Neural Comput & Applic (2010) 19:667–676
DOI 10.1007/s00521-009-0323-6

ORIGINAL ARTICLE

Research on an online self-organizing radial basis function neural network

Honggui Han · Qili Chen · Junfei Qiao

Received: 20 October 2008 / Accepted: 9 December 2009 / Published online: 9 January 2010

Abstract A new growing and pruning algorithm is proposed for radial basis function (RBF) neural network structure design in this paper, which is named as self-organizing RBF (SORBF). The structure of the RBF neural network is introduced in this paper first, and then the growing and pruning algorithm is used to design the structure of the RBF neural network automatically. The growing and pruning approach is based on the radius of the receptive field of the RBF nodes. Meanwhile, the parameters adjusting algorithms are proposed for the whole RBF neural network. The performance of the proposed method is evaluated through functions approximation and dynamic system identification. Then, the method is used to capture the biochemical oxygen demand (BOD) concentration in a wastewater treatment system. Experimental results show that the proposed method is efficient for network structure optimization, and it achieves better performance than some of the existing algorithms.

Keywords Self-organizing RBF neural network (SORBF) · Growing and pruning approach · BOD soft measurement

1 Introduction

Radial basis function (RBF) neural networks offer an efficient mechanism for approximating complex nonlinear functions [1], pattern recognition [2], modeling and controlling dynamic systems [3, 4] from the input–output data. In fact, the selection of RBF neural network for a special application is dependent on its structure and learning abilities. Recently, with the research objects becoming more and more complex, the conventional RBF neural network with the fixed structures can not satisfy the requirement. The most difficult bottlenecks are the initial number of the hidden nodes, the initial position, and width of the RBF nodes.

In order to solve the previously mentioned problems, some colleagues have found several kinds of methods. A significant contribution to adding nodes to the neural network is made by Platt [5] through the development of a resource allocation network (RAN), in which hidden nodes are added sequentially based on the novelty of the new data. Karayiannis et al. [6] propose a growing RBF neural network, in which a merging supervised and unsupervised learning algorithm is presented. The structure of the RBF neural network can be changed by the research objects, but this method does not consider the redundant nodes in the hidden layer. On the other side, Yingwei et al. [7] introduce a pruning strategy based on the relative contribution of each hidden neuron to the overall network output to reduce the complex of the RBF neural network. The resulting neural network is minimal in theory by the presented method. Other pruning methods for RBF neural networks have been proposed by Salmerón et al. [8] and Rojas et al. [9]. Unfortunately, one of the disadvantages of pruning algorithms is that the computational cost is heavy, since the majority of the training time is spent on networks larger than necessary. A promising alternative to growing or pruning alone is to combine them. Wu et al. [10] propose a hierarchical online self-organizing learning algorithm for dynamic RBF neural networks. However, all the widths of

H. Han · Q. Chen · J. Qiao (✉)
College of Electronic and Control Engineering,
Beijing University of Technology, Beijing, China
e-mail: adqiao@sina.com

H. Han
e-mail: Rechard112@emails.bjut.edu.cn

Q. Chen
e-mail: shuang3045@163.com

Gaussian membership functions are the same. This usually does not coincide with the reality, especially when input variables have significantly different operating intervals. Guang-Bin Huang et al. [11] propose a simple sequential learning algorithm for RBF neural network referred to growing and pruning algorithm for RBF (GAP-RBF). And in the later, they advance this GAP-RBF to GGAP-RBF [12]. Both GAP-RBF and GGAP-RBF neural networks decide to add new nodes or reduce the redundant nodes based on the “significance” of a neuron and links it to the learning accuracy. However, as these algorithms are online procedures, they do not optimize the network over all past training data. Besides, the network initialization requires a prior knowledge about the data which may not be available.

For the position and the width of the RBF nodes in the RBF neural network, there are some other papers which adjust the parameters of the position and the width in the learning process. Juan Ignacio et al. [13] analyze the bounded-ness of the coefficients involved in Gaussian expansion series. These series arise from the reconstruction of band-limited functions, applying the sampling theorem with Gaussians as the reconstruction filters. The width can be changed by the approximation errors and consequently to the accuracy of the estimation. Sheng Chen et al. [14] introduce a powerful symmetric RBF neural network classifier for nonlinear detection. By exploiting the inherent symmetry property of the optimal Bayesian detector, the proposed symmetric RBF neural network is capable of approaching the optimal classification performance using noisy training data. The RBF neural network construction process is robust to the choice of the RBF width and is computationally efficient. And the paper [15] shows an adaptive RBF neural network method to estimate two immeasurable physical parameters on-line and to compensate for the model uncertainty and engine time varying dynamics. The adaptive law of the neural network is based on the Lyapunov theory, so that this RBF neural network can adjust the width of the RBFs, and the stability of the network was guaranteed. In recent years, wavelet neural networks [16, 17] are considered to be better and simpler alternatives to adjust the position and width of the RBF nodes. In the wavelet RBF neural networks (WRBF), the activation function of hidden layer nodes is substituted with a type of wavelet functions. In these new networks, the position and dilation of the wavelet are fixed, and only the weights are optimized. Although there are other papers [18–22] which are about the position and width of the RBF nodes, few of them consider the optimal number of the hidden layer nodes of the RBF network simultaneity.

In this paper, a new online SORBF is proposed for the RBF structure design which can add the new RBF nodes and reduce the redundant RBF nodes in the hidden layer. In the structure design phase, the growing and pruning method is used to decide the satisfaction of the architecture of the neural network, which is based on the radius of the receptive field of the RBF nodes and the stable error of the requirement. At the same time, the position and width of the RBF nodes in the hidden layer are adjusted by the gradient-descend algorithm. For highlighting the effect of the SORBF, the performance of the SORBF is compared with other well-known RBF neural networks on some benchmark problems in the nonlinear functions approximation area and nonlinear dynamic system identification. And then, this SORBF is used to capture the key variables in a wastewater treatment system. The results indicate the SORBF can provide comparable generalization performance with less computational complexity.

This paper is organized as follows. Section 2 describes the details of RBF neural network. Section 3 presents the RBF nodes selection algorithm based on self-organizing method. The algorithms for the position and width of the RBF nodes adjusting are also given in this section. Experimental results of the simulations are presented in Sect. 4, in order to demonstrate the superior performances of this proposed SORBF algorithm, the results are compared with other self-organizing RBF algorithms. Section 5 summarizes the conclusions from this study.

2 Radial basis function (RBF) neural network

The standard radial basis function (RBF) neural network consists of three layers: an input layer, a hidden layer, and an output layer. Figure 1 shows a schematic representation of the RBF network. The number of the nodes in the input and output layers is decided by the research objects. The nodes in the input layer and output layer represent the vector from an input space and a desired network response, respectively. Through a defined learning algorithm, the error between the actual and desired response is minimized by optimization criterions.

As depicted in Fig. 1, the ith output node of the RBF network can be expressed as follows:

$$y_i = \sum_{k=1}^{N} \varphi_k(\|x - v_k\|)w_{ik}, i = 1, 2, \ldots, m \tag{1}$$

where $x = [x_1, x_2, \ldots, x_n]^T$ is an input value; n is the number of input node; c_k is the center of the kth RBF node in the hidden layer, $k = 1, 2, \ldots, N$, and N is the number of hidden nodes; $\|x - v_k\|$ denotes Euclidean distance between v_k and x; $\varphi_k(\bullet)$ is the nonlinear transfer function of the kth RBF node; w_{ik} is the weighting value between the kth RBF node and the ith output node; and m is the number of output nodes.

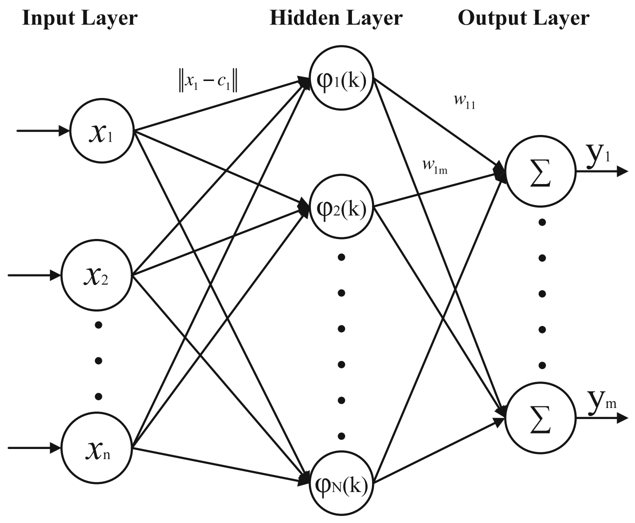

Fig. 1 Schematic representation of RBF neural network

Equation (1) reveals that the output of the network is computed as a weighted sum of the hidden layer outputs. The nonlinear output of the hidden layer is described as $\varphi_k(\bullet)$, which are radial symmetrical. In this paper, the function chosen for this neural network is Gaussian function, and the description is shown as follows:

$$\varphi(x) = e^{\left(-\frac{(x-v)^2}{\delta^2}\right)} \tag{2}$$

where v and δ are the parameter of position and width of the RBF nodes. Figure 2 shows the RBF nodes with different width and position. The activation function most commonly used for classification and regression problems in the Gaussian function, because it is continuous, differentiable; it provides a softer output and improves the interpolation capabilities. The procedure to design an RBF neural network for functional approximation problem is shown below:

1. Initialize number N of the RBF nodes;
2. Initialize position v of the RBF nodes;
3. Initialize the width δ of the RBF nodes;
4. Calculate the optimum value for the weights w;

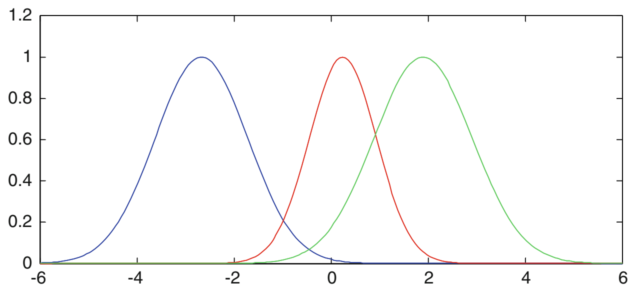

Fig. 2 RBF nodes with different widths and positions

5. Apply local search algorithm to adjust the widths and the positions of the RBF nodes.

Based on 1–3, the initializations of the number of the RBF nodes are very important, if an unsuitable initialization number of the RBF nodes is performed, the training time will be heavy, and the approximation error hard to be achieved. The reason is that during the execution of a local search algorithm to make a fine tuning of the positions and widths of the RBF nodes, the search algorithm is hard to offset the flaw of the architecture. So, a self-organizing algorithm is necessary for the RBF structure design. This algorithm can be used to add new RBF nodes and reduce redundancy RBF nodes, which solves the problem of the architecture. Meanwhile, the position and width of the RBF nodes are very important, if the position and width are not appropriate for the RBF nodes, there is a possibility of falling into a bad local minimum. When the RBF neural network selects the correct number of the RBF nodes, the parameters of the weights, the position and width of the RBF nodes will be adjusted at the same time.

3 Self-organizing RBF neural network

The key problems of the RBF neural network are the structure design and the parameters-learning approaches. The structure design mainly relies on the RBF nodes in the hidden layer. This SORBF algorithm can construct the RBF neural network automatically. The procedure to design the SORBF neural network for the application problems is shown below:

1. Initialize the number of RBF nodes N;
2. Initialize the position v and width δ of the RBF nodes;
3. Initialize the radius r of the receptive field for each RBF node;
4. Calculate the optimum number of the RBF nodes;
5. Apply local search algorithm to adjust the position v, the width δ and radius r;
6. Calculate the optimum value for the weights w.

This SORBF neural network can add or reduce the RBF nodes. It is a modification of the neural network structure design method with the aim of catching the suitable RBF neural network architecture.

3.1 RBF nodes selection method

The main steps of the RBF nodes selection method are shown as follows:

1. Initialization, $k = 1$; randomly weight value connecting the output layer with the hidden layer $\mathbf{W} = (w_{ij})_{N\times m}$, $0 \le w_{ij} \le 1$; the number of the nodes

in hidden layer are pre-given as N. The radius r of the receptive field for each RBF node is given initially as r_0, the expected error is E_d. The positions v of the RBF nodes are given randomly, and the width of the RBF nodes is given initially as δ_0; $i = 1, 2, \ldots, N$.

2. Find the winner node x in hidden layer, the winner node at time k can be ensured by:

$$\|v_x - P(k)\| = \min_{x=1,2,\ldots,N} \|v_x - P(k)\|. \quad (3)$$

And $\|\cdot\|$ is the Euclidean distance, $P(k)$ is the input rector at time k.

3. Adding new RBF nodes or reducing redundant RBF nodes.

If: $E(k) > \alpha E_d$, and $\|v_x - P(k)\| > \beta r_x$, go to step 4;
If: $E(k) \le \alpha E_d$, and $\|v_x - P(k)\| > \beta r_x$, go to step 6;
If: $E(k) \le \alpha E_d$, and $\|v_x - P(k)\| \le \beta r_x$, go to step 5;
If: $E(k) > \alpha E_d$, and $\|v_x - P(k)\| \le \beta r_x$, go to step 6.
where $1 < \alpha < 2$, and $1 < \beta < 2$.

4. Adding a new RBF node.

4.1 Finding out the nearest RBF node v_x connected with $P(k)$ by the formula (3) in the hidden layer.

4.2 Inserting a new RBF node r between the node x and $P(k)$, the positions and the width values of the new RBF node are as follows:

$$v_r = aP(k) + (1 - a)v_x.$$
$$\delta_r = \delta_0 \quad (4)$$

where, a is a plus constant less than 1.

4.3 Initializing radius r_r of the receptive field for the new RBF node r:

$$r_r = r_0. \quad (5)$$

5. Deleting a redundant RBF node.

5.1 Computing the distance d_x between the node v_x and the nearest node f. If $d_x < r_x$ and $d_x < r_f$, go to next; else, break and go to (6).

5.2 Deleting the node x or f (because the widths of these nodes are not the same, the final node left according to the value of the width).

If $r_f < r_x$, deleting the node v_f; else deleting the node v_x.

5.3 Modifying the radius value r_x of the receptive field for the retained node (we assume the left node is x).

$$r_x = br_x + (1 - b)r_f \quad (6)$$

where b is a plus constant more than 0.5 and less than 1.

6. Updating the positions and widths of the RBF nodes. Stopping until catching the training time or expected error E_d.

The RBF node selection method is essentially implemented to seek the suitable RBF nodes in the hidden layer, where an input acts as the principle to access the respective nodes containing the adjustable weight parameters to compute the activity of RBF nodes. The self-organizing algorithm searches the correct RBF node response to each input vector by choosing the minimum Euclidean distance. For each input, the SORBF can find a suitable RBF node or change the activity nodes based on it. The criterion of the RBF selection is based on the radius of the receptive field.

According to the former explanation of the RBF nodes selection, the structure of the initial RBF can be modified along with the learning process. The final architecture of the RBF neural network is suitable for the current objects. Following these steps, the SORBF will adjust the other parameters of the RBF neural network.

3.2 Adjusting radius of the receptive field for the RBF nodes

In this paper, the radius is used to judge whether the structure of the neural network should be reset or not. The radius of the receptive field can affect the capabilities of the final neural network. It is one of the essentials which relate to the results of the RBF neural network.

The radiuses of the receptive field adjust along with the learning process based on the winner times. The principles are described as follows (we take the node x for example):

$$r_x = \begin{cases} \varepsilon_x r_x & \text{if } x \text{ is a winner} \\ \tau_x r_x & \text{if } x \text{ is not a winner} \end{cases} \quad (7)$$

where ε_x, τ_x are the weight value-modifying parameters of node. In fact, ε_x is more than 1, and τ_x is less than 1. The real values of these two parameters are computed as:

$$\varepsilon_x = 1 + \frac{1}{\sqrt{2M_x}}.$$
$$\tau_x = 1 - \frac{1}{10\sqrt{M_x}}. \quad (8)$$

where M_x is the winner times of the RBF node v_x. In order to simplify the computing steps, we give ε_x as 1.01 and τ_x as 0.99. But the constant values may bring some problems. For example, the structures of the neural networks change acutely in the complex systems, if these two parameters are the constant values, the structures change slowly, finally, and they can hardly obtain the optimal networks.

3.3 Adjusting weights, positions, and widths

This RBF type function has three parameters: the position v; the width δ of the RBF nodes; and the weights w. In fact, these three parameters relate to the final capabilities of the RBF neural networks directly.

In order to train a neural network, the parameter-adjusting algorithm is based on the mean squared error (MSE) which is defined as:

$$E(t) = \frac{1}{N_s}\sum_{t=1}^{N_s}(y_d(t) - y(t))^2 \tag{9}$$

where N_s is the total number of the samples, $y_d(t)$ is the expected output of the t step and $y(t)$ is the neural network output of the t step. The goal of this method is to reach $E(t) < E_d$ by learning. E_d is the expected stable error. The details of the adjusting process are as follows:

1. The weights w;

$$w_{ij}(t+1) = w_{ij}(t) - \eta_1 \frac{\partial E(t)}{\partial w_{ij}(t)}. \\ i = 1,2,\ldots,N; j = 1,2,\ldots,m. \tag{10}$$

where η_1 is a plus constant, and it is less than 1.

2. The width δ of the RBF nodes;

$$\delta_i(t+1) = \delta_i(t) - \eta_2 \frac{\partial E(t)}{\partial \delta_i(t)}. \quad i = 1,2,\ldots,N. \tag{11}$$

where η_2 is a plus constant, and it is less than 1.

3. The position v of the RBF nodes;

In Sect. 3.1, the position v of the new inserting nodes has been discussed, and the position v of the other RBF nodes will be discussed here.

$$v_i(t+1) = \begin{cases} v_i(t) - \eta_3(P(k) - v_i(t)) & \text{if } v_i \text{ is the winner} \\ v_i(t) - \eta_4 \frac{\partial E(t)}{\partial v_i(t)} & \text{others} \end{cases} \\ i = 1,2,\ldots,N. \tag{12}$$

where η_3, η_4 are the plus constants, and they are less than 1.

3.4 Computational complexity and memory requirements

In this SORBF algorithm, the structure has to be updated after pruning or addition of a neuron. This requires updating the RBF nodes using (5). The time required is, therefore, of the order $O(N_sTM^2)$, and the memory required is of the order $O(N_sM)$. Where N_s is the number of training date, T is the number of times the structure needs to be updated by adding or pruning a neuron, and $M = N \times (n+1)$, in which n is the number of input variables, and N is the number of RBF nodes. In order to show the good performances of SORBF, the computational complexity and memory requirements of SORBF are compared with other dynamic adaptation methods such as DFNN [10] and GAP-RBF [11]. The DFNN used the linear least square (LLS) algorithm in batch mode, the time required in the approach based on LLS algorithm was of the order $O(N_s^2M^2)$, and the memory required is of the order $O(N_s^2M)$. Usually, $T \ll N_s$. Besides, GAP-RBF needs only one-step (times and division) operation for pruning checking and a simple sparse matrix operation for parameter adjustment at each step by using the EKF (extended Kalman filter), since only the nearest neuron is checked for significance. The time required is, therefore, of the order $O(N_sTM^2)$, and the memory required is of the order $O(N_sM)$. So, the computational complexity and memory requirements of GAP-RBF are the same as SORBF, and the time required for DFNN is more than that of SORBF.

4 Simulations

To demonstrate the effectiveness of the proposed algorithm, there are three examples analyzed in this paper. They are functions approximation, nonlinear systems identification, and the nonlinear dynamic systems modeling. In order to show the performances of this SORBF algorithm, the results of the approximation and the identification are compared with other algorithms such as DFNN [10] and GAP-RBF [11].

4.1 Function approximation

A common function which was widely used to demonstrate the effect of the algorithms [11] is used in this paper:

$$y = 0.8 \times e^{-0.2x} \times \sin(10x). \tag{13}$$

For each trial, the size of training samples is 200, and the size of testing samples is 200, the values of x are randomly distributed in the interval [0, 2].

The real output at step k is $y(k)$, the required value at time k is $y_d(k)$, the error at step k will be $y(k) = y_d(k) - y(k)$. The goal training MSE is 0.01, the initial radius of every hidden node is 0.1, the initial weight of every hidden node is randomly given in the interval [0, 1], and the initial RBF nodes of SORBF are 2. The parameters a and b are given as 0.8 and 0.9 in this paper. The simulation results are shown as follows.

Figure 3 shows the approximating results by the SORBF, the result demonstrates SORBF can approximate this function exactly. Figure 4 shows the error values in the approximating process within 1,000 steps; Fig. 5 shows the dynamic number of the nodes in the training process; these

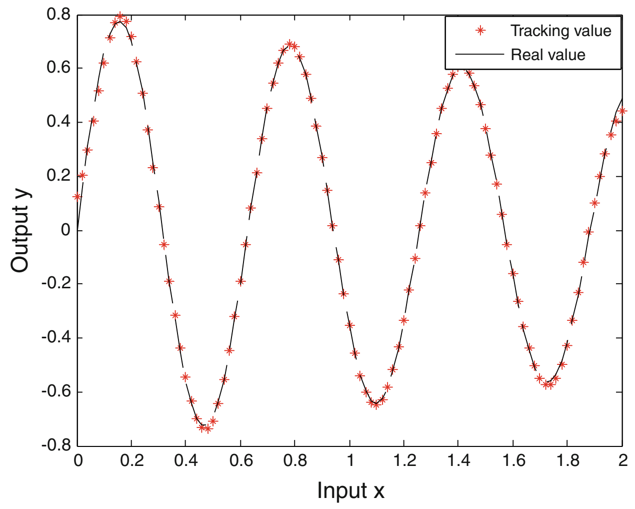

Fig. 3 The results of function approximation

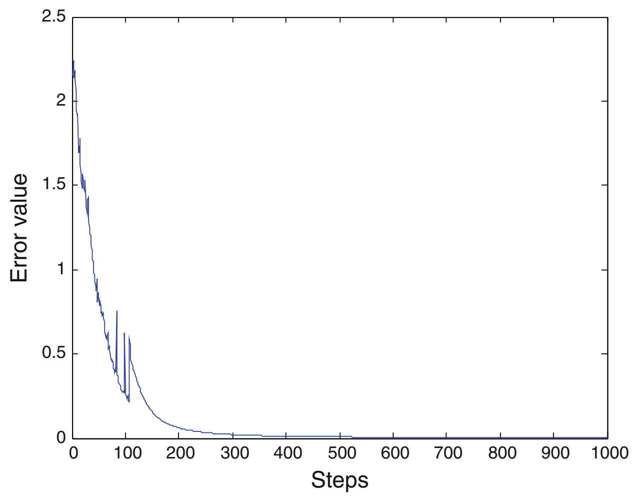

Fig. 4 The error results in the training process

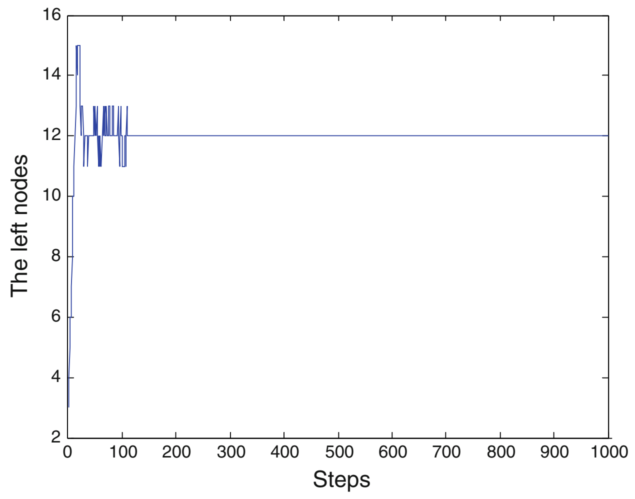

Fig. 5 The hidden nodes in the training process

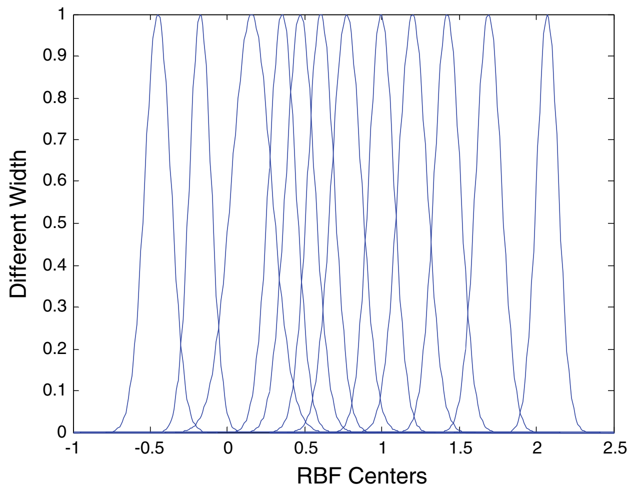

Fig. 6 RBF nodes with different widths and positions after training

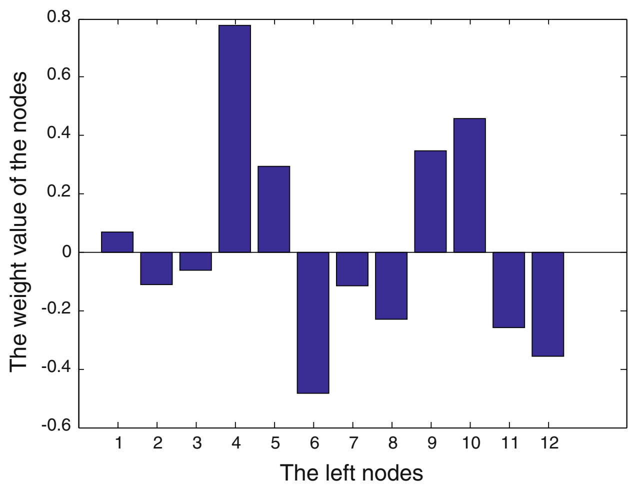

Fig. 7 The weight values of the left nodes after training

two figures describe when the nodes add or eliminate in the hidden layer, the error value will shake; and finally, the error can catch the goal quickly. Figure 6 shows RBF nodes with different width and position after training; Fig. 7 shows weight values of the RBF nodes after training. Some averaged values as measures of the performances are selected for this algorithm of this example by the 20 independent runs: the CPU running time of the training process, the test MSE, and the number of preserved RBF nodes in the hidden layer. The detail results compared with other different algorithms are given in Table 1. Based on the results, this proposed SORBF owns better performances than DFNN and GAP-RBF. The results prove that this proposed SORBF demonstrates a good performance for this function approximation. It should be noticed that this SORBF algorithm used for this functions approximation is insensitive to the initial structure of the neural network.

Table 1 The performances comparison with different algorithms

Algorithm	CPU Time (s)	Training error	Testing error	No. of nodes
DFNN	62.32	0.01	0.0861	25
GAP-RBF	26.86	0.01	0.0415	19
SORBF	22.67	0.01	0.0248	12

The final structure of this SORBF is simpler; the memory space is fewer because of the simple structure.

4.2 Nonlinear dynamic system identification

The plant is given as follows. It was used in [10] and some other papers to demonstrate the effect of the algorithms:

$$y(t+1) = \frac{y(t)y(t-1)[y(t)+2.5]}{1+y^2(t)+y^2(t-1)} + u(t). \tag{14}$$

$$t \in [1, 100], y(0) = 0, y(1) = 0, u(t) = \sin\left(\frac{2\pi t}{25}\right).$$

A set of 100 input-target data is chosen as training data. The different initializations for the two cases are as follows: Case B.1, the initial number of RBF nodes is 2. Case B.2, the initial number of RBF nodes is 30. The training MSE for this example is 0.01. The other initializations of SORBF, DFNN, and GAP-RBF are the same as *example A*. This model is identified in series–parallel mode, as given below:

$$\hat{y}(t+1) = f(y(t), y(t-1), u(t)). \tag{15}$$

There are three inputs ($y(t)$, $y(t+1)$, $u(t)$) and one output $y(t+1)$ in the networks. Another 100 input-target data in the interval [301, 400] are chosen as the testing data. The results are shown as Figs. 8, 9, 10, and 11 (Case B.1). Some averaged values as measures of the performances are selected for this algorithm of the two cases by the 20 independent runs: the CPU running time of the training process, the test MSE, and the number of preserved RBF nodes in the hidden layer.

As shown in Fig. 10, the SORBF organizes its structure with seven nodes after training (Case B.1). This is an important issue for the practical applications, and the structure is not changed after about 500 steps. The widths and the other parameters are adjusted continuously until achieving the required error. Note that in the learning process, the nodes may be added when the error is large, or the convergence speed is slow. This phenomenon is the same as the self-organizing strategy. Figure 9 shows the error value in the learning process within 1,000 steps (Case B.1). This illuminates the learning performance of the SORBF. The detail results compared with other different

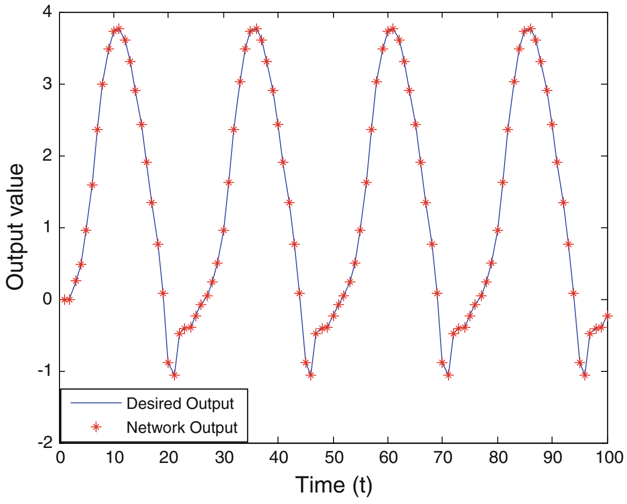

Fig. 8 The training results (Case B.1)

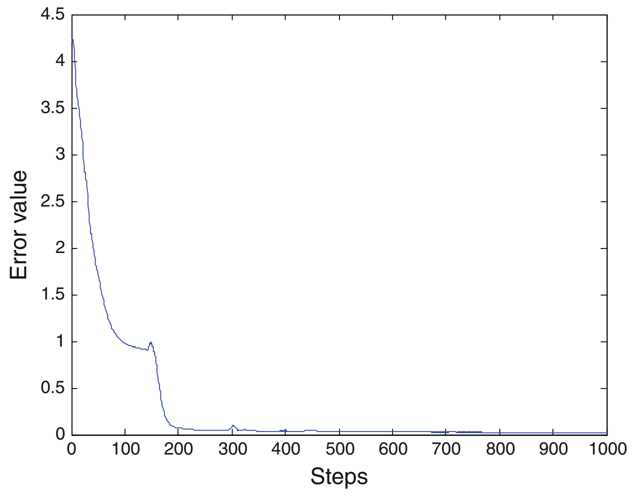

Fig. 9 The error value in the training process (Case B.1)

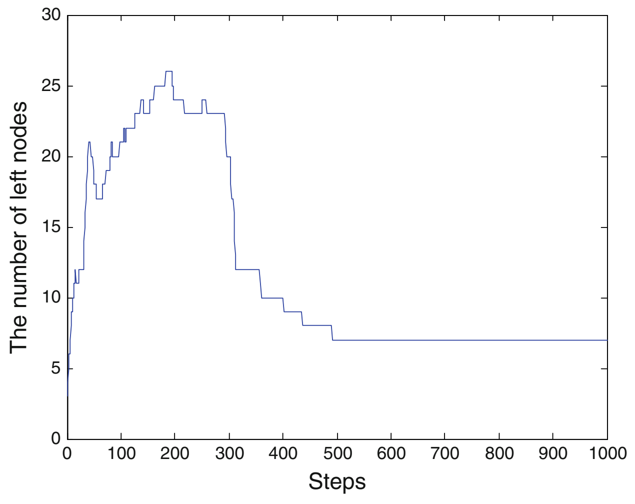

Fig. 10 The number of the nodes in the training process (Case B.1)

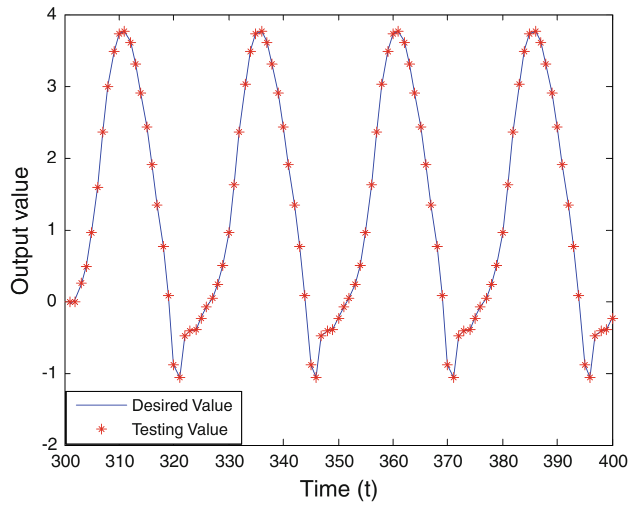

Fig. 11 The testing results (Case B.1)

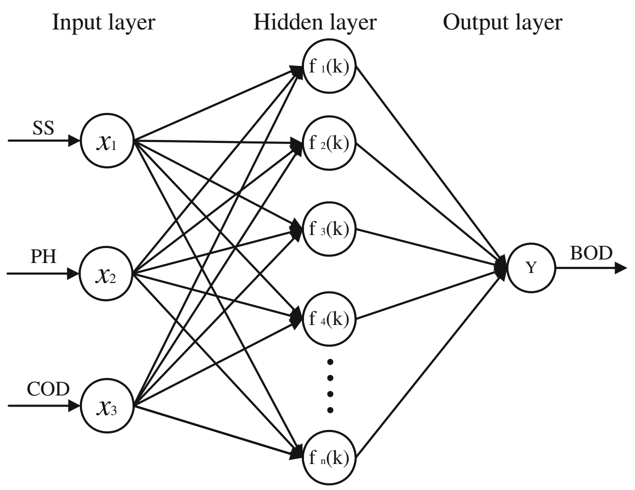

Fig. 12 The structure of soft measurement technique for BOD

algorithms are given in Table 2. The results prove that the SORBF demonstrates a good performance in this example.

4.3 Soft measurement for BOD

Rapid, accurate, and reliable measurements of BOD are a very desirable basis for monitoring or controlling wastewater treatment. Unfortunately, producing satisfactory BOD using hardware instrumentation has proved to be difficult. This paper addresses the issue of BOD estimation using a model-based approach which is relied on SORBF. The model estimates the BOD of the settled sewage using suspend solids (SS), chemical oxygen demand (COD), and PH data. The model is straightforward to apply online and offer a method of estimating BOD in real-time that is likely to be cheaper, more reliable and easier to maintain than hardware instrumentation. The structure of soft measurement technique based on SORBF is given as Fig. 12.

The SS, COD, and PH data are used as inputs for the model to estimate the settled sewage BOD. The initial structure of the neural network is 3-2-1; the samples used are from a wastewater treatment plant of Beijing in 2006. Hundred groups are for training, and another 100 groups are for validating. The accuracy of the model estimate was quantitated using the absolute error Ee which is defined as follows:

$$Ee(t) = |y_d(t) - y(t)| \tag{16}$$

where $y_d(t)$ is the tth sample of settled sewage BOD data and $y(t)$ is model estimate of the tth sample of settled sewage BOD data. The results are shown Figs. 13, 14, 15, 16, 17, and 18.

Figure 13 shows the results of BOD after training; Fig. 14 describes the absolute error value of the modeling results after training; the absolute errors are less than 1 mg/L which are highly accurate. Based on the results, SORBF is demonstrated to be suitable for the BOD soft measurement online. Figure 15 shows the dynamic RBF nodes within 1,000 steps; and Fig. 16 describes the *MSE* error value of the training process, these results also illuminate the good performances of the SORBF. Figure 17 shows the predictions results of BOD; Fig. 18 shows the absolute error value of the predicting process. The results have demonstrated that the BOD trends in the settled sewage could be predicted with acceptable accuracy using only SS, COD, and PH data as model inputs. This SORBF-based approach is relatively straightforward to implement online, it could offer real-time predictions of BOD, whereas hardware instruments typically measure

Table 2 The performance comparison of different algorithms

Approaches	Case B.1: with 2 initial hidden nodes			Case B.2: with 30 initial hidden nodes		
	CPU time (min)	Testing MSE	No. of RBF nodes	CPU time (min)	Testing MSE	No. of RBF nodes
DFNN	2.554	0.0264	15	3.864	0.0164	16
GAP-RBF	0.823	0.0212	12	1.223	0.0112	13
SORBF	0.521	0.0112	7	0.987	0.0112	8

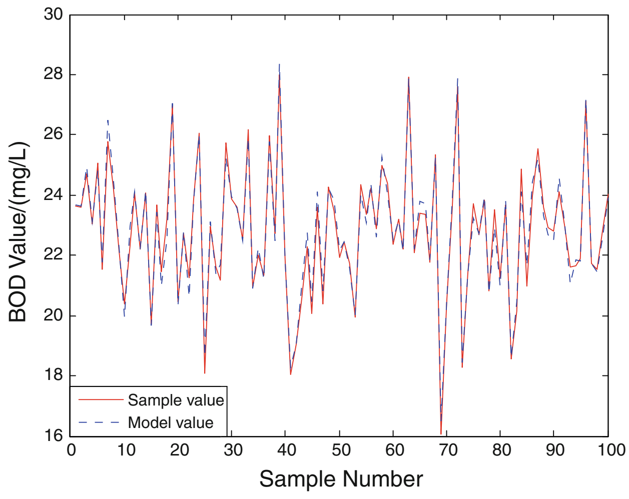

Fig. 13 The training process results of BOD

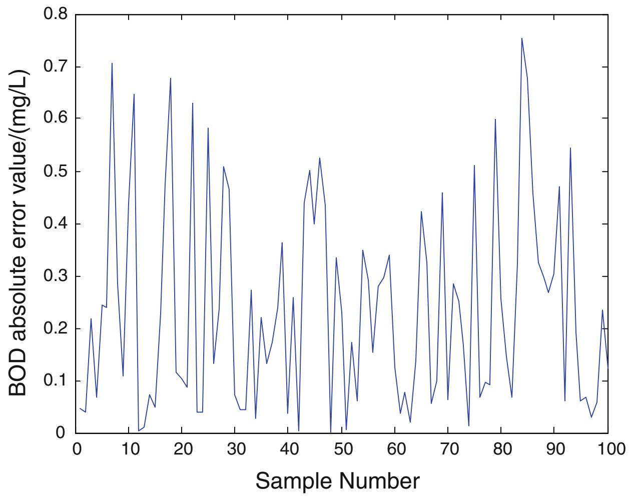

Fig. 14 The absolute error *Ee* value of the trained results

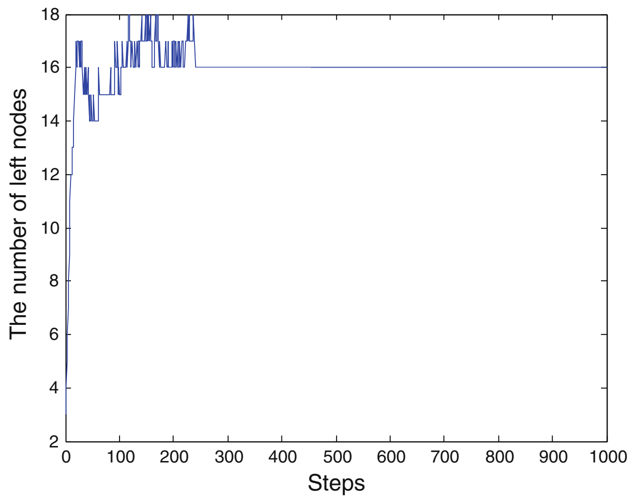

Fig. 15 The number of hidden nodes in the training process

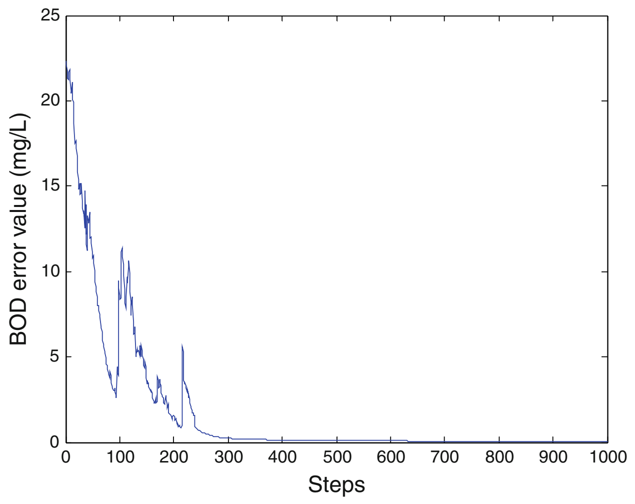

Fig. 16 The *MSE* error value of the training process

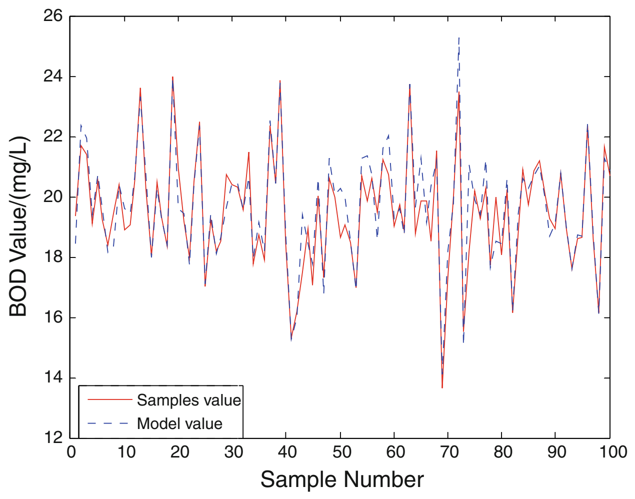

Fig. 17 The predictions results of BOD

short-term BOD. It can be concluded that this is a significant feature, since BOD is the more commonly used and readily understood measure. Finally, this type of SORBF-based approach has the potential to be used in estimating a range of variables which are typically troublesome to measure using hardware. This suggests that such new algorithm can be relatively a cost-effective approach for measuring and other useful applications.

5 Conclusions and discussion

A new self-organizing RBF (SORBF) algorithm is proposed in this paper to design as well as train RBF. Neither the number of nodes in the hidden layer nor the parameters need to be predefined and fixed. They are adjusted automatically in the learning process. And then, two examples

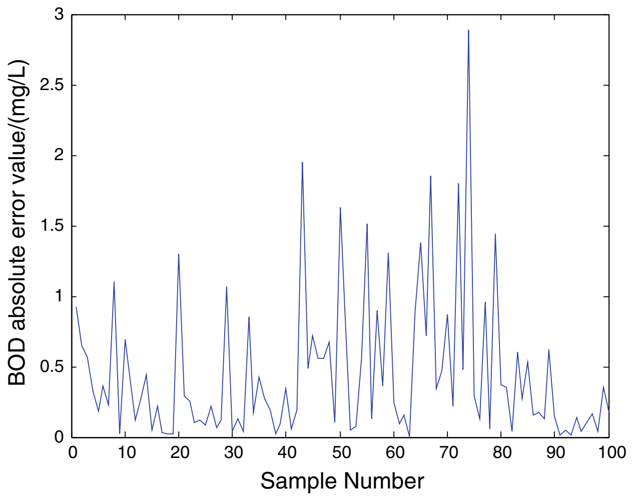

Fig. 18 The absolute error *Ee* value of the predictions results

are used to demonstrate the effect of this algorithm in contrast to DFNN and GAP-RBF. The results of these two examples prove that this SORBF performs better than the other algorithms. Finally, this SORBF is used to measure and predict the BOD value online. This type of SORBF-based approach offers a promisingly inexpensive approach to real-time measurement of variables that have typically proved difficult to measure reliably using hardware.

Acknowledgments The authors would like to thank Prof. Guo-Qiang Bi and Prof. Huan-you Zhang for reading the manuscript and providing valuable comments. And, we would like to express our deepest gratitude to all anonymous reviewers and the editor-in-chief for their valuable comments. This work was supported by the National 863 Scheme Foundation of China under Grant 2007AA04Z160 and 2007A-A04Z160, National Science Foundation of China under Grants 60674066 and 60873043, Ph.D. Program Foundation from Ministry of Chinese Education under Grant 200800050004, and Beijing Municipal Natural Science Foundation under Grant 4092010.

References

1. Nam MD, Thanh TC (2003) Approximation of function and its derivatives using radial basis function networks. Appl Math Model 27(3):197–220
2. Sing JK, Basu DK, Nasipuri M, Kundu M (2007) Face recognition using point symmetry distance-based RBF network. Appl Soft Comput 7(1):58–70
3. Zhao T (2008) RBFN-based decentralized adaptive control of a class of large-scale non-affine nonlinear systems. Neural Comput Appl 17(4):357–364
4. Ram D, Srivastava L, Pandit M, Sharma J (2007) Corrective action planning using RBF neural network. Appl Soft Comput 7(3):1055–1063
5. Platt J (1991) A resource-allocating network for function interpolation. Neural Comput 3(2):213–225
6. Karayiannis NB, Mi GW (1997) Growing radial basis neural networks: merging supervised and unsupervised learning with network growth techniques. IEEE Trans Neural Netw 8(6):1492–1506
7. Yingwei L, Sundararajan N, Saratchandran P (1997) A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks. Neural Comput 9(2):461–478
8. Salmerón M, Ortega J, Puntonet CG, Prieto A, Rojas I (2002) SSA, SVD, QR-cp, and RBF model reduction. In Lecture notes in computer science, vol 2415. Springer, Germany, pp 589–594
9. Rojas I, Pomares H, Bernier JL, Ortega J, Pino B, Pelayo FJ, Prieto A (2002) Time series analysis using normalized PG-RBF network with regression weights. Neurocomputing 42:267–285
10. Wu S, Er MJ, Gao Y (2002) A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks. IEEE Trans Fuzzy Syst 9(4):578–584
11. Huang G-B, Saratchandran P, Sundararajan N (2004) An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks. IEEE Trans Syst Man Cybern B 34(6):2284–2292
12. Huang G-B, Saratchandran P, Sundararajan N (2005) A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. IEEE Trans Neural Netw 16(1):57–67
13. Mulero-Martinez JI (2007) Boundedness of the nominal coefficients in Gaussian RBF neural networks. Neurocomputing 71(1–3):197–220
14. Chen S, Wolfgang A, Harris CJ, Hanzo L (2008) Symmetric RBF classifier for nonlinear detection in multiple-antenna-aided systems. IEEE Trans Neural Netw 19(5):737–745
15. Wang S, Yu DL (2008) Adaptive RBF network for parameter estimation and stable air–fuel ratio control. Neural Netw 21(1):102–112
16. Jin N, Liu D (2008) Wavelet basis function neural networks for sequential learning. IEEE Trans Neural Netw 19(3):535–540
17. Gholizadeh S, Salajegheh E, Torkzadeh P (2008) Structural optimization with frequency constraints by genetic algorithm using wavelet radial basis function neural network. J Sound Vib 312(1–2):316–331
18. Suetake N, Uchino E (2007) An RBFN–Wiener hybrid filters using higher order signal statistics. Appl Soft Comput 7(3):915–922
19. Falcao AO, Langlois T, Wichert A (2007) Flexible kernels for RBF networks. Neurocomputing 69(16–18):2356–2359
20. Sun YF, Liang YC, Zhang WL, Lee HP, Lin WZ, Cao LJ (2005) Optimal partition algorithm of the RBF neural network and its application to financial time series forecasting. Neural Comput Appl 14(1):36–44
21. Guillen A, Pomares H, Rojas I, Gonzalez J, Herrera LJ, Rojas F, Valenzuela O (2008) Studying possibility in a clustering algorithm for RBFNN design for function approximation. Neural Comput Appl 17(1):75–89
22. Er MJ, Wu S (2002) A fast learning algorithm for parsimonious fuzzy neural systems. Fuzzy Sets Syst 126(3):337–351