 10, Issue: 4, pp. 779-789.
10. S. Chen, Y. Wu, B. Luk, 1999. Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks, IEEE Transaction on Neural Networks, Volume: 10, Issue: 5, pp. 1239-1243.
11. McKay B.; Wills, M.J.; Hidden, H.G.; Montague, G.A. and Barton, G.W. March, 1996. Identification of industrial processes using genetic programming. Proceedings of International Conference on Identification in Engineering Systems, pp.328-337.

12. S. Kirkpatrick, Cl. Gelatt, M. Vecchi, 1983. Optimization by simulated annealing, Volume: 220, no: 4598, pp. 671 - 680.
13. J. Moody and C. Darken, 1989. Fast learning in networks of locally tuned processing units. Neural computation, 1(2): pp.281–294.
14. X. Fu and L. Wang, 2001. Rule extraction by genetic algorithms based on a simplified RBFNN neural network. In Proceedings of the Congress on Evolutionary Computation volume: 2, pp.753–758.
15. K. Huber and M. Berthold, 1995. Building precise classifiers with automatic rule extraction. Neural Networks, 1995. Proceedings., IEEE International Conference on Neural Networks, volume:3, pp. 1263–1268.
16. K. McGarry, S. Wermter, and J. MacIntyre, 2001. Knowledge extraction from local function networks. In Proceedings of the International Joint Conference on Neural Networks, pp.765–770.
17. J Kennedy, RC Eberhart, Y Shi. 1995, "Particle Swarm Optimization", Proceedings of the IEEE International Joint Conference on Neural Networks, volume: 4, pp. 1942-1948.
18. J Kennedy, RC Eberhart, Y.Shi. 2002, "Swarm Intelligence",International Journal of Computer Research, pp.434-452.
19. Xiaohui Hu, Yuhui Shi, and Russ Eberhart, 2004. Recent advances in particle swarm. In *Proceedings of* IEEE Congress on Evolutionary Computation (CEC), pp. 90-97.
20. J. Kennedy, 2000. Stereotyping: improving particle swarm performance with cluster analysis. In *Proceedings of the* IEEE Congress on Evolutionary Computation (CEC), pp.1507-1512.
21. S. K. Pal and S. Mitra. 1994, "Fuzzy versions of Kohonen's net and MLP-based classification: Performance evaluation for certain nonconvex decision regions"', Information Sciences, Volume: 76, pp. 297-337.
22. Kenneth J. McGarry, John Tait, Stefan Wermter, and John Macintyre, 1999, "Rule-Extraction From Radial Basis Function Networks", IEE, pp 613-618.
23. Bahram G. Kermani, Mark W. White, H. Tory Nagle 1997, "Feature Extraction By Genetic Algorithms for Neural Networks in Breast Cancer Classification", IEEE, pp.831-832.