We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

26

A Radial Basis Function Neural Network with Adaptive Structure via Particle Swarm Optimization

Tsung-Ying Sun, Chan-Cheng Liu, Chun-Ling Lin, Sheng-Ta Hsieh and Cheng-Sen Huang
National Dong Hwa University
Taiwan, R.O.C.

1. Introduction

Radial Basis Function neural network (RBFNN) is a combination of learning vector quantizer LVQ-I and gradient descent. RBFNN is first proposed by (Broomhead & Lowe, 1988), and their interpolation and generalization properties are thoroughly investigated in (Lowe, 1989), (Freeman & Saad, 1995). Since the mid-1980s, RBFNN has been used to apply on many applications, such as pattern classification, system identification, nonlinear function approximation, adaptive control, speech recognition, and time-series prediction, and so on. In contrast to the well-known Multilayer Perceptron (MLP) Networks, the RBF network utilizes a radial construction mechanism. MLP were trained by the error Back Propagation (BP) algorithm, since the RBFNN has a faster training procedure substantially and adopts typical two-stage training scheme, it can avoid solution to fall into local optima.
A key point of RBFNN is to decide a proper number of hidden nodes. If the hidden node number of RBFNN is too small, the generated output vectors may be in low accuracy. On the contrary, it with too large number of hidden nodes may cause over-fitting for the input data, and influences global generalization performance. In conventional RBF training approach, the number of hidden node is usually decided according to the statistic properties of input data, then determine the centers and spread width for each hidden nodes by means of k-means clustering algorithm (Moddy & Darken, 1989). The drawback of this approach is that the network performance is depended on the pre-selected number of hidden nodes. If an unsuitable number is chosen, RBFNN may present a poor global generalization capability, as slow training speed, and requirement for large memory space. To solve this problem, the self-growing RBF techniques were proposed in (Karayiannis & Mi, 1997), (Zheng et al, 1999). However, the predefined parameters and local searching on solution space cause the inaccuracy of approximation from a sub-solution.
Evolutionary computation is a globally optimization technique, where the aim is to improve the ability of individual to survive. Among that, Genetic Algorithm (GA) is a parallel searching technique that mimics natural genetics and the evolutionary process. In (Back et al, 1997), they employed GA to determine the RBFNN structure so the optimal number and distribution of RBF hidden nodes can be obtained automatically. A common approach is applied GA to search for the optimal network structure among several candidates

constructed initially by the unsupervised clustering method (Chen et al, 1999). However, its results depend on the pre-selected RBFNN structures which may not be appropriate. Another method is to fix the number of RBF nodes and adopted GA to search optimal network parameters, for example, centers and spread widths for RBF hidden nodes, and the weights connected to the output layer (Aiguo & Jiren, 1998). This method requires heavy computational cost while the number of RBF hidden nodes is too large, the dimension of each chromosome has to extend to corresponding length. It will spend too much time for training. GA based self-growing RBF network training method was proposed by (Yunfei & Zhang, 2002) to overcome the mentioned drawbacks. It searches single parameter, the cluster distance factor, which can avoid organizing a large dimension in a chromosome. It performs a fast training speed and well convergences while the GA operators (reproduction, recombination, and mutation, etc.) and fitness evaluation is properly applied. However, GA-based approaches are poorer in several aspects, as premature convergence and falling into local optima, than new evolutionary computation techniques.

The particle swarm optimization (PSO) is a novel and popular search algorithm based on the simulation of the social behavior of birds within a flock in evolutionary computation. As opposed to (Yunfei & Zhang, 2002), this paper proposes a PSO based RBFNN self-structure algorithm to overcome the drawbacks that mentioned above. PSO is a swarm intelligence method that roughly models the social behavior of swarms and has been proved to be efficient on many optimization problems in science and engineering. The social behavior of PSO allows particles to stochastically return toward previously successful regions in the search space. We propose a PSO-based approach for searching the optimal cluster distance factor to provide a suitable criterion on self-structure RBFNN training. The results of simulation experiments exhibit the rapid convergence and more better optimal solutions than other related approaches. Furthermore, it yields efficient training for constructing RBFNN.

The paper is organized as follows. Section II describes structure and the training of the RBF network. Section III describes the principle and procedures of the self-structure RBF algorithm. Section IV presents the application of a PSO to search the cluster distance factor. Section V evaluates our method for modeling nonlinear function and predicting time series by RBF Network and comparing the results with the GA-RBF Network and K-means methods. Section VI is the conclusion.

2. Radial Basis Function Neural Network

Generally, a RBFNN consists of three layers: the input layer, the RBF layer (hidden layer) and the output layer. The inputs of hidden layer are the linear combinitions of scalar weights and the input vector $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T$, where the scalar weights are usually assigned unity values. Thus the whole input vector appears to each neuron in the hidden layer. The incoming vectors are mapping by the radial basis functions in each hidden node. The output layer yields a vector $\mathbf{y} = [y_1, y_2, \cdots, y_m]$ for m outputs by linear combination of the outputs of the hidden nodes to produce the final output. Fig. 1 presents the structure of a single output RBF network; the network output can be obtained by

$$\mathbf{y} = f(\mathbf{x}) = \sum_{i=1}^{k} w_i \phi_i(\mathbf{x}) \tag{1}$$

where $f(\mathbf{x})$ is the final output, $\phi_i(\cdot)$ denotes the radial basis function of the *i*-th hidden node, w_i denotes the hidden-to-output weight corresponding to the *i*-th hidden node, and *k* is the total number of hidden nodes.

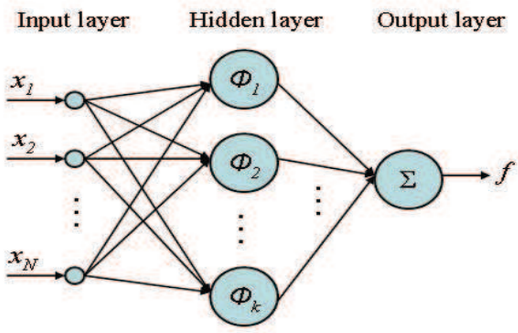


Figure 1. The structure of a RBFNN

A radial basis function is a multidimensional function that describes the distance between a given input vector and a pre-defined center vector. There are different types of radial basis function. A normalized Gaussian function usually used as the radial basis function, that is

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}-\mu_i\|^2}{2\sigma_i^2}\right) \tag{2}$$

where μ_i and σ_i denote the center and spread width of the *i*-th node, respectively.

Generally, the RBFNN training can be divided into two stages:

1. Determine the parameters of radial basis functions, i.e., Gaussian center and spread width. In general, k-means clustering method was commonly used here.
2. Determine the output weight w by supervised learning method. Usually Least-Mean-Square (LMS) or Recursive Least-Square (RLS) was used.

The first stage is very crucial, since the number and location of centers in the hidden layer will influence the performance of the RBFNN directly. In the next section, the principle and procedure of self-structure RBF algorithm will be described.

3. Self-structure RBFNN

The hidden layer of an RBFNN acts as a receptive field operating on the input data space. The number of hidden node based on the distribution of the training data set. The proposed approach performs this task by defining a cluster distance factor, ε, which is the maximum distance between an input sample and a specific RBF node center and allowing the number of basis function to increase iteratively according to this factor.

The rationale of this learning is described as follows: the hidden layer starts with no hidden node and ε is pre-determined by PSO to control the clusters production. The first RBF node center μ_1 is set by choosing one data, x_1, randomly from N_T input data sample. The value of Euclidean 2-norm distance between μ_1 and the next input sample, x_2, is compared with ε.

If it is greater, a new cluster whose center location is x_2 is created as μ_2; otherwise, the elements of μ_1 are updated as

$$\mu_{1i}(\text{new}) = \mu_{1i}(\text{old}) + \alpha\|x_{2i} - \mu_{1i}(\text{old})\|, i = 1,2,\cdots,N \tag{3}$$

where μ_{1i} and x_{2i} are the i-th component of vectors $\boldsymbol{\mu}_1$ and $\mathbf{x}_2$, respectively, $\|\cdot\|$ denotes the Euclidean distance and $0 < \alpha < 1$ is the updating ratio. Thus, this procedure is carried out on the remaining training samples. The number of clusters grows or RBF nodes center self-adjust continuously until all of the samples are processed. The proposed self-structure RBFNN algorithm can be summarized as follows:

1. Assuming that there are p clusters with their centers, $\mu_1,\cdots,\mu_p$, are generated from previous iterations. Taking a new input sample x_n to calculate the distances with the each clusters $\|x_n - \mu_i\|$, where $i = 1,\cdots,p$.
2. The cluster whose center μ_q is $\arg\min_{\mu_i}\left(\|x_n - \mu_i\|, \text{where } i = 1,...,p\right)$ will be focused.
3. Comparing $\|x_n - \mu_q\|$ with the distance criterion parameter, ε. If it is greater than ε, then a new cluster center, μ_{p+1}, is created at the position of the sample point, x_n. Otherwise the elements of μ_p are updated by (3).
4. Repeating the above steps until all of the samples are processed.

For L clusters, a global spread width σ can be derived by the average of Euclidean distance between each cluster center and its nearest neighbor as

$$\sigma = \left\langle \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\| \right\rangle \tag{4}$$

where $\langle\cdot\rangle$ denotes the expression for the average value for $1 \le i \le L$, $1 \le j \le L$ and $i \ne j$.

In (Yunfei & Zhang, 2002), the cluster distance factor, $\varepsilon \in (0,\infty)$, is obviously a critical factor to determine input space partitioning and obtains the hidden node number and locations in RBFNN. An unduly large value of ε does not reflect an enough number of cluster so it may cause a poor-generalized precision solution. On the contrary, an unduly small value of ε will create redundant clusters; therefore, it may cause overlap between RBF neurons; moreover, it may lead to poor accuracy and slow convergence either. This paper proposes a PSO-based searching approach to determine the proper value of ε; further, the optimal structure of RBF network can be obtained. And, an objective function to evaluate the effectiveness of applying PSO is proposed. Following section will describe how to employ PSO technique to search a potential optimal value ε.

4. PSO-based Self-structure RBFNN

The PSO is a population based optimization technique that was proposed by Kennedy and Eberhart in 1995 (Eberhart & Kennedy, 1995), which the population is referred to as a *swarm*. The particles express the ability of fast convergence to local and/or global optimal position(s) over a small number of generations.

4.1 Evolution of PSO

A swarm of PSO consists of a number of particles. Each particle represents a potential solution of the optimization task. All of the particles iteratively discover the probable solution. Each particle generates a position according to the new velocity and the previous positions of the particle, and it is compared with the best position which is generated by previous particles according to the cost function. The best solution is then kept; i.e., each particle accelerates in the directions of not only the local best solution but also the global best position. If a particle discovers a new probable solution, other particles will move closer to it so as to explore the region more completely in the process (Gudise & Venayagamoorthy, 2003).

Let N denotes the swarm numbers. In general, there are three attributes, current position a_{ij}, current velocity v_{ij} and past best position Pb_{ij}, for particles in the search space to present their features. Each particle in the swarm is iteratively updated according to the aforementioned attributes assuming that the objective function f is to be minimized so that the dimension consists of n particles and the new velocity of every particle is updated by (5).

$$\begin{aligned} v_{ij}(t+1) = w v_{ij}(t) + c_1 r_{1,i}(t)[Pb_{ij}(t) - a_{ij}(t)] \\ + c_2 r_{2,i}(t)[Gb_i(t) - a_{ij}(t)] \end{aligned} \tag{5}$$

where v_{ij} is the velocity of the j-th particle of the i-th swarm for all $i \in 1 \ldots N$, w is the inertia weight of velocity, c_1 and c_2 denote the *acceleration coefficients*, r_1 and r_2 are two uniform random values falling in the range between (0, 1), and t is the number of generations. The new position of the i-th particle is calculated as follows:

$$a_{ij}(t+1) = a_{ij}(t) + v_{ij}(t+1) \tag{6}$$

The past best solution of each particle is updated by:

$$Pb_i(t+1) = \begin{cases} Pb_i(t), & \text{if } f(a_i(t+1)) \geq f(Pb_i(t)) \\ a_i(t+1), & \text{otherwise} \end{cases} \tag{7}$$

The global best solution Gb will be found from all of particles during previous three steps are defined as:

$$Gb\ (t+1) = \arg\min_{Pb_i} f(Pb_i(t+1)),\ \ 1 \leq i \leq n \tag{8}$$

4.2 Disturbance

Since initial particles are generated by randomly, they may not uniform enough to distribute over the solution space. Therefore, it may trap particles into local optimal solution inevitably. To avoid solution falling into the local minimal and jumping it out to find the global minimal, this paper added a mutation-like disturbance strategy into the PSO process (Sun et al, 2005). The disturbance mechanism randomly activates under a disturbance probability. While the disturbance mechanism is active, the selected particle will be randomly placed at a new position (ε value in this paper), then this particle will keep following the PSO process to search a better solution. The other non-selected particle will keep following the PSO iteration as usual and trying to find a new solution.

4.3 Objective function

For searching a suitable ε value for RBFNN training, a function of root mean squared error (RMSE) which evaluates discrepancies between the sampling data output y_n and the predictive output y_n^* is applied. Thus, the objective function for N_T sample is defined as

$$f(\varepsilon, y_n^*) = \mathrm{RMSE}(y) = \sqrt{\frac{\sum_{k=1}^{N_T} \left(y_n(k) - y_n^*(k)\right)}{N_T}} \tag{9}$$

where $y_n^*(k)$ is the predictive output of the *k*-th sample data which is obtained by ε value during training.

In the section II, the relationship between self-structure RBF network training and cluster distance factor ε was discussed. If (9) can be reduced to a sufficiently small value, a suitable value of ε could be obtained to train the structure of RBFNN. Thus, the predictive RBFNN output would be closed to the sampling data output.

4.4 RBFNN structure determination by PSO

In this paper, our goal is to minimize the value of $f(\varepsilon, y_n^*)$. The objective function minimized by PSO and found potential optimal solution finally. Since we only search one parameter by PSO (i.e., the cluster distance factor ε), the swarm number *i*=1, and defined the particle number as $1 \le j \le m$. In the initial state of PSO, all the particles' positions a_j (i.e., initial cluster distance factor ε) were set as 0.02, v_j were set as 0, and the Pb_j and Gb_j were initialized by a random number generator in the range of [0, 1]. After particles moved by (6), each particle will find a potential solution, the new past best position would be updated by (7), and the global best position would be updated by (8). The particle would keep moving to find a better solution until it reaches the goal or meets the termination condition (Lin et al, 2005). The pseudo code of our PSO-based cluster distance factor searching approach presented in Fig. 2.

```
Create and initiate an N-dimension PSO: P
Repeat:
    Execute PSO to update P by (5) and (6)
    for each particle i ∈ [1...m]
                    if f(ε_ij , y_n*) < f(Pb_ij , y_n*)
            then Pb_ij = ε_ij
                    if f(Pb_ij , y_n*) < f(Gb_i , y_n*)
            then Gb_i = Pb_ij
    endfor
Until Termination condition is met
```

Figure 2. The pseudo code of PSO-based cluster distance factor searching

5. Simulations and Results

5.1 Setting of simulation

The six nonlinear functions with different complexities are tested here. These tested functions are listed as follows:

Ex.	Tested function	Range
1	$y=\frac{(x-2)(2x-1)}{(1+x^2)}$	$x\in[-8,12]$
2	$y=\frac{\sin(x)}{x}$	$x\in[-10,10]$
3	$y=1.1(1-x^2+2x^2)e^{-x^2/2}$	$x\in[-5,5]$
4	$y=0.5\sin(\frac{x}{2})+\frac{(1+\cos x)}{2}$	$x\in[-4.5,10.8]$
5	$y=\sin(\frac{2\pi x}{5})+\sin(\frac{2\pi x}{10})$	$x\in[0,10]$
6	$y=-\sin(\frac{\pi x}{10})+\cos(\frac{2\pi x}{5})$	$x\in[-10,10]$

Table 1. The six tested nonlinear functions

In order to confirm the advantages of the proposed approach, the K-means algorithm (Moddy & Darken, 1989) and GA-based self-growing RBFNN training algorithm (Yunfei & Zhang, 2002) are also carries out in these tested functions. Due to (Yunfei & Zhang, 2002) adopted Simple Genetic Algorithm (SGA) which using binary coding to train RBF structure for saving computation time, but it will loose some accuracy compared to the real-valued, i.e., this method may not present the optimal solution. So we implemented it with Real-value Genetic Algorithm (RGA) to obtain accuracy results.

For every simulation, the training data set consists of 50 input-output data samples taken at random, and the testing data set includes 75 samples different from the training data set. For the definition of parameters in the proposed approach, w, c_1 and c_2 are given 0.12, 0.25 and 0.25 respectively, and the search range of ε is bounded between 0.2 and 1, the particle number is 10. For the GA-based self-growing RBFNN training algorithm the search range of ε in the input space is also in the range from 0.2 to 1, the crossover rate P_c is given 0.8, and mutation rate P_m is given 0.01, the population size is 10. For the K-means method, the optimal number of RBF neurons in the hidden layer is chosen to be 30 by experience.

5.2 Simulation results

After simulations, the RMSE of training data, RMSE of testing data, maximal error and number of hidden node will be presented in tables for each case. In these tables, the three involved algorithms are denotes as PSO-based, GA-based (Yunfei & Zhang, 2002) and K-means (Moddy & Darken, 1989). Additionally, the real data and approximated data will be shown in the same figure; meantime, the error from each approximation will be presented by figures. There three sub-figures in each figure, the results from the left sub-figure to the

right sub-figure are generated by PSO-based approach, GA-based approach and K-means approach, respectively.

Example 1.

	PSO-based	GA-based	K-means
RMSE for training data	0.0332	0.0584	0.1552
RMSE for testing data	0.0520	0.0786	0.1962
Maximal error	0.3852	0.2355	0.9508
Number of hidden node	33	29	30

Table 2. Comparison between the three approaches in example 1

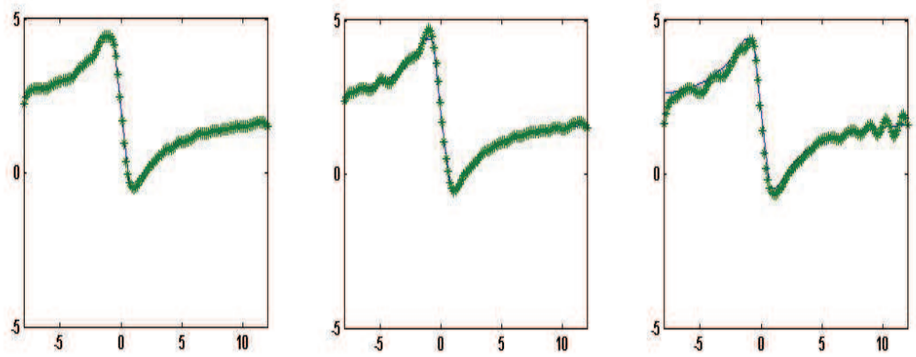


Figure 3. Curves of RBFNN output and real data in example 1. (solid-line represents the real data, dashed-line represents the output data)

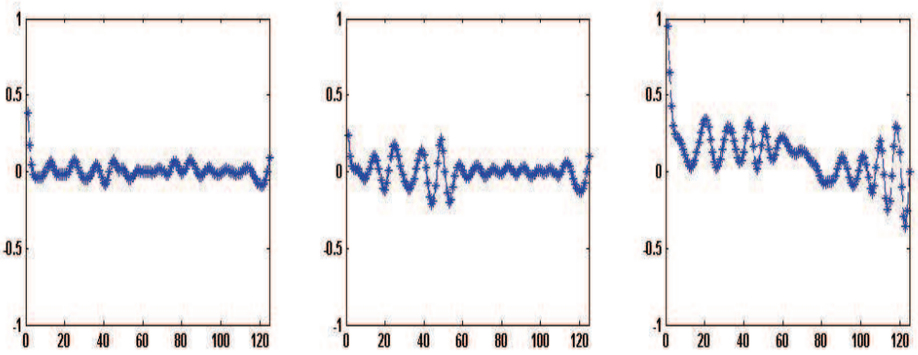


Figure 4. The errors between the real data and approximations in example 1

Example 2.

	PSO-based	GA-based	K-means
RMSE for training data	0.0035	0.0046	0.0234
RMSE for testing data	0.0099	0.0113	0.0357
Maximal error	0.0460	0.0509	0.1006
Number of hidden node	28	28	30

Table 3. Comparison between the three approaches in example 2

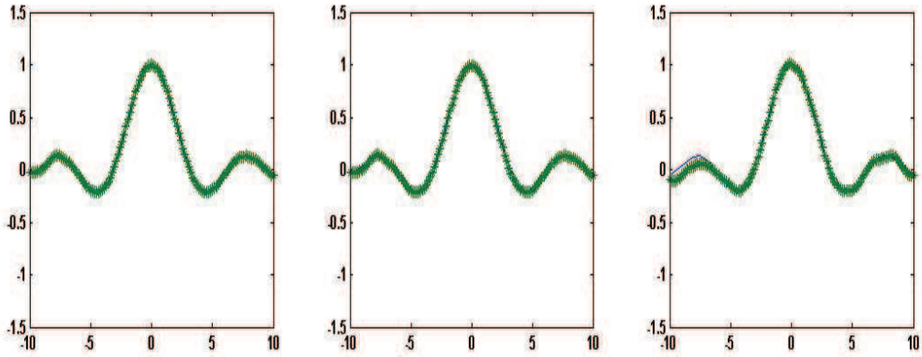


Figure 5. Curves of RBFNN output and real data in example 2. (solid-line represents the real data, dashed-line represents the output data)

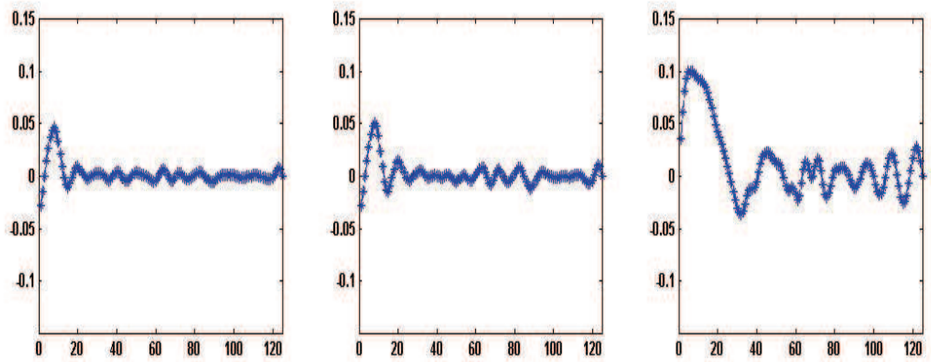


Figure 6. The errors between the real data and approximations in example 2

Example 3.

	PSO-based	GA-based	K-means
RMSE for training data	0.0056	0.0173	0.1271
RMSE for testing data	0.0057	0.0188	0.0803
Maximal error	0.0134	0.0432	0.2441
Number of hidden node	24	22	30

Table 4. Comparison between the three approaches in example 3

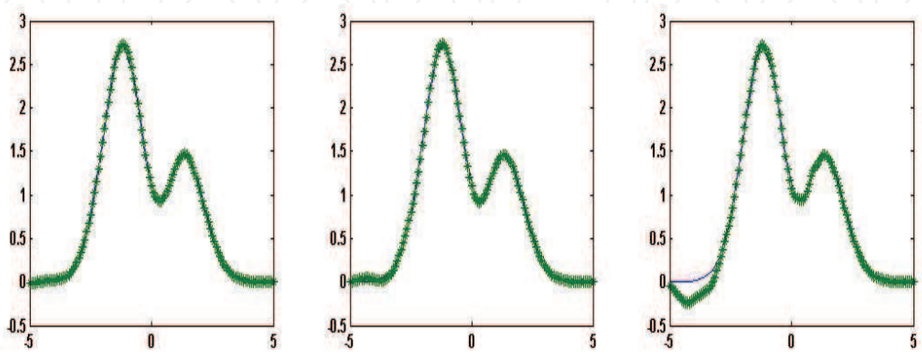


Figure 7. Curves of RBFNN output and real data in example 3. (solid-line represents the real data, dashed-line represents the output data)

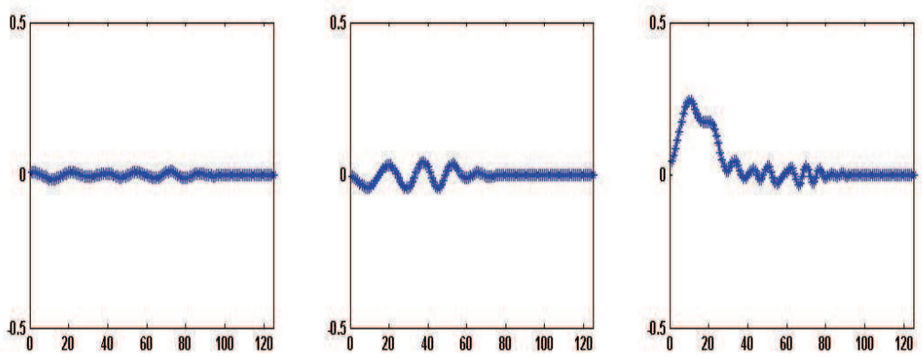


Figure 8. The errors between the real data and approximations in example 3

Example 4.

	PSO-based	GA-based	K-means
RMSE for training data	0.0079	0.0112	0.0337
RMSE for testing data	0.0192	0.0292	0.0740
Maximal error	0.1217	0.0939	0.1854
Number of hidden node	19	31	30

Table 5. Comparison between the three approaches in example 4

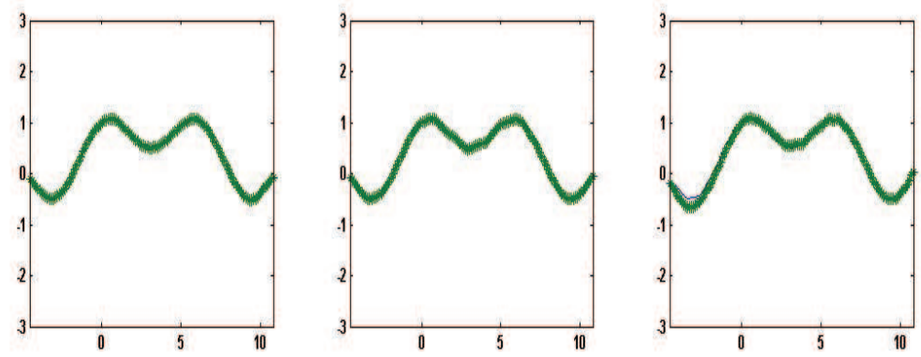


Figure 9. Curves of RBFNN output and real data in example 4. (solid-line represents the real data, dashed-line represents the output data)

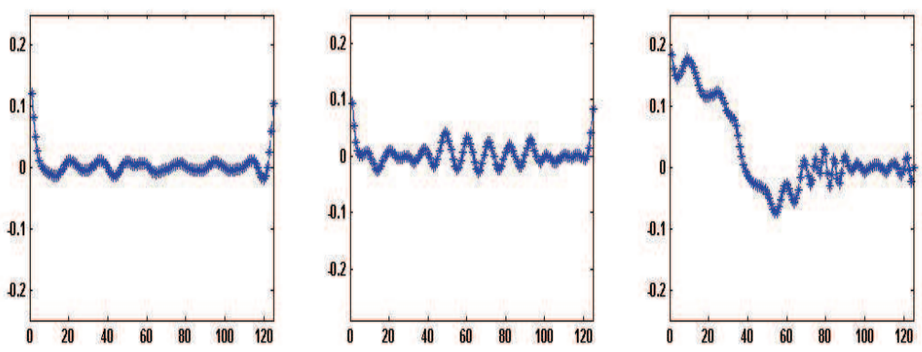


Figure 10. The errors between the real data and approximations in example 4

Example 5.

	PSO-based	GA-based	K-means
RMSE for training data	0.0460	0.0519	0.0637
RMSE for testing data	0.0546	0.0564	0.0867
Maximal error	0.3056	0.3236	0.2208
Number of hidden node	20	21	30

Table 6. Comparison between the three approaches in example 5

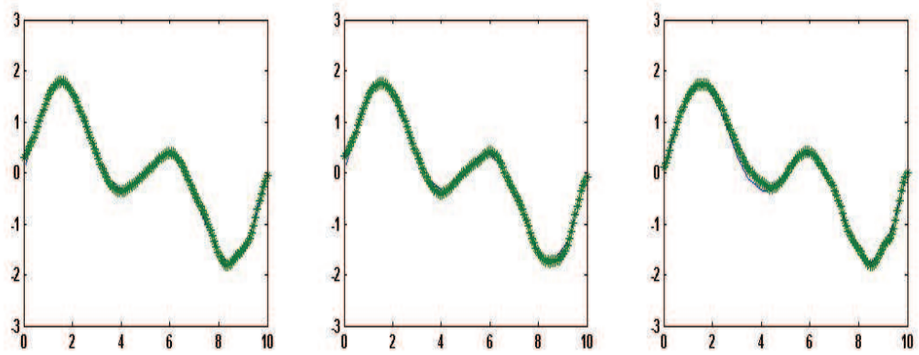


Figure 11. Curves of RBFNN output and real data in example 5. (solid-line represents the real data, dashed-line represents the output data)

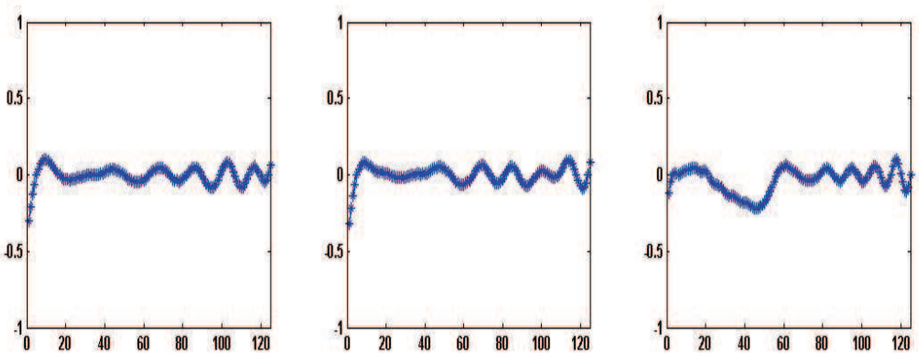


Figure 12. The errors between the real data and approximations in example 5

Example 6.

	PSO-based	GA-based	K-means
RMSE for training data	0.0079	0.0092	0.0690
RMSE for testing data	0.0439	0.0509	0.0859
Maximal error	0.2159	0.2486	0.1855
Number of hidden node	29	27	30

Table 7. Comparison between the three approaches in example 6

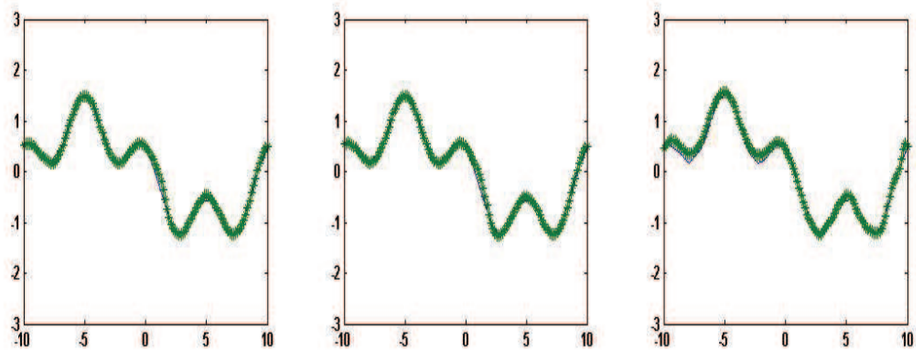


Figure 13. Curves of RBFNN output and real data in example 6. (solid-line represents the real data, dashed-line represents the output data)

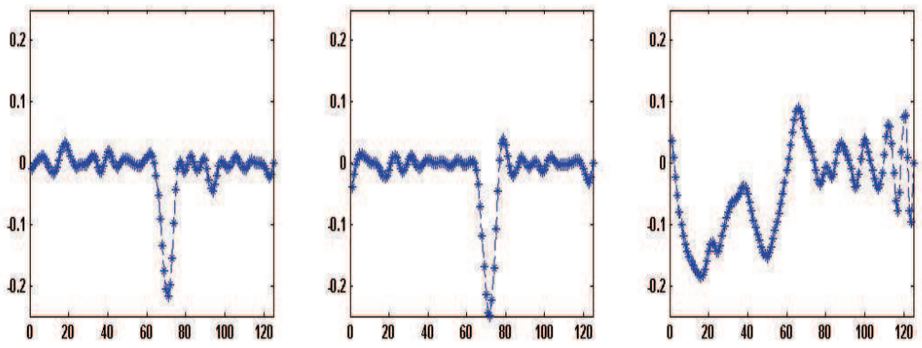


Figure 14. The errors between the real data and approximations in example 6

5.3 Discussion

In the simulation results in tables, PSO-based approach has lower RMSE for training data and testing data. It means that over fitting does not happen in the proposed approach. From figures of the curves of RBFNN output and real data, the approximated curves by PSO-based approach is closer to the real data than these by others. From figures of the approximated errors, it could be shown that PSO-based approach results small error in most of sample, whereas the K-means approach has largest error.

We know that RBFNN needs different number of hidden node and cluster radius for different complexities. K-means approach usually performs a larger error because it is not able to decide a suitable number of hidden node. Though GA-based approach decides a suitable number of hidden node, its cluster radius is not good enough to classify whole data. The proposed approach is able to find out the optimal cluster radius to further decide a number of hidden node because PSO has better capacity of global searching than GA.

6. Conclusion

This paper has presented a novel approach for self-structure RBFNN. A very important step for the RBFNN training is to decide a proper number of hidden node. If the number of hidden node does not chosen properly, the RBFNN may present poor global generalization capability, slow training speed, and the requirement of large memory space. Therefore, to

decide a suitable cluster distance factor (ε) is the crucial condition for creating an optimal self-structure RBFNN. This paper proposed a PSO-based approach for searching the optimal ε; further, RBFNN is able to determine the optimal number of hidden node automatically.
For proofing benefits of the proposed PSO-based approach, the simulations consisting of six nonlinear system modeling were tested; meanwhile, GA-based approach and K-means approach were also carried out for comparison. Simulation results show that the PSO-RBFNN algorithm outperforms the GA-RBFNN and K-means methods by the minimal training RMSE and the minimal testing RMSE.

7. References

Aiguo, S. & Jiren, L. (1998). Evolving Gaussian RBF network for nonlinear time series modeling and prediction, *IEEE Electronics Letters*, Vol. 34 (12), pp. 1241-1243

Broomhead, D. S. & Lowe, D. (1988). Multivariable functional interpolation and adaptive networks, *Complex Systems*, Vol. 2, pp. 321-355

Back, T.; Hammel, U. & Schwefel, H. P. (1997). Evolutionary computation: comments on the history and current state, *IEEE Trans. on Evolutionary Computation*, Vol. 1, pp. 3-17

Chen, S.; Wu, Y. & Luk, B. L. (1999). Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks, *IEEE Trans. on Neural Networks*, Vol. 10 (5), pp. 1239-1243

Chen, S. (1995). Nonlinear time series modeling and prediction using Gaussian RBF networks with enhances clustering and RLS learning, *Electronics Letters*, Vol. 31, No. 2, pp. 117-118

Eberhart, R. C. & Kennedy, J. (1995). A new optimizer using particle swarm theory, *Proceeding of 6th Int. Symp. Micro Machine and Human Science*, pp. 39-43

Freeman, J. A. S. & Saad, D. (1995). Learning and generalization in radial basis function networks, *Neural Computation*, Vo. 9 (7), pp. 1601-1622

Gudise, V. G. & Venayagamoorthy, G. K. (2003). Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks, *Proceeding of IEEE Swarm Intelligence Symposium*, pp. 110-117

Karayiannis, N.B & Mi, G.W. (1997). Growing radial basis neural networks : merging supervised and unsupervised learning with network growth techniques, *IEEE Trans. on Neural Networks*, Vol. 8 (6), pp. 1492-1506

Lin, C. L.; Hsieh, S. T.; Sun, T. Y. & Liu, C. C. (2005). PSO-based learning rate adjustment for blind source separation, *Proceeding of International Symposium on Intelligent Signal Processing and Communications Systems*, pp. 181-184

Lowe, D. (1989). Adaptive radial basis function nonlinearities, and the problem of generalization, *Procedings of IEE International Conference on Artificial Neural Networks*, pp. 171-175

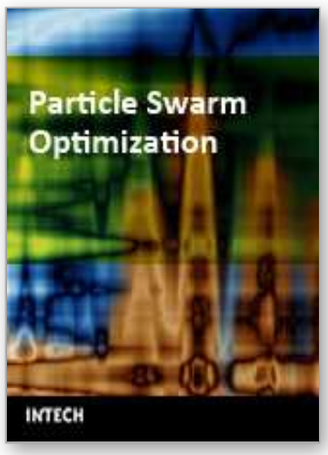
Moddy, Y. & Darken, C. J. (1989). Fast learning in network of locally tuned processing unites, *Neural computation*, Vol.1, pp. 281-294

Song, A. & Lu, J. (1988). Evolving Gaussian RBF network for nonlinear time series modeling and prediction, *Electronics Letters*, Vol. 34, No.12, pp. 1241-1243

Sun, T. Y.; Hsieh, S. T. & Lin, C. W. (2005). Particle Swarm Optimization Incorporated with Disturbance for Improving the Efficiency of Macrocell Overlap Removal and Placement, *Proceeding of The 2005 International Conference on Artificial Intelligence*, pp. 122-125

Yunfei, B. & Zhang, L. (2002). Genetic algorithm based self-growing training for RBF neural Network, *IEEE Neural Networks*, Vol. 1, pp. 840-845

Zheng, N.; Zhang, Z.; Shi, G. & Qiao, Y. (1999). Self-creating and adaptive learning of RBF networks: merging soft-completion clustering algorithm with network growth technique, *Proceeding of International Joint Conference on Neural Networks*, Vol. 2, pp. 1131-1135



Particle Swarm Optimization

Edited by Aleksandar Lazinica

ISBN 978-953-7619-48-0
Hard cover, 476 pages
Publisher InTech
Published online 01, January, 2009
Published in print edition January, 2009

Particle swarm optimization (PSO) is a population based stochastic optimization technique influenced by the social behavior of bird flocking or fish schooling.PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. This book represents the contributions of the top researchers in this field and will serve as a valuable tool for professionals in this interdisciplinary field.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen