A Hybrid Forward Algorithm for RBF Neural Network Construction

Jian-Xun Peng, Kang Li, *Member, IEEE*, and De-Shuang Huang, *Senior Member, IEEE*

***Abstract*—This paper proposes a novel hybrid forward algorithm (HFA) for the construction of radial basis function (RBF) neural networks with tunable nodes. The main objective is to efficiently and effectively produce a parsimonious RBF neural network that generalizes well. In this study, it is achieved through simultaneous network structure determination and parameter optimization on the continuous parameter space. This is a mixed integer hard problem and the proposed HFA tackles this problem using an integrated analytic framework, leading to significantly improved network performance and reduced memory usage for the network construction. The computational complexity analysis confirms the efficiency of the proposed algorithm, and the simulation results demonstrate its effectiveness.**

***Index Terms*—Analytic framework, computational complexity analysis, parameter optimization, radial basis function (RBF) neural network, structure determination.**

I. Introduction

DUE to the simple topological structure and universal approximation ability [33], radial basis function (RBF) neural networks have been widely used in many areas, such as data mining, pattern recognition, signal processing, time series prediction, and nonlinear system modeling and control. See, for example, [1], [7], [12], [13], [15], [18], [19], [23], [25], [31], [35], and [40]–[42]. Gaussian radial basis functions have also been widely used in support vector machines, an important class of machine learning algorithms [5], [37]. One of the most important issues in the RBF neural network applications is the network learning, i.e., to optimize the adjustable parameters, which include the center vectors, the variances (or the widths of the basis functions), and the linear output weights connecting the RBF hidden nodes to the output nodes. Another important issue is to determine the network structure or the number of RBF nodes based on the parsimonious principle [8], [20], [22], [26].

Both the issues to determine the network size and to adjust the parameters on the continuous parameter space are closely coupled. It is a mixed integer hard problem if the two issues are considered simultaneously. Evolutionary algorithms have been used to address this problem [15], [22], however, they are computationally very expensive to implement and it is also well known that these algorithms suffer the slow and premature convergence problems. Despite that no analytic method is available to efficiently and effectively address this integrated problem, the two separate issues have been studied extensively in the literature.

With respect to the RBF neural network learning, conventional approach takes a two-stage procedure, i.e., unsupervised learning of both the centers and widths for the RBF nodes and supervised learning of the linear output weights. With respect to the center location, clustering techniques have been proposed [38], [39]. For the width learning, if the input samples are uniformly distributed, an identical width can be set for all the basis functions, otherwise a particular width has to be set for each individual basis function to reflect the input distribution [33]. Once the centers and the widths are determined, the linear output weights can be obtainable using Cholesky factorization, orthogonal least squares, or singular value decomposition [9].

In contrast to the conventional two-stage learning procedure, supervised learning methods aim to optimize all the network parameters [21], [27], [29]. To improve the convergence, various techniques have been introduced. For example, hybrid algorithms combine the gradient-based search for the nonlinear parameters (the widths and centers) of the RBF nodes and the least squares estimation of the linear output weights [28], [32], [34]. Second-order algorithms have also been proposed, which use an additional adaptive momentum term to the Levenberg–Marquardt algorithm in order to maintain the conjugacy between successive minimization directions, resulting in good convergence for some well-known hard problems [4]. In [29], the performances of three different RBF learning methods are compared—a gradient-based algorithm (gradient descent with a momentum term), a three-step hybrid learning algorithm, and a genetic algorithm. Generally speaking, although supervised learning is thought to be superior to conventional two-stage approaches, it can be computationally more demanding.

Nevertheless, these above learning methods are only applicable to RBF networks of fixed structure. If the network size also has to be determined, one of the simplest ways is to repeat these above learning procedures with different network size until the optimal one is acquired based on some network selection criterion such as the Akaike information criterion (AIC) [3]. This method is, however, computationally too demanding.

With respect to the determination of the RBF neural network structure, a popular approach is to formulate it as a linear-in-the-parameters problem, where all the training patterns/samples are usually used as the candidate RBF centers, and the RBF widths are chosen *a priori*. A parsimonious network is then determined from these candidates using an efficient forward subset selection

Manuscript received August 2, 2005; revised May 3, 2006. The work of K. Li was supported by EPSRC, U.K., under Grant GR/S85191/01.

J.-X. Peng and K. Li are with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Belfast BT9 5AH, U.K. (e-mail: k.li@qub.ac.uk).

D.-S. Huang is with the Institute of Intelligent Machines, Chinese Academy of Sciences, Anhui 230031, China.

Digital Object Identifier 10.1109/TNN.2006.880860

method, such as the orthogonal least squares (OLS) algorithms [9], [14], [42] or the forward recursive algorithm (FRA) [24]. To improve the network generalization, the regularized forward selection (RFS) algorithm has been proposed [30], which combines subset selection with zero-order regularization. Backward selection methods have also been used in RBF center selection [16], [17]. However, forward selection algorithms are thought to be superior to backward methods in terms of computational efficiency, since they do not need to solve the equations explicitly with the full set of initial candidate centers.

Generally speaking, existing (forward and backward) subset selection methods have several major disadvantages. First, since the RBF widths are set *a priori* and the centers of the RBF nodes are selected from a set of training samples with limited size, the optimal values on the continuous parameter space for the center and width parameters of RBF nodes can be easily missed out on. This means that the stepwise forward or backward procedures can easily miss out on a good RBF neural network. Second, in order to increase the chance of obtaining a satisfactory RBF network, one has to use a very large set of candidate RBF nodes of different centers and widths. This is, however, sometimes computationally too expensive or impossible to implement, since all the candidate RBF nodes have to be stored for batch operations and the number of all candidates will increase exponentially as the search space dimension increases. Part of this is usually referred to as the curse of dimensionality problem in the literature.

In order to optimize the RBF center and width parameters along with the network structure determination process, a sparse incremental regression (SIR) modeling method was proposed very recently [10]. This method appends regressors in an incremental modeling process. For each regressor to be appended, the nonlinear parameters are tuned using a boosting search based on a correlation criterion. In this way, the network structure and the associated nonlinear parameters are determined simultaneously. However, the search for the optimal values of the nonlinear parameters (RBF centers and widths) is a continuous optimization problem. The boosting approach in SIR, which employs a stochastic search process, tends to be slower in convergence than calculus-based optimization techniques. In addition, all the nonlinear parameters are treated equally in SIR, and the difference between the center and width parameters of a RBF node is ignored; this is again another factor that slows down the search process. Finally, the boosting search in SIR has three to five parameters that need to be tuned empirically. All the above potential problems with SIR are illustrated in the simulation examples at the end of this paper.

Different from existing methods in RBF neural network construction, this paper proposes a novel hybrid forward algorithm (HFA), which performs simultaneous network growing and parameter optimization within an integrated analytic framework, leading to two main technical advantages. First, the network performance can be significantly improved through the optimization of the nonlinear RBF parameters on the continuous parameter space. Second, conventional forward selection algorithms tend to use all training samples to produce a very large set of candidate RBF nodes from which the final RBF network is selected [6], [9]. The currently proposed method, however, only uses a very small number of training samples just for the initialization of the RBF centers, aiming to speed up the continuous optimization procedure. As a result, the memory requirement is significantly reduced.

This paper is organized as follows. Section II gives the problem formulation for the forward RBF network construction, and the corresponding forward RBF network construction procedure is introduced in Section III. Section IV proposes the hybrid RBF network construction algorithm, and Section V gives the computational complexity analysis for the proposed algorithm. Application examples are given in Section VI. Section VII concludes this paper.

II. Problem Formulation for Forward RBF Neural Network Construction

A wide class of multiple-input–single-output systems can be modelled by the RBF neural networks given by

$$\hat{y} = \text{RBF}\,(\mathbf{x}, \sigma, \mathbf{c}, \mathbf{w}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}, \sigma_i, \mathbf{c}_i) \tag{1}$$

where $\hat{y}$ denotes the network output, $\mathbf{x} \in \Re^n$ is the input vector to the network, $\phi_i(\mathbf{x}, \sigma_i, \mathbf{c}_i)$ denotes the radial basis function (e.g., Gaussian basis function) of the ith hidden node with the center $\mathbf{c}_i \in \Re^n$ and the width $\sigma_i \in \Re^1$, and w_i is the linear output weight. The adjustable parameters in network (1) are therefore the center vector $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_m)$, the width vector $\sigma = (\sigma_1, \ldots, \sigma_m)$, and the linear output weight vector $\mathbf{w} = (w_1, \ldots, w_m)^{\mathrm{T}}$. For the Gaussian radial basis function $\phi_i(\mathbf{x}, \sigma_i, \mathbf{c}_i) = \exp(-\|\mathbf{x} - \mathbf{c}_i\|^2/\sigma_i^2)$, where $\| \bullet \|$ denotes the Euclidean norm.

Now suppose a set of training samples (patterns), denoted as $\{(\mathbf{x}(\kappa), \mathrm{y}(\kappa)), \kappa = 1, \ldots, N\}$, is used for the network training, and the desired network output series is $\mathbf{y} = [y(1), \ldots, y(N)]^{\mathrm{T}}$. The network training aims to optimize $\mathbf{c}, \sigma$ and $\mathbf{w}$ such that the sum squared error (SSE) becomes minimal

$$Q(\sigma, \mathbf{c}, \mathbf{w}) = \sum_{\kappa=1}^{N} [y(\kappa) - \text{RBF}\,(\mathbf{x}(\kappa), \sigma, \mathbf{c}, \mathbf{w})]^2 \rightarrow \min. \tag{2}$$

Let $\phi_i = [\phi_i(\mathbf{x}(1)), \ldots, \phi_i(\mathbf{x}(N))]^{\mathrm{T}} \in \Re^N$, where $\phi_i(\kappa) = \phi_i(\mathbf{x}(\kappa), \sigma_i, \mathbf{c}_i)$, be the output vector of the ith RBF node when all data samples are fed into the network. It is referred to as the basis vector in the following context. Let $\Phi = [\phi_1, \ldots, \phi_m] \in \Re^{N \times m}$. Then the least square estimate of the linear output weights is given by

$$\mathbf{w} = (\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}\mathbf{y} \tag{3}$$

if Φ is of full column rank.

If (3) is used as a constraint, the cost function (2) is a function of $\mathbf{c}$ and σ, i.e.,

$$Q(\sigma, \mathbf{c}) = \mathbf{e}^{\mathrm{T}}\mathbf{e} = (\mathbf{y} - \hat{\mathbf{y}})^{\mathrm{T}}(\mathbf{y} - \hat{\mathbf{y}}). \tag{4}$$

The objective of network training is then to minimize $Q(\sigma, \mathbf{c})$ defined in (4) against continuous parameter $\mathbf{c}$ and σ, subject to constraint (3). However, the network size m (i.e., the number of RBF nodes) is unknown. A practical solution is to use part or all data samples as the centers with the widths of the radial

basis functions being set *a priori* [7], [18], producing a large number of candidate RBF nodes, say, M nodes (this network is sometimes called the saturated full neural network). Then some subset selection algorithm is used to select a small set of RBF nodes based on a network selection criterion [7], [24], [34], leading to a neural network of small size.

In the forward subset selection, the M candidate basis vectors form the full regression matrix $\Phi = [\phi_1, \ldots, \phi_M], \Phi \in \Re^{N\times M}$. Now suppose k RBF basis vectors out of the M candidates, denoted as $\phi_1, \ldots, \phi_k$, have been selected, and the remaining candidates in Φ are denoted as $\phi_{k+1}, \ldots, \phi_M$. The least squares estimation of the network output weight vector is

$$\mathbf{w} = \left(\Phi_k^{\mathrm{T}}\Phi_k\right)^{-1}\Phi_k^{\mathrm{T}}\mathbf{y} \tag{5}$$

where $\Phi_k = [\phi_1, \ldots, \phi_k]$.

The cost function expressed in (4) is then given by

$$\begin{aligned} Q(\Phi_k) &= (\mathbf{y} - \Phi_k\mathbf{w})^{\mathrm{T}}(\mathbf{y} - \Phi_k\mathbf{w}) \\ &= \mathbf{y}^{\mathrm{T}}\left[\mathbf{I} - \Phi_k\left(\Phi_k^{\mathrm{T}}\Phi_k\right)^{-1}\Phi_k^{\mathrm{T}}\right]\mathbf{y}. \end{aligned} \tag{6}$$

If a new RBF basis vector $\forall\phi \in \{\phi_{k+1}, \ldots, \phi_M\}$ is selected into the neural network, the selected regression matrix increases by one column, becoming $\Phi_{k+1} = [\Phi_k, \phi]$. The cost function SSE is updated as

$$Q(\Phi_{k+1}) = \mathbf{y}^{\mathrm{T}}\left[\mathbf{I} - \Phi_{k+1}\left(\Phi_{k+1}^{\mathrm{T}}\Phi_{k+1}\right)^{-1}\Phi_{k+1}^{\mathrm{T}}\right]\mathbf{y} \tag{7}$$

and the net reduction in the cost function value due to adding ϕ into the network is given by

$$\Delta Q_{k+1}(\phi) = Q(\Phi_k) - Q([\Phi_k, \phi]) \tag{8}$$

which is referred to as the contribution of ϕ or the $(k+1)$th RBF node in this paper.

In order to select the $(k+1)$th RBF basis vector into the neural network, the contribution (8) has to be computed for each of the $M-k$ remaining candidates, i.e., $\forall\phi \in \{\phi_i, i = k+1, \ldots, M\}$. The one, say, ϕ_j, that gives the maximum contribution will be selected into the neural network as the $(k+1)$th RBF basis vector. The difference between various forward methods lies in how to compute the contribution (8) more efficiently [7], [9], [24].

In the following section, a forward RBF construction algorithm [24] will be briefly introduced as the starting point for the derivation of the hybrid forward algorithm to be given in Section IV.

III. A Forward Network Construction Procedure

First define a matrix series

$$\mathbf{R}_k \underline{\underline{\Delta}} \begin{cases} \mathbf{I} - \Phi_k(\Phi_k^{\mathrm{T}}\Phi_k)^{-1}\Phi_k^{\mathrm{T}}, & 0 < k \le M \\ \mathbf{I}, & k = 0. \end{cases} \tag{9}$$

Then, the cost function (6) with k RBF nodes is rewritten as

$$Q(\Phi_k) = \mathbf{y}^{\mathrm{T}}\mathbf{R}_k\mathbf{y}. \tag{10}$$

To efficiently compute the net contribution (8), two theorems about the matrix series $\{\mathbf{R}_k, k = 0, 1, \ldots, M\}$ defined in (9) have to be proposed.

Theorem 1: Let $\{\phi_1, \ldots, \phi_M\}$ be a set of $N\times 1$ column vectors and $M \le N$, and suppose matrices $\Phi_k = [\phi_1, \ldots, \phi_k], k = 1, \ldots, M$ are of full column rank. Then, the following property holds:

$$\mathbf{R}_{k+1} = \mathbf{R}_k - \frac{\mathbf{R}_k\phi_{k+1}\phi_{k+1}^{\mathrm{T}}\mathbf{R}_k^{\mathrm{T}}}{\phi_{k+1}^{\mathrm{T}}\mathbf{R}_k\phi_{k+1}}, \qquad k = 0, 1, \ldots, M-1. \tag{11}$$

Theorem 2: Let $\{\phi_1, \ldots, \phi_M\}$ be a set of linearly independent $N\times 1$ column vectors. All matrices $\mathbf{R}_k, k = 0, 1, \ldots, M$, $M \le N$, defined in (9) have the following properties:

$$\mathbf{R}_k^{\mathrm{T}} = \mathbf{R}_k \quad \mathbf{R}_k\mathbf{R}_k = \mathbf{R}_k, \qquad k = 0, 1, \ldots, M \tag{12}$$

$$\mathbf{R}_i\mathbf{R}_j = \mathbf{R}_j\mathbf{R}_i = \mathbf{R}_i, \qquad i \ge j, \quad i, j = 0, 1, \ldots, M \tag{13}$$

$$\mathbf{R}_k\phi = \begin{cases} \mathbf{0}, & \mathrm{rank}([\phi_1, \ldots, \phi_k, \phi]) = k \\ \phi^{(k)} \ne \mathbf{0}, & \mathrm{rank}([\phi_1, \ldots, \phi_k, \phi]) = k+1, \end{cases} \quad k = 0, 1, \ldots, M. \tag{14}$$

The proofs of Theorems 1 and 2 are given in the Appendix. In (14) and hereafter, for any $N\times 1$ column vector, say, an RBF basis vector ϕ and the desired output vector $\mathbf{y}$, denote

$$\phi^{(k)} \underline{\underline{\Delta}} \mathbf{R}_k\phi, \ \ \mathbf{y}^{(k)} \underline{\underline{\Delta}} \mathbf{R}_k\mathbf{y}. \tag{15}$$

Obviously, $\phi^{(0)} = \phi$ and $\mathbf{y}^{(0)} = \mathbf{y}$.

According to (12) and (13), for two arbitrary $N\times 1$ column vectors ϕ and p, it holds that

$$\begin{aligned} \phi^{\mathrm{T}}\mathbf{R}_k\mathbf{p} &= \left(\phi^{(k)}\right)^{\mathrm{T}}\mathbf{p} = \phi^{\mathrm{T}}\mathbf{p}^{(k)} \\ &= \phi^{\mathrm{T}}\mathbf{R}_k^{\mathrm{T}}\mathbf{R}_k\mathbf{p} = \left(\phi^{(k)}\right)^{\mathrm{T}}\mathbf{p}^{(k)}. \end{aligned} \tag{16}$$

From (10) and (11), the net contribution of ϕ given in (8) can then be computed as

$$\Delta Q_{k+1}(\phi) = \frac{\mathbf{y}^{\mathrm{T}}\mathbf{R}_k\phi\phi^{\mathrm{T}}\mathbf{R}_k^{\mathrm{T}}\mathbf{y}}{\phi^{\mathrm{T}}\mathbf{R}_k\phi} = \frac{[(\mathbf{y}^{(k)})^{\mathrm{T}}\phi^{(k)}]^2}{(\phi^{(k)})^{\mathrm{T}}\phi^{(k)}}. \tag{17}$$

According to Theorems 1 and 2, $\phi^{(k)}$ and $\mathbf{y}^{(k)}$ can be recursively updated, giving the final form

$$\left.\begin{aligned} \phi^{(k)} = \mathbf{R}_k\phi &= \left(\mathbf{R}_{k-1} - \frac{\mathbf{R}_{k-1}\phi_k\phi_k^{\mathrm{T}}\mathbf{R}_{k-1}^{\mathrm{T}}}{\phi_k^{\mathrm{T}}\mathbf{R}_{k-1}\phi_k}\right)\phi \\ &= \phi^{(k-1)} - \frac{\left(\phi_k^{(k-1)}\right)^{\mathrm{T}}\phi^{(k-1)}}{\left(\phi_k^{(k-1)}\right)^{\mathrm{T}}\phi_k^{(k-1)}}\phi_k^{(k-1)} \\ \mathbf{y}^{(k)} = \mathbf{R}_k\mathbf{y} &= \mathbf{y}^{(k-1)} - \frac{\left(\phi_k^{(k-1)}\right)^{\mathrm{T}}\mathbf{y}^{(k-1)}}{\left(\phi_k^{(k-1)}\right)^{\mathrm{T}}\phi_k^{(k-1)}}\phi_k^{(k-1)} \end{aligned}\right\} \tag{18}$$

where ϕ_k is the kth selected RBF basis vector.

Based on (17) and (18), a fast stepwise forward RBF neural network construction algorithm can be easily proposed (see [24]

for details). Despite the great efficiency of stepwise forward methods for RBF neural network construction, they have several technical disadvantages. First, in the forward network construction process, all candidate basis vectors have to be processed, stored, and updated each time a new node is added, which is computationally very demanding if the set of candidate basis vectors is very large in size. Second, the basis vectors are only selected from a candidate set $\{\phi_i, i = k+1, \ldots, M\}$, where M is a limited number, and the neural network performance is severely restricted. In detail, each basis vector can be regarded as a function of the width and the center in the continuous parameter space $\{\sigma, \mathbf{c}\}$, which is defined as follows:

$$\phi(\kappa, \sigma, \mathbf{c}) = \phi(\mathbf{x}(\kappa), \sigma, \mathbf{c}), \qquad \sigma \in \Re, \qquad \mathbf{c} \in \Re^n. \quad (19)$$

If all the adjustable parameters for an RBF node are grouped as a vector

$$\omega = [\omega_0, \omega_1, \ldots, \omega_n] = [\sigma, \mathbf{c}] \quad (20)$$

then the basis function (19) can be rewritten as

$$\phi(\kappa, \omega) = \phi(\mathbf{x}(\kappa), \omega), \qquad \omega \in \Re^{n+1}. \quad (21)$$

With the previous notations, one can easily conclude that the subset selection algorithms essentially select the parameter vector ω for each basis vector from a discrete candidate set, instead of searching in the continuous parameter space $\omega \in \Re^{n+1}$. Obviously, further improvement can be made if all the adjustable parameters are optimized on the continuous parameter space. This analysis forms the basis of the proposed HFA for the RBF neural network construction.

In the proposed algorithm, the RBF neural network is constructed using stepwise forward method, i.e., the RBF nodes are added into the network one by one. In the meantime, a gradient-based search for the optimal parameters $\omega \in \Re^{n+1}$ is carried out on the continuous parameter space to maximize the contribution of each selected basis function. In order to speed up the search process, a small number of candidate RBF basis vectors, which are randomly selected from the training samples, are used to initialize the gradient-based search. This above procedure results in a hybrid algorithm for the RBF neural network construction which will be fully explored in the following section.

IV. Hybrid Forward Algorithm for RBF Neural Network Construction

Given N training samples, the contribution $\Delta Q_{k+1}(\phi)$ of the $(k+1)$th newly added basis vector given in (17) is a function of parameter vector ω, and it can be rewritten as

$$\Delta Q_{k+1}(\omega) = C^2(\omega)/D(\omega) \quad (22)$$

with

$$\left.\begin{aligned} C(\omega) &= \mathbf{y}^{\mathrm{T}}\mathbf{R}_k\phi(\omega) = \left(\mathbf{y}^{(k)}\right)^{\mathrm{T}}\phi^{(k)}(\omega) \\ &= \textstyle\sum_{\kappa=1}^{N}\left[y^{(k)}(\kappa)\phi^{(k)}(\mathbf{x}(\kappa), \omega)\right] \\ D(\omega) &= [\phi(\omega)]^{\mathrm{T}}\mathbf{R}_k\phi(\omega) = \left[\phi^{(k)}(\omega)\right]^{\mathrm{T}}\phi^{(k)}(\omega) \\ &= \textstyle\sum_{\kappa=1}^{N}\left[\phi^{(k)}(\mathbf{x}(\kappa), \omega)\right]^2 \end{aligned}\right\} \quad (23)$$

where

$$\mathbf{R}_k\phi(\omega) = \phi^{(k)}(\omega) = \left[\phi^{(k)}(\mathbf{x}(1), \omega), \ldots, \phi^{(k)}(\mathbf{x}(N), \omega)\right]^{\mathrm{T}}. \quad (24)$$

The maximum contribution is expressed as

$$\Delta Q_{k+1}(\omega_{k+1}) = \max\{\Delta Q_{k+1}(\omega), \omega \in \Re^{n+1}\}. \quad (25)$$

The adjustable parameter vector ω_{k+1} has to be optimized for the $(k+1)$th RBF basis vector. This is an $(n+1)$-dimensional continuous optimization problem, and various optimization techniques are available. In this paper, the Newton method [36] is used. In order to facilitate the Newton search, the $(n+1)$-by-1 gradient vector and the $(n+1)$-by-$(n+1)$ Hessian matrix have to be derived.

To obtain the gradient vector of the contribution (22), the first-order partial derivatives of vector in (24) with respect to the adjustable parameters ω_i are denoted as

$$\left.\begin{aligned} \phi^{(k)}_{\omega_i}(\omega) &\triangleq \frac{\partial\phi^{(k)}(\omega)}{\partial\omega_i} = \mathbf{R}_k\phi_{\omega_i}(\omega) \\ \phi_{\omega_i}(\omega) &= [\phi_{\omega_i}(\mathbf{x}(1), \omega), \ldots, \phi_{\omega_i}(\mathbf{x}(N), \omega)]^{\mathrm{T}} \end{aligned}\right\} \quad i = 0, 1, \ldots, n. \quad (26)$$

The gradient vector $\mathbf{g}(\omega)$ of the contribution (22) with respect to ω is then given by

$$\left.\begin{aligned} \mathbf{g}(\omega) &= [g_i(\omega)]_{(n+1)\times 1} \\ g_i(\omega) &= \frac{\partial\Delta Q_{k+1}(\omega)}{\partial\omega_i} \\ &= 2r(\omega)[\mathbf{y} - r(\omega)\phi(\omega)]^{\mathrm{T}}\mathbf{R}_k\phi_{\omega_i}(\omega) \end{aligned}\right\} \quad (27)$$

where

$$r(\omega) \triangleq C(\omega)/D(\omega). \quad (28)$$

The Hessian matrix $\mathbf{H}(\omega)$ of (22) is derived as

$$\left.\begin{aligned} \mathbf{H}(\omega) &= [h_{i,j}(\omega)]_{(n+1)\times(n+1)} \\ h_{i,j}(\omega) &= h_{j,i}(\omega) = \frac{\partial^2\Delta Q_{k+1}(\omega)}{\partial\omega_i\partial\omega_j} \\ h_{i,j}(\omega) &= \frac{2\beta_i\beta_j}{D(\omega)} - 2r^2(\omega)\phi^{\mathrm{T}}_{\omega_j}(\omega)\mathbf{R}_k\phi_{\omega_i}(\omega) \\ &\quad + 2r(\omega)[\mathbf{y} - r(\omega)\phi(\omega)]^{\mathrm{T}}\mathbf{R}_k\phi_{\omega_i\omega_j}(\omega) \end{aligned}\right\} \quad (29)$$

where $\phi_{\omega_i\omega_j}$ and $h_{i,j}, i, j = 0, \ldots, n$ are the second-order partial derivatives of the basis vector and the contribution with respect to ω_i and ω_j, respectively, and

$$\beta_i = \beta_i(\omega) \triangleq [\mathbf{y} - 2r(\omega)\phi(\omega)]^{\mathrm{T}}\mathbf{R}_k\phi_{\omega_i}(\omega). \quad (30)$$

Remark 1: The derivative information of the cost function with regards to the unknown network parameters is always required in a gradient-based learning algorithm. Since this paper deals with the simultaneous RBF neural network construction and parameter optimization, the derivative information used for the parameter optimization is therefore also acquired, but in a quite different approach from the conventional method [29]. In the conventional approach, the linear output weights in the RBF networks are treated as independent parameters in the learning

process. In this paper, due to the introduction of the matrix series (9), the linear output weights are dependent parameters and therefore are eliminated from the equation by substituting the least squares solution (5) into the cost function (10). Thus, the dimension of the optimization problem is reduced for the whole network construction and parameter optimization process.

For those RBF neural networks that use Gaussian function as the basis function, the gradient and the Hessian matrix of the Gaussian function can be explicitly expressed.

Consider the Gaussian function

$$\phi(\mathbf{x}(\kappa),\sigma,\mathbf{c}) = \exp\left(-\sum_{i=1}^{n}[x_i(\kappa)-c_i]^2/\sigma^2\right) \tag{31}$$

which is ill defined at $\sigma = 0$, which makes it difficult to implement some calculus-based optimization algorithms. To overcome the problem, define

$$|\omega_0| = (\sigma^2)^{-1/2}, \qquad \omega_i = c_i, \qquad i = 1,\ldots,n \tag{32}$$

then rewrite (31) as

$$\phi(\mathbf{x}(\kappa),\omega) = \exp\left(-\omega_0^2\sum_{i=1}^{n}[x_i(\kappa)-\omega_i]^2\right) \tag{33}$$

and the first-order partial derivatives of the Gaussian basis vector with respect to the adjustable parameters are

$$\left.\begin{array}{c}\phi_{\omega_0}(\omega) = -2\omega_0\mathbf{s}\circ\phi(\omega)\\ \phi_{\omega_i}(\omega) = 2\omega_0^2(\mathbf{x}_i-\omega_i)\circ\phi(\omega), i=1,\ldots,n\end{array}\right\} \tag{34}$$

and the second-order partial derivatives are derived as

$$\left.\begin{array}{c}\phi_{\omega_0\omega_0}(\omega) = 4\omega_0^2\mathbf{s}\circ\mathbf{s}\circ\phi(\omega) - 2\mathbf{s}\circ\phi(\omega)\\ \phi_{\omega_0\omega_i}(\omega) = \phi_{\omega_i\omega_0}(\omega)\\ = 4\omega_0\left[1-\omega_0^2\mathbf{s}\right]\circ(\mathbf{x}_i-\omega_i)\circ\phi(\omega), i=1,\ldots,n\\ \phi_{\omega_i\omega_j}(\omega) = \phi_{\omega_j\omega_i}(\omega)\\ = 4\omega_0^4(\mathbf{x}_i-\omega_i)\circ(\mathbf{x}_j-\omega_j)\circ\phi(\omega)\\ i,j=1,\ldots,n, i\neq j\\ \phi_{\omega_i\omega_i}(\omega) = 2\omega_0^2\left[2\omega_0^2(\mathbf{x}_i-\omega_i)^2-1\right]\circ\phi(\omega)\\ i=1,\ldots,n\end{array}\right\} \tag{35}$$

where $\circ$ denotes the element-by-element Kronecker product between two vectors and

$$\mathbf{s} = \mathbf{s}(\omega) = \sum_{i=1}^{n}[(x_i(1)-\omega_i)^2,\ldots,(x_i(N)-\omega_i)^2]^{\mathrm{T}}. \tag{36}$$

Given (34), the elements of the gradient of the contribution for the Gaussian function can be computed as

$$\left.\begin{array}{c}g_0(\omega)\\ = -4\omega_0 r(\omega)[\mathbf{y}-r(\omega)\phi(\omega)]^{\mathrm{T}}\mathbf{R}_k[\mathbf{s}(\omega)\circ\phi(\omega)]\\ g_i(\omega)\\ = 4\omega_0^2 r(\omega)[\mathbf{y}-r(\omega)\phi(\omega)]^{\mathrm{T}}\mathbf{R}_k[\mathbf{x}_i\circ\phi(\omega)]\\ i=1,\ldots,n\end{array}\right\} \tag{37}$$

and the elements of the Hessian matrix for the Gaussian function are given by

$$\left.\begin{array}{c}h_{0,0} = 4\omega_0\left\{\dfrac{2\omega_0\gamma_0\gamma_0}{D} + r(2\omega_0\mathbf{s}-1)^{\mathrm{T}}\chi_0^{(k)}\right.\\ \left.-2\omega_0 r^2(\mathbf{s}\circ\phi)^{\mathrm{T}}\mathbf{R}_k(\mathbf{s}\circ\phi)\right\}\\ h_{0,i} = -8\omega_0\left\{\dfrac{\omega_0^2\gamma_0\gamma_i}{D} + r(\omega_0^2\mathbf{s}-1)^{\mathrm{T}}\chi_i^{(k)}\right.\\ \left.-\omega_0^2 r^2(\mathbf{x}_i\circ\phi)^{\mathrm{T}}\mathbf{R}_k(\mathbf{s}\circ\phi)\right\}, i=1,\ldots,n\\ h_{i,j} = 8\omega_0^4\left[\dfrac{\gamma_i\gamma_j}{D} + r\mathbf{x}_j^{\mathrm{T}}\chi_i^{(k)}\right.\\ \left.-r^2(\mathbf{x}_j\circ\phi)^{\mathrm{T}}\mathbf{R}_k(\mathbf{x}_i\circ\phi)\right]\\ i,j=1,\ldots,n\end{array}\right\} \tag{38}$$

where $D, r, \mathbf{s}$ are given in (23), (28), and (36), respectively, and

$$\left.\begin{array}{c}\gamma_0 = \gamma_0(\omega)\\ = [\mathbf{y}-2r(\omega)\phi(\omega)]^{\mathrm{T}}\mathbf{R}_k[\mathbf{s}(\omega)\circ\phi(\omega)]\\ \gamma_i = \gamma_i(\omega)\\ = [\mathbf{y}-2r(\omega)\phi(\omega)]^{\mathrm{T}}\mathbf{R}_k[\mathbf{x}_i\circ\phi(\omega)]\\ \chi_0^{(k)} = \chi_0^{(k)}(\omega)\\ = [\mathbf{R}_k(\mathbf{y}-r(\omega)\phi(\omega))]\circ\mathbf{s}(\omega)\circ\phi(\omega)\\ \chi_i^{(k)} = \chi_i^{(k)}(\omega)\\ = [\mathbf{R}_k(\mathbf{y}-r(\omega)\phi(\omega))]\circ\mathbf{x}_i\circ\phi(\omega)\end{array}\right\}. \tag{39}$$

In (37) and (38), $\mathbf{s}\circ\phi, \mathbf{x}_i\circ\phi, \chi_0^{(k)}$, and $\chi_i^{(k)}$ are intermediate quantities which are used to reduce the computational complexity. In addition, $\mathbf{R}_k(\mathbf{s}\circ\phi)$ and $\mathbf{R}_k(\mathbf{x}_i\circ\phi)$ for $i=1,\ldots,n$ can be computed recursively using just the same way as for $\phi^{(k)} = \mathbf{R}_k\phi$ given in (18).

Both the gradient vector and the Hessian matrix of the contribution are used in the Newton method to search for the points in the continuous space that gives the maximum contribution.

To speed up the convergence of the Newton search and also based on the definition of the RBF centers [29], a subset of training samples is randomly selected as the initial RBF centers. When a new RBF node is to be added into the network, the best one selected from this subset of training samples is used as the starting point for the continuous gradient-based searching for the optimal parameter vector. It should be pointed out that, in the proposed new algorithm, the subset of training samples used to initialize the RBF centers is usually much smaller in size than the set of candidate centers used in the conventional subset selection algorithms [7], [9], [17]. In detail, for conventional subset selection algorithms, all the node centers in the RBF neural network are selected from the candidates. Therefore, the number of candidate centers has to be very large. In the proposed algorithm, the subset of training samples is simply used as the initial values in the gradient-based search on the continuous parameter space for the optimal RBF centers. Therefore, its size can be very small.

The complete procedure for the Newton search over continuous space can be found in [36], and the Newton search procedure is combined with the forward RBF network construction procedure presented in Section III, leading to the proposed hybrid forward algorithm.

To facilitate efficient numerical implementation, several intermediate matrices and vectors have to be introduced. Suppose k RBF basis vectors have been selected and optimized using the hybrid forward selection procedure. Denote these RBF basis vectors as $p_i, \ldots, p_k$ to distinguish from the candidate RBF basis vectors. For these k optimized basis vectors, define a $k \times k$ upper triangular matrix $\mathbf{A}$ as

$$\mathbf{A} \underline{\underline{\Delta}} [a_{i,j}]_{k\times k}$$
$$a_{i,j} = \begin{cases} 0, & j < i \\ p_i^{\mathrm{T}} \mathbf{R}_{i-1} p_i = \left(p_i^{(i-1)}\right)^{\mathrm{T}} p_i^{(i-1)}, & j = i \\ \dfrac{p_i^{\mathrm{T}} \mathbf{R}_{i-1} p_j}{p_i^{\mathrm{T}} \mathbf{R}_{i-1} p_i} = \left(p_i^{(i-1)}\right)^{\mathrm{T}} p_j^{(i-1)} / a_{i,i}, & i < j \le k \end{cases} \tag{40}$$

and a $k\times 1$ vector $\mathbf{a}_y$ as

$$\mathbf{a}_y \underline{\underline{\Delta}} [a_{i,y}]_{k\times 1}$$
$$a_{i,y} = \frac{\mathbf{y}^{\mathrm{T}} \mathbf{R}_{i-1} p_i}{p_i^{\mathrm{T}} \mathbf{R}_{i-1} p_i} = \left(\mathbf{y}^{(i-1)}\right)^{\mathrm{T}} p_i^{(i-1)} / a_{i,i}. \tag{41}$$

For the M candidate RBF basis vectors, define a $k\times M$ matrix $\mathbf{B}$ as

$$\mathbf{B} \underline{\underline{\Delta}} [b_{i,j}]_{k\times M}, b_{i,j} = p_i^{\mathrm{T}} \mathbf{R}_{i-1} \phi_j = \left(p_i^{(i-1)}\right)^{\mathrm{T}} \phi_j \tag{42}$$

and two $M \times 1$ vectors $\mathbf{c}^{k)}$ and $\mathbf{d}^{k)}$ as

$$\left.\begin{aligned} \mathbf{c}^{k)} \underline{\underline{\Delta}} [c_i^{k)}]_{M\times 1}, c_i^{k)} = \mathbf{y}^{\mathrm{T}} \mathbf{R}_k \phi_i \\ \mathbf{d}^{k)} \underline{\underline{\Delta}} [d_i^{k)}]_{M\times 1}, d_i^{k)} = \phi_i^{\mathrm{T}} \mathbf{R}_k \phi_i \end{aligned}\right\} \tag{43}$$

where the superscript k for vectors $\mathbf{c}^{k)}$ and $\mathbf{d}^{k)}$ indicates the updated values for $\mathbf{c}$ and $\mathbf{d}$ as the number of RBF nodes k increases.

With these above notations, (22) can be rewritten as

$$\Delta Q_{k+1}(\phi_i) = \left(c_i^{k)}\right)^2 / d_i^{k)}, \qquad i = 1, \ldots, M. \tag{44}$$

Now suppose a new RBF node, say, the $(k+1)$th optimized basis vector $\mathbf{p}_{k+1}$, will be added into the network. Then all the intermediate matrix and vectors defined in (40)–(43) have to be updated.

1) First, the $(k+1)$th column of the triangular matrix $\mathbf{A}$ has to be computed and the new basis vector p_{k+1} has to be updated from $p_{k+1}^{(0)} = p_{k+1}$ to $p_{k+1}^{(k)}$ using the following recursive formulas:

$$\left.\begin{aligned} a_{i,k+1} &= \left(p_i^{(i-1)}\right)^{\mathrm{T}} p_{k+1}^{(i-1)} / a_{i,i} \\ p_{k+1}^{(i)} &= p_{k+1}^{(i-1)} - \frac{\left(p_i^{(i-1)}\right)^{\mathrm{T}} p_{k+1}^{(i-1)}}{(p_i^{(i-1)})^{\mathrm{T}} p_i^{(i-1)}} p_i^{(i-1)} \\ &= p_{k+1}^{(i-1)} - a_{i,k+1} p_i^{(i-1)} \\ & i = 1, \ldots, k \\ a_{k+1,k+1} &= \left(p_{k+1}^{(k)}\right)^{\mathrm{T}} p_{k+1}^{(k)}, \end{aligned}\right\}. \tag{45}$$

2) Then the output vector has to be updated and the $(k+1)$th element of $\mathbf{a}_y$ has to be computed, which can be done as follows:

$$\left.\begin{aligned} a_{k+1,y} &= \left(\mathbf{y}^{(k)}\right)^{\mathrm{T}} p_{k+1}^{(k)} / a_{k+1,k+1} \\ \mathbf{y}^{(k+1)} &= \mathbf{y}^{(k)} - \frac{\left(\mathbf{y}^{(k)}\right)^{\mathrm{T}} p_{k+1}^{(k)}}{\left(p_{k+1}^{(k)}\right)^{\mathrm{T}} p_{k+1}^{(k)}} p_{k+1}^{(k)} \\ &= \mathbf{y}^{(k)} - a_{k+1,y} p_{k+1}^{(k)} \end{aligned}\right\}. \tag{46}$$

3) Finally, the $(k+1)$th row of $\mathbf{B}$ has to be computed according to (42), and vectors $\mathbf{c}^{k)}$ and $\mathbf{d}^{k)}$ have to be updated for further RBF network construction (adding more RBF nodes)

$$\left.\begin{aligned} c_i^{k+1)} &= \mathbf{y}^{\mathrm{T}} \mathbf{R}_{k+1} \phi_i \\ &= \mathbf{y}^{\mathrm{T}} \left(\mathbf{R}_k - \frac{\mathbf{R}_k p_{k+1} p_{k+1}^{\mathrm{T}} \mathbf{R}_k^{\mathrm{T}}}{p_{k+1}^{\mathrm{T}} \mathbf{R}_k p_{k+1}}\right) \phi_i \\ &= c_i^{k)} - a_{k+1,y} b_{k+1,i} \\ d_i^{k+1)} &= \phi_i^{\mathrm{T}} \mathbf{R}_{k+1} \phi_i \\ &= \phi_i^{\mathrm{T}} \left(\mathbf{R}_k - \frac{\mathbf{R}_k p_{k+1} p_{k+1}^{\mathrm{T}} \mathbf{R}_k^{\mathrm{T}}}{p_{k+1}^{\mathrm{T}} \mathbf{R}_k p_{k+1}}\right) \phi_i \\ &= d_i^{k)} - \frac{b_{k+1,i}^2}{a_{k+1,k+1}} \\ & i = 1, \ldots, M \end{aligned}\right\}. \tag{47}$$

With all the above-detailed formulas, the detailed procedures of the HFA for the RBF network construction can now be given as follows.

A. Algorithm: HFA for RBF Neural Network Construction

Step 1—Initialization: Assign an initial value for the width parameter of the basis functions and randomly select a small set of, say, M RBF centers from the whole set of training samples. Then construct the corresponding candidate RBF basis vector set $\{\phi_i, i = 1, \ldots, M\}$. Let $k = 0, \mathbf{y}^{(0)} = \mathbf{y}$, compute $\mathbf{c}^{0)} = [\mathbf{y}^{\mathrm{T}}\phi_1, \ldots, \mathbf{y}^{\mathrm{T}}\phi_M]^{\mathrm{T}}$ and $d^{0)} = [\phi_1^{\mathrm{T}}\phi_1, \ldots, \phi_M^{\mathrm{T}}\phi_M]^{\mathrm{T}}$ according to (43), and compute SSE$= \mathbf{y}^{\mathrm{T}}\mathbf{y}$.

Step 2: Add the $(k+1)$th RBF node into the network. The $(k+1)$th RBF node is optimized as follows.

A) Compute the contribution for all the candidate RBF basis vectors using (44) and identify the one that gives the maximum contribution.

B) Use the width and the center vector that gives the maximum contribution as the initial parameter value, denoted as $\omega_{k+1}^{0)}$, for the $(k+1)$th RBF node. Search for the optimum ω_{k+1} over the continuous parameter space using the Newton method.

C) Denote the $(k+1)$th optimized basis vector as $p_{k+1} = \phi(\omega_{k+1})$. Compute $p_{k+1}^{(k)}$ and the $(k+1)$th column of matrix $\mathbf{A}$ in (45), the $(k+1)$th row of matrix $\mathbf{B}$ in (42) and $\mathbf{y}^{(k+1)}$, and the $(k+1)$th element of vector $\mathbf{a}_y$ in (46). Update vectors $\mathbf{c}^{k)}$ and $\mathbf{d}^{k)}$ into $\mathbf{c}^{k+1)}$ and $\mathbf{d}^{k+1)}$ using (47).

D) Reduce the cost function SSE by SSE $\leftarrow$ SSE $-$ $\Delta Q_{k+1}(p_{k+1})$ and let $k \leftarrow k + 1$.

Step 3: Repeat Step 2 to add more RBF nodes into the network, until some neural network construction criterion is satisfied. This, for example, can be that a certain number, say, m, of RBF nodes have been selected, or the SSE is reduced to a given level or some information criterion such as AIC [3] is satisfied.

The linear output weights of the final RBF network of, say, m hidden nodes can be solved from a linear equation with an upper triangular coefficient matrix. Based on the least squares solution of the linear output weights given in (3) and the definition of $\mathbf{R}_k$, it follows that

$$\mathbf{P}\mathbf{w} = \mathbf{P}(\mathbf{P}^{\mathrm{T}}\mathbf{P})^{-1}\mathbf{P}^{\mathrm{T}}\mathbf{y} = (\mathbf{I} - \mathbf{R}_m)\mathbf{y} \tag{48}$$

$$w_1 p_1 + w_2 p_2 + \cdots + w_m p_m = \mathbf{y} - \mathbf{R}_m \mathbf{y} \tag{49}$$

where $\mathbf{P} = [p_1, p_2, \ldots, p_m]$ denotes the regression matrix consisting of the m optimized basis vectors and $w = [w_1, w_2, \ldots, w_m]^{\mathrm{T}}$ is the output weight vector.

For $k = 1$ to m, left-multiplying (49) with $(p_k^{(k-1)})^{\mathrm{T}} = (\mathbf{R}_{k-1}p_k)^{\mathrm{T}}$, respectively, and taking into account the property $R_k p_i = 0$ for $i \le k$ given in (14) as well as the notations given in (40) and (41), it follows that

$$a_{k,k}w_k + a_{k,k}\sum_{i=k+1}^{m} a_{k,i}w_i = a_{k,k}a_{k,y}, \qquad k = 1, \ldots, m. \tag{50}$$

Equation (50) is a set of linear equations relating to the upper triangular coefficient matrix $\mathbf{A}$. According to Theorem 2, if the regression matrix $\mathbf{P}$ is fully ranked, then $a_{k,k} \neq 0$ and therefore the m linear output weights can then be easily solved from (50) using back substitution

$$w_k = a_{k,y} - \sum_{i=k+1}^{m} w_i a_{k,i}, \qquad k = m, \quad m-1, \ldots, 1. \tag{51}$$

V. Computational Complexity

For the Gaussian functions (33), if each evaluation of the exponential function is regarded as one floating-point operation (FPO), then the total number of FPOs to compute the contribution for the $(k+1)$th RBF node using (22) and (23), including computing $\phi^{(k)}(\omega) = \mathbf{R}_k\phi(\omega)$ for (23) in the same recursive way as in (45), is

$$C_{\Delta Q_{k+1}} = (3n + 4k + 7)N. \tag{52}$$

The additional numbers of FPOs to compute the $n+1$ elements of the gradient in (37) and the $(n+1)(n+1)$ elements of the Hessian matrix in (38) and (39) are, respectively

$$\left.\begin{array}{c} C_{\mathrm{Grad}} = (2n+5)N + 3 \\ C_{\mathrm{Hessian}} = [(n+1)(2n+4k+8)+4]N \\ \quad +4(n+2)(n+1) \end{array}\right\}. \tag{53}$$

If (52) is compared with (53), it could be found that less than two-thirds of additional FPOs are required to compute the $n+1$ gradient components; and less than $n+1$ times additional FPOs for the $(n+1)$-by-$(n+1)$ Hessian matrix computation. It should be noted that, for the Gaussian basis function, no additional evaluation of the exponential function is required in computing the gradient vector and the Hessian matrix.

VI. Simulation Example

Among various RBF neural network construction algorithms, the forward subset selection such as the modified Gram–Schmidt (MGS) based OLS is perhaps the most popular method used in the open literature [7]–[9]. As a most recent development in this area, the SIR modeling method [10] was proposed, aiming to improve the network performance. In this paper, the proposed new algorithm HFA was compared with the above two algorithms, and these algorithms were tested on all the simulation examples given in this section. All the following tests were carried out using MATLAB V5.3 on a PIII-800 MHz desktop PC with Windows XP.

Example 1: Consider the following nonlinear function taken from [10]:

$$y(x) = 0.1x + \frac{\sin x}{x} + \sin 0.5x, \qquad -10 \le x \le 10. \tag{54}$$

As in [10], 1000 data points were generated using $y = f(x) + \xi$, where x was uniformly distributed within $[-10, 10]$ and ξ was a Gaussian white noise with zero mean and variance 0.01. The first 500 points were used for network training and the other 500 points for network validation.

For MGS all 500 training data samples were used as the candidate centers. Based on a series of judicious trial-and-error tests for $\sigma \in \{0.5, 1.0, \ldots, 10\}$, the width for the Gaussian basis function is chosen as $\sigma = 3.0$. For HFA, ten candidate centers are randomly selected from the 500 training samples, and the initial value for the width parameter for all the ten candidate RBF basis vectors was simply set to be $\omega_0 = 1/20$ (which can cover the whole input range). For SIR, similar algorithm parameters as in [10] were used for the boosting search, i.e., the population size $s = 5$, the number of iterations $M_I = 20$, and the repeating times $M_R = 10$. These three algorithms were then used to generate the RBF networks of different sizes.

The rooted mean squared error (RMSE) [2], [20] over the training data set, measured running time, and flops of the three algorithms are listed in Table I. The RBF networks produced by the three algorithms were then tested on the validation data of the rest of the 500 samples, and the RMSE values over the validation data set are listed in Table II. The RMSE is defined as

$$\mathrm{RMSE} = [(\hat{\mathbf{y}} - \mathbf{y})^{\mathrm{T}}(\hat{\mathbf{y}} - \mathbf{y})/N]^{1/2} \tag{55}$$

where $\hat{\mathbf{y}}$ is the RBF neural network predictions for target $\mathbf{y}$ and N is the vector length.

Both the measured running time and flops listed in Table I indicate that the computational complexity of the proposed HFA is significantly lower than the conventional MGS-based OLS algorithm and the SIR method. It should be noted that the running time and the number of flops in Table I and the following experiments include calculations of all the candidate RBF basis vectors. The trial and error tests for the optimal width for MGS were, however, not included.

It is also shown in Tables I and II that the training and validation RMSE values obtained by the proposed HFA are quite the same as the MGS and SIR in all cases; however, the proposed HFA is able to produce a RBF neural network with significantly reduced computational complexity.

TABLE I
RMSE VALUES OVER THE TRAINING DATA AND THE COMPUTATIONAL COMPLEXITIES FOR EXAMPLE 1

Hidden	Training error RMSE			Running time (s)			Number of flops		
nodes	HFA	MGS	SIR	HFA	MGS	SIR	HFA	MGS	SIR
4	0.1156	0.1523	0.1271	0.15	0.69	1.22	2.98e+06	1.05e+07	1.81e+07
5	0.1128	0.1073	0.1160	0.22	0.83	1.60	4.33e+06	1.27e+07	2.54e+07
6	0.1002	0.1034	0.1088	0.24	0.98	2.00	4.98e+06	1.49e+07	3.39e+07
7	0.0964	0.0970	0.0986	0.29	1.09	2.44	6.20e+06	1.72e+07	4.35e+07
8	0.0955	0.0964	0.0962	0.36	1.23	3.00	6.84e+06	1.94e+07	5.42e+07

TABLE II
VALIDATION PERFORMANCES FOR EXAMPLE 1

Hidden	Validation error (RMSE)		
nodes	HFA	MGS	SIR
4	0.1255	0.1524	0.1240
5	0.1197	0.1183	0.1149
6	0.1062	0.1135	0.1117
7	0.1032	0.1047	0.1038
8	0.1018	0.1039	0.1026

TABLE III
RESULTS FOR THE HFA USING DIFFERENT NUMBER OF CANDIDATES FOR EXAMPLE 1

Hidden	1 candidate		3 candidates		5 candidates		20 candidates	
nodes	T.RMSE	V.RMSE	T.RMSE	V.RMSE	T.RMSE	V.RMSE	T.RMSE	V.RMSE
4	0.2278	0.2130	0.2278	0.2130	0.1173	0.1262	0.1173	0.1262
5	0.1092	0.1157	0.1092	0.1157	0.1156	0.1221	0.1156	0.1221
6	0.1009	0.1077	0.1003	0.1067	0.1010	0.1076	0.1010	0.1076
7	0.1007	0.1073	0.0965	0.1010	0.1002	0.1067	0.1002	0.1067
8	0.0997	0.1059	0.0963	0.1010	0.0961	0.1009	0.0959	0.0995

It should be noted again that, in the proposed HFA, the candidate set of the RBF basis vectors are randomly selected from the train samples, and it is only used to initialize the RBF neural parameters for the Newton search. Therefore the size of the candidate set for the proposed algorithm is not as vital as for the conventional subset selection algorithms.

To further examine how the number of candidate RBF basis vectors affects the resultant network performance for the proposed algorithm, the experiments on HFA were repeated with different numbers of candidate centers. The results are listed in Table III, where T.RMSE stands for the training RMSE values and V.RMSE stands for the validation RMSE values. It is shown in Table III that, except for the RBF networks of four nodes with one and three candidates, respectively, are significantly worse, all the others have quite similar performance.

Example 2: Prediction of the chaotic time series generated from the Mackey–Glass differential equation

$$\frac{dy(t)}{dt} = \frac{ay(t-\tau)}{1+y^{10}(t-\tau)} - by(t), \qquad a = 0.2, \qquad b = 0.1, \qquad \tau = 17. \quad (56)$$

The Mackey–Glass time series was widely used as a benchmark problem for neural network training in the literature [2], [20], [34]. Although it was reported in [34] that the prediction error variance for (56) is much smaller when RBF-AR models are used rather than RBF networks, the objective of this paper is to test the three algorithms on RBF networks construction. Therefore, to use other model types to predict this particular time series is out of the scope of this paper. According to [20], the input vector to the RBF network is selected as $\mathbf{x} = [y(t), y(t-6), y(t-12), y(t-18)]$ and the network output is $\hat{y}(t+1)$. The produced networks were used to predict $y(t+6), y(t+50)$ and $y(t+84)$ from $y(t-l)$ for $l \in \{0, 6, 12, 18\}$.

The trapezoidal rule for integration over the time interval $[t, t+\Delta t]$ yields

$$[y(t+\Delta) - y(t)] = \left[\frac{dy(t+\Delta)}{dt} + \frac{dy(t)}{dt}\right]\frac{\Delta}{2}. \quad (57)$$

The discrete equation corresponding to (56) is then given as follows according to [20]:

$$y(t+\Delta) = \frac{(2-b\Delta)}{(2+b\Delta)}y(t) + \frac{a\Delta}{(2+b\Delta)}\left[\frac{y(t+\Delta-\tau)}{1+y^{10}(t+\Delta-\tau)} + \frac{y(t-\tau)}{1+y^{10}(t-\tau)}\right]. \quad (58)$$

Following [20], by setting $\Delta = 1$ and the initial condition for $y(t) = 0.3$ for $-\tau \le t \le 0$, a time series of 4500 samples was generated using (58). The first 4000 samples were used as the training data set and the remaining 500 samples for network validation. The three algorithms were used to construct the RBF neural networks of different sizes. For MGS, 2000 samples were randomly selected from the training sample set as the candidate centers, which is the best choice based on a series of experiments with 500, 1000, and 3500 candidates, respectively. The experiment failed when all 4000 training samples were used as the candidates, due to the matrix being too large to carry out on the PC. The width is chosen as $\sigma = 1.5$ for all 2000 candidate Gaussian basis functions again based on a number of judicious trial-and-error tests. For SIR, a series of tests are performed in advance [10], and the algorithm parameters are tuned to be $s = 15, M_I = 40$, and $M_R = 40$. For the HFA, 80 candidate centers are randomly selected from the 4000 training samples. The initial value for ω_0 was simply set at one for all the 80 candidate basis vectors, given the range of the time series being [0.3, 1.32]. The training RMSE, measured running time, and flops are listed in Table IV for the three algorithms.

It is shown in Table IV that the three algorithms again gave equivalent training RMSE values in all the cases. However, the HFA is computationally much cheaper to implement than MGS and SIR. It is noticed that the measured flops for the MGS are about two times of those for the HFA. However, the MGS takes even longer running time than the HFA because MGS requires many more evaluations of the exponential function and has to operate on a very large matrix (a 4000-by-2000 matrix storing all the candidate RBF basis vectors).

The produced RBF neural networks were then applied to perform 6-, 50- and 84-steps-ahead predictions over the validation

TABLE IV
RMSE VALUES OVER THE TRAINING DATA AND THE COMPUTATIONAL COMPLEXITIES FOR EXAMPLE 2

Hidden nodes	RMSE ($\times 10^{-3}$)			Run-time (s)			Number of flops		
	HFA	MGS	SIR	HFA	MGS	SIR	HFA	MGS	SIR
10	6.46	8.06	6.32	6.88	43.02	244.52	2.88e+08	8.14e+08	6.68e+09
11	6.10	7.54	5.87	7.75	46.77	235.42	3.10e+08	8.86e+08	7.76e+09
12	4.81	5.82	5.72	8.44	50.58	259.86	3.41e+08	9.57e+08	8.92e+09
13	3.99	4.55	5.53	9.75	54.23	309.74	3.99e+08	1.03e+09	1.02e+10
14	3.97	3.90	5.40	10.11	58.30	339.66	4.23e+08	1.10e+09	1.15e+10
15	3.63	3.69	5.10	10.84	61.86	372.36	4.57e+08	1.17e+09	1.29e+10
16	3.44	3.44	4.47	11.88	66.22	415.39	4.92e+08	1.24e+09	1.43e+10
17	3.23	3.23	3.99	14.44	69.52	453.08	6.23e+08	1.31e+09	1.59e+10
18	3.15	3.18	3.48	15.94	74.81	494.30	6.94e+08	1.39e+09	1.75e+10
19	3.11	2.91	3.28	16.72	77.45	557.34	7.50e+08	1.46e+09	1.92e+10
20	3.04	2.72	3.14	19.03	81.72	621.49	9.12e+08	1.53e+09	2.09e+10

TABLE V
LONG-TERM PREDICTION PERFORMANCES (RMSE $\times 10^{-1}$) OVER THE VALIDATION DATA SET FOR EXAMPLE 2

Hidden nodes	6-step-ahead			50-step-ahead			84-step-ahead		
	HFA	MGS	SIR	HFA	MGS	SIR	HFA	MGS	SIR
10	0.41	6.10	1.11	0.78	10.50	3.42	1.51	10.68	4.15
11	0.40	7.85	0.91	0.78	11.23	2.51	1.48	11.31	3.06
12	0.23	9.09	1.07	0.50	11.83	3.00	0.85	11.81	3.18
13	0.18	5.60	1.20	0.42	10.46	2.71	0.56	10.54	2.42
14	0.18	3.53	1.13	0.43	6.34	2.62	0.58	7.99	2.40
15	0.16	0.58	1.19	0.28	0.97	2.65	0.44	1.67	2.51
16	0.15	4.07	1.26	0.30	4.98	2.94	0.51	4.37	2.78
17	0.23	1.79	1.19	0.37	2.86	2.69	0.74	3.35	3.00
18	0.25	0.65	1.10	0.35	1.19	2.50	0.61	1.99	2.80
19	0.25	5.24	1.16	0.35	11.30	2.69	0.58	11.43	2.99
20	0.19	1.85	1.11	0.29	2.24	2.67	0.45	2.21	3.03

data set, and the RMSE values of the predictions are compared in Tables V. It is shown that the proposed HFA produces significantly better long-term predictions than the other two algorithms over the validation data set.

Example 3–Modeling of Continuously Stirred Tank Reactor: Simulated continuously stirred tank reactor (CSTR) [11] represents a wide class of chemical processes exhibiting a high degree of nonlinearity. Within a CSTR, two chemicals are mixed and react to produce a product compound at a concentration $C_a(t)$, and the temperature of the mixture is $T(t)$. The CSTR system considered here is a single-input single-output system, where the input variable is the flow rate of a coolant $q_c(t)$ and the output variable is the concentration of the product $C_a(t)$. The reaction is exothermic; if uncooled, the heat it generates acts to slow it down. Fig. 1 shows a simple schematic of the plant, and the system can be described as follows:

$$\dot{C}_a(t) = \frac{q}{v}(C_{ao} - C_a(t)) - k_o C_a(t) \exp\left(-\frac{E}{R \cdot T(t)}\right) \quad (59)$$

$$\dot{T}(t) = \frac{q}{v} \cdot (T_o - T(t)) + k_1 \cdot C_a(t) \cdot \exp\left(-\frac{E}{R \cdot T(t)}\right) + k_2 \cdot q_c(t) \cdot \left(1 - \exp\left(-\frac{k_3}{q_c(t)}\right)\right) \cdot (T_{co} - T(t)) \quad (60)$$

where C_a is the product concentration, C_{ao} is the inlet feed concentration, q is the process flow rate, and T_o and T_{co} are the inlet feed and coolant temperatures, respectively. $k_o, E/R, v, k_1, k_2$, and k_3 are thermodynamic and chemical constants relating to this particular problem.

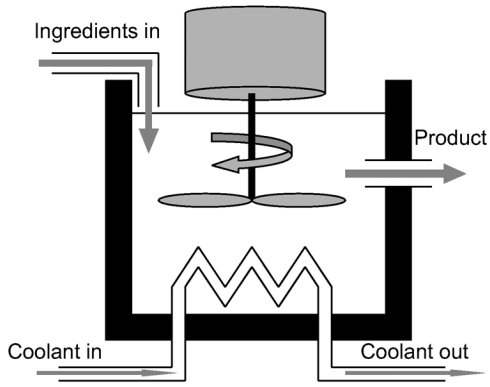


Fig. 1. Schematic of a CSTR plant.

TABLE VI
STATISTICS OF THE CSTR DATA

Data	Mean	Deviation
q_c (l/min)	103.42	5.3705
C_a (mol/l)	0.1052	0.0305

The system was simulated with a sampling interval of 0.2 s, and the system input $q_c(t)$ is subject to uniformly distributed random perturbation over the range [−10, 10] l/min from the operating point. A normally distributed random signal was added to the output to simulate measurement noise. A set of 1500 data samples was generated and used for modeling. The statistics of the plant data are shown in Table VI.

The inputs to the RBF networks were selected as $q_c(t), q_c(t-2), q_c(t-3), q_c(t-4), C_a(t), C_a(t-1), C_a(t-2)$, and $C_a(t-3)$. The RBF network output predicts $C_a(t+1)$. The plant data were split into two sets; the first 700 samples were used for network training, the remaining 800 as validation data. All data samples were zero meaned and normalized prior to the following modeling experiments.

The three algorithms were used to produce the RBF network models. For MGS, all 700 training samples are used as the candidate centers, and the widths of the Gaussian basis functions were chosen as $\sigma = 8.0$ again based on a number of judicious trial and error tests. For SIR, a series of preliminary tests are performed to determine the algorithm parameters, and the algorithm parameters were chosen as $s = 14, M_I = 60$ and $M_R = 50$. For HFA, 40 candidate centers are randomly selected from the 700 training samples. The initial value for ω_0 was simply chosen to be 0.5 for all the 40 candidate basis vectors, given the range of the normalized data as [−1.85, 1.69] for $q_c(t)$ and [−1.86, 5.46] for $C_a(t)$.

RBF neural networks of size from four to eight were generated using the three algorithms. The training RMSE values, running time, and flops are listed in Table VII. It is shown that the three algorithms produced equivalent training RMSE values. The computational complexities of both the HFA and the MGS are quite similar but both are much simpler than the SIR.

The produced networks were then used to perform one-step-ahead and long-term predictions, and the results are compared in Table VIII. It is shown that networks produced by the HFA have the best generalization performance for both one-step-ahead and long-term predictions. As an example, the long-term predictions

TABLE VII
RMSE VALUES OVER THE TRAINING DATA AND THE COMPUTATIONAL COMPLEXITIES FOR EXAMPLE 3

Hidden nodes	Training RMSE ($\times 10^{-1}$)			Running time (s)			Number of flops		
	HFA	MGS	SIR	HFA	MGS	SIR	HFA	MGS	SIR
4	3.46	2.68	4.18	0.59	1.44	34.89	2.11e+07	2.75e+07	7.82e+08
5	2.54	2.47	3.01	0.97	1.67	43.67	3.25e+07	3.18e+07	1.04e+09
6	2.40	2.16	2.25	1.88	1.94	55.78	6.05e+07	3.62e+07	1.32e+09
7	1.96	2.11	2.18	2.33	2.17	69.11	7.73e+07	4.05e+07	1.62e+09
8	1.92	1.95	2.11	2.59	2.53	74.31	8.50e+07	4.48e+07	1.94e+09

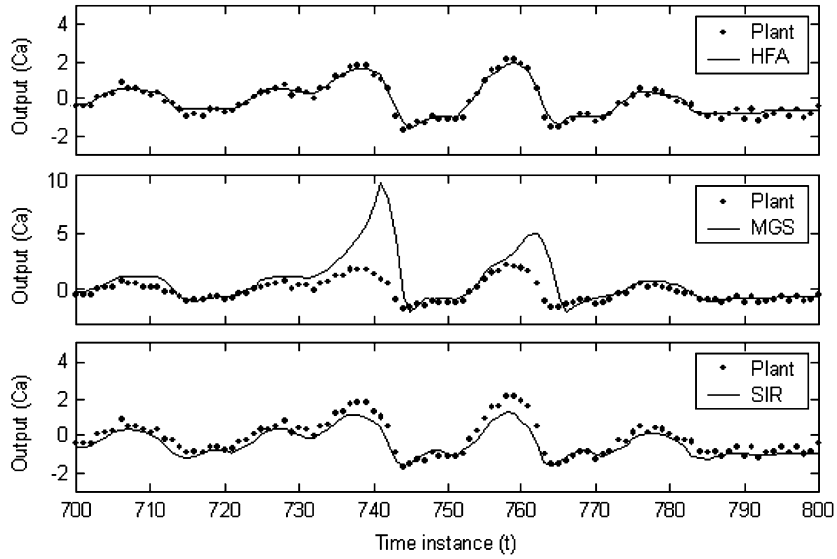


Fig. 2. Long-term prediction for the last 100 outputs of the validation set by six-hidden-node RBF networks produced by HFA, MGS, and SIR of eight hidden nodes with different RBF width settings.

TABLE VIII
VALIDATION PERFORMANCES (RMSE $\times$ 10^{-1}) FOR EXAMPLE 3

Hidden nodes	One-step-ahead			Long-term		
	HFA	MGS	SIR	HFA	MGS	SIR
4	2.63	4.18	4.81	4.58	24.92	6.13
5	2.89	3.60	3.87	5.23	18.57	5.96
6	2.75	4.06	4.11	2.72	19.94	5.56
7	2.52	4.75	4.21	2.38	24.37	5.38
8	2.44	5.95	4.13	2.34	32.29	5.26

of the six-hidden-nodes RBF networks for the last 100 outputs are illustrated in Fig. 2. Note that the RMSE values indicated in Tables VII and VIII and the data in Fig. 2 are based on the normalized plant data.

Remark 2: It should be pointed out that, by convention, all the results shown in the above three case studies were acquired by optimizing the nonlinear parameters (widths and centers) for the RBF neurons with regards to the training error (RMSE). For the proposed HFA, all the parameters were optimized over the continuous solution space with regards to the training error (RSME). For the conventional MGS method, the RBF width, however, has to be set *a priori*, and in the three examples, this parameter was set with the optimal value which minimizes the training error (RMSE) through a series of exhaustive tests for $\sigma \in \{0.5, 1.0, \ldots, 10.0\}$. Apparently, different choices of the width parameters could lead to different network performance for the MGS method. To illustrate this limitation for the MGS method, the RMSE values of both the one-step-ahead and long-term prediction errors over both the training and validation data sets in Example 3 are compiled in Table IX for the MGS method for $\sigma \in \{0.5, 1.0, \ldots, 10.0\}$. According to Table IX, in terms of the long-term prediction performance over the validation data set, $\sigma = 4.5$ would be the best choice for the MGS method, though the training performance is not the best. Table IX and the above analysis reflect the difficult issues for the MGS method in that it requires manual setting of the nonlinear parameters *a priori* and the network performance cannot be guaranteed.

VII. CONCLUSION

The performance of an RBF neural network constructed by the conventional subset selection algorithms mainly depends on the size and quality of the candidate RBF basis vectors from which the network is solely selected. In the proposed HFA, the network structure is determined by combining both the stepwise forward network configuration and the continuous RBF parameter optimization. These two tasks are performed within a unified analytic framework, leading to significantly improved network performance and reduced memory usage. In the proposed HFA, a small set of candidate RBF center vectors is still used, but it is only used to initialize the parameter optimization in order to speed up the convergence of the Newton search. Therefore, the resultant network performance is no longer significantly dependent on the candidate RBF basis vectors, and the size of the candidate basis vectors therefore can be very small.

TABLE IX
MGS TRAINING RESULTS (RMSE $\times 10^{-1}$) FOR RBF NETWORKS

Width	Training data		Validation data	
setting	1-step	Long-term	1-step	Long-term
0.5	8.162	10.675	8.278	10.053
1.0	5.418	8.590	7.020	7.799
1.5	3.785	5.297	5.961	6.797
2.0	3.133	4.814	4.133	6.537
2.5	2.748	4.394	3.220	10.000
3.0	2.377	4.248	3.250	11.733
3.5	2.390	4.296	2.720	4.580
4.0	2.424	5.674	2.617	4.833
4.5	2.059	5.440	2.274	2.804
5.0	2.025	4.725	2.117	2.952
5.5	2.159	3.629	2.885	5.151
6.0	2.249	5.312	3.342	17.787
6.5	2.085	4.286	2.504	4.913
7.0	2.085	5.450	2.999	16.133
7.5	1.955	5.699	3.793	22.713
8.0	1.948	5.716	5.949	32.295
8.5	1.973	5.781	6.020	32.556
9.0	2.058	4.760	5.642	28.314
9.5	1.982	4.636	7.379	36.852
10.0	1.967	4.976	12.055	48.914

The computational complexity analysis has confirmed the efficiency of the proposed algorithm, and the simulation results have demonstrated its effectiveness.

APPENDIX A
PROOF OF THEOREM 1

From the definition of $\mathbf{R}_k$ in (9), it is obvious that (11) holds for $k=0$. For $k>0, \boldsymbol{\Phi}_{k+1}=[\boldsymbol{\Phi}_k, \phi_{k+1}]$ is of full rank and it can be verified that

$$\begin{aligned}\left(\boldsymbol{\Phi}_{k+1}^{\mathrm{T}}\boldsymbol{\Phi}_{k+1}\right)^{-1} &= \begin{bmatrix}\boldsymbol{\Phi}_k^{\mathrm{T}}\boldsymbol{\Phi}_k & \boldsymbol{\Phi}_k^{\mathrm{T}}\phi_{k+1}\\ \phi_{k+1}^{\mathrm{T}}\boldsymbol{\Phi}_k & \phi_{k+1}^{\mathrm{T}}\phi_{k+1}\end{bmatrix}^{-1}\\ &= \begin{bmatrix}\left(\boldsymbol{\Phi}_k^{\mathrm{T}}\boldsymbol{\Phi}_k\right)^{-1} & \mathbf{0}\\ \mathbf{0} & 0\end{bmatrix} + \frac{1}{\gamma}\begin{bmatrix}\mathbf{v}\mathbf{v}^{\mathrm{T}} & \mathbf{v}\\ \mathbf{v}^{\mathrm{T}} & 1\end{bmatrix}\end{aligned} \tag{61}$$

where the $k\times 1$ column vector $\mathbf{v}$ and the scalar γ are defined as follows:

$$\left.\begin{aligned}\mathbf{v} &= -\left(\boldsymbol{\Phi}_k^{\mathrm{T}}\boldsymbol{\Phi}_k\right)^{-1}\boldsymbol{\Phi}_k^{\mathrm{T}}\phi_{k+1}\\ \gamma &= \phi_{k+1}^{\mathrm{T}}\phi_{k+1} - \phi_{k+1}^{\mathrm{T}}\boldsymbol{\Phi}_k\left(\boldsymbol{\Phi}_k^{\mathrm{T}}\boldsymbol{\Phi}_k\right)^{-1}\boldsymbol{\Phi}_k^{\mathrm{T}}\phi_{k+1}\\ &= \phi_{k+1}^{\mathrm{T}}\mathbf{R}_k\phi_{k+1}\end{aligned}\right\}. \tag{62}$$

It follows from (9) for $\mathbf{R}_k$ and $\mathbf{R}_{k+1}$ that

$$\begin{aligned}\mathbf{R}_{k+1} &= \mathbf{I} - [\boldsymbol{\Phi}_k, \phi_{k+1}]\\ &\quad\times\left(\begin{bmatrix}\left(\boldsymbol{\Phi}_k^{\mathrm{T}}\boldsymbol{\Phi}_k\right)^{-1} & \mathbf{0}\\ \mathbf{0} & 0\end{bmatrix} + \frac{1}{\gamma}\begin{bmatrix}\mathbf{v}\mathbf{v}^{\mathrm{T}} & \mathbf{v}\\ \mathbf{v}^{\mathrm{T}} & 1\end{bmatrix}\right)\\ &\quad\times[\boldsymbol{\Phi}_k, \phi_{k+1}]^{\mathrm{T}}\\ &\triangleq \mathbf{R}_k - \Delta\mathbf{R}_k\end{aligned} \tag{63}$$

where

$$\Delta\mathbf{R}_k = \frac{1}{\gamma}[\boldsymbol{\Phi}_k, \phi_{k+1}]\begin{bmatrix}\mathbf{v}\mathbf{v}^{\mathrm{T}} & \mathbf{v}\\ \mathbf{v}^{\mathrm{T}} & 1\end{bmatrix}[\boldsymbol{\Phi}_k, \phi_{k+1}]^{\mathrm{T}}. \tag{64}$$

Substitute $\mathbf{v}$ and γ in (62) into (64), and note (9) for $\mathbf{R}_k$, yielding

$$\Delta\mathbf{R}_k = \frac{\mathbf{R}_k\phi_{k+1}\phi_{k+1}^{\mathrm{T}}\mathbf{R}_k}{\phi_{k+1}^{\mathrm{T}}\mathbf{R}_k\phi_{k+1}.} \tag{65}$$

Noting the symmetric property of $\mathbf{R}_k$ and based on (63) and (65), (11) is obvious. It also should be noted, according to Theorem 2, that when $[\boldsymbol{\Phi}_k, \phi_{k+1}]$ is of full rank, γ given in (62) is nonzero. Therefore, $\mathbf{R}_k$ defined in (9) is well defined.

APPENDIX B
PROOF OF THEOREM 2

a) Equation (12) is obvious according to the definition (9) of $\mathbf{R}_k$.

b) For $i=j$, the result is obvious from (12).
For $i=j+1$, from (11) and (12), it follows that

$$\begin{aligned}\mathbf{R}_i\mathbf{R}_j &= \left(\mathbf{R}_j - \frac{\mathbf{R}_j\phi_{j+1}\phi_{j+1}^{\mathrm{T}}\mathbf{R}_j^{\mathrm{T}}}{\phi_{j+1}^{\mathrm{T}}\mathbf{R}_j\phi_{j+1}}\right)\mathbf{R}_j\\ &= \mathbf{R}_j - \frac{\mathbf{R}_j\phi_{j+1}\phi_{j+1}^{\mathrm{T}}\mathbf{R}_j^{\mathrm{T}}}{\phi_{j+1}^{\mathrm{T}}\mathbf{R}_j\phi_{j+1}} = \mathbf{R}_i.\end{aligned} \tag{66}$$

Similarly, it can be proved that $\mathbf{R}_j\mathbf{R}_i = \mathbf{R}_i$; therefore (13) holds for $i=j+1$.
For $i>j+1$, apply $\mathbf{R}_{k+1}\mathbf{R}_k=\mathbf{R}_{k+1}$ for $k=i-1, i-2, \ldots, j+1$, resulting in

$$\begin{aligned}\mathbf{R}_i\mathbf{R}_j &= \mathbf{R}_i\mathbf{R}_{i-1}\mathbf{R}_j\\ &= \cdots = \mathbf{R}_i\mathbf{R}_{i-1}\cdots\mathbf{R}_j\\ &= \mathbf{R}_i\mathbf{R}_{i-1}\cdots\mathbf{R}_{j+1} = \cdots = \mathbf{R}_i\end{aligned} \tag{67}$$

Similarly, using the property $\mathbf{R}_{k+1}\mathbf{R}_k=\mathbf{R}_{k+1}$ for $k=i-1, i-2, \ldots, j+1$, gives $\mathbf{R}_j\mathbf{R}_i=\mathbf{R}_i$.
In summary, (13) holds for $i\geq j$.

c) First, the following property is to be proved:

$$\mathbf{R}_i\phi_j = \mathbf{0}, \qquad 1\leq j\leq i. \tag{68}$$

From (11), it is obvious that

$$\begin{aligned}\mathbf{R}_i\phi_i &= \mathbf{R}_{i-1}\phi_i - \frac{\mathbf{R}_{i-1}\phi_i\phi_i^{\mathrm{T}}\mathbf{R}_{i-1}^{\mathrm{T}}}{\phi_i^{\mathrm{T}}\mathbf{R}_{i-1}\phi_i}\phi_i\\ &= \mathbf{R}_{i-1}\phi_i - \mathbf{R}_{i-1}\phi_i = \mathbf{0}\end{aligned} \tag{69}$$

therefore, (68) holds for $j=i$.
For $j=i-1$, noting $\mathbf{R}_{i-1}\phi_{i-1}=\mathbf{0}$ as proved in the above

$$\begin{aligned}\mathbf{R}_i\phi_{i-1} &= \mathbf{R}_{i-1}\phi_{i-1}\\ &\quad - \frac{\mathbf{R}_{i-1}\phi_i\phi_i^{\mathrm{T}}\mathbf{R}_{i-1}^{\mathrm{T}}}{\phi_i^{\mathrm{T}}\mathbf{R}_{i-1}\phi_i}\phi_{i-1} = \mathbf{0}.\end{aligned} \tag{70}$$

This process continues for all $j<i$ to confirm (68).
Now for an arbitrary vector ϕ of appropriate dimension, from (11) and noting $\mathbf{R}_0=\mathbf{I}$, it follows that

$$\begin{aligned}\mathbf{R}_1\phi &= \phi - \frac{\phi_1\phi_1^{\mathrm{T}}}{\phi_1^{\mathrm{T}}\phi_1}\phi\\ &= \phi - \frac{\phi_1^{\mathrm{T}}\phi}{\phi_1^{\mathrm{T}}\phi_1}\phi_1 = \phi - b_1\phi_1\end{aligned} \tag{71}$$

where b_1 is some scalar determined by ϕ_1 and ϕ.
Therefore, it can be further derived that

$$\begin{aligned}\mathbf{R}_2\phi &= \mathbf{R}_1\phi - \frac{\mathbf{R}_1\phi_2\phi_2^{\mathrm{T}}\mathbf{R}_1^{\mathrm{T}}}{\phi_2^{\mathrm{T}}\mathbf{R}_1\phi_2}\phi\\ &= \phi - b_1'\phi_1 - \frac{\phi_2^{\mathrm{T}}\mathbf{R}_1^{\mathrm{T}}\phi}{\phi_2^{\mathrm{T}}\mathbf{R}_1\phi_2}(\phi_2 - b_1'\phi_1)\end{aligned} \tag{72}$$

or rewritten as

$$\mathbf{R}_2\phi = \phi - b_1\phi_1 - b_2\phi_2 \tag{73}$$

where b_1 and b_2 are some scalars determined by ϕ_1, ϕ_2, and ϕ.

More generally, for $k \geq 0$, it holds that

$$\mathbf{R}_k\phi = \phi - b_1\phi_1 - b_2\phi_2 - \cdots - b_k\phi_k \tag{74}$$

with the scalar b_i for $i = 1, \ldots, k$ determined by $\phi_1, \ldots, \phi_k$ and ϕ.

In order to prove (14), it is supposed that rank $([\phi_1, \ldots, \phi_k, \phi]) = k$; then ϕ is linear dependent on $\phi_1, \ldots, \phi_k$, i.e., $\phi = a_1\phi_1 + a_2\phi_2 + \cdots + a_k\phi_k$ with a_i's being some scalars, given that rank$([\phi_1, \ldots, \phi_k]) = k$. Noting (68), (14) always should hold, i.e.,

$$\begin{aligned}\mathbf{R}_k\phi &= a_1\mathbf{R}_k\phi_1 + a_2\mathbf{R}_k\phi_2 + \cdots + a_k\mathbf{R}_k\phi_k \\ &= \mathbf{0}.\end{aligned} \tag{75}$$

Now if rank$([\phi_1, \ldots, \phi_k, \phi]) = k + 1$ and assuming that $\mathbf{R}_k\phi = \mathbf{0}$, it follows from (74) that $\phi - b_1\phi_1 - b_2\phi_2 - \cdots - b_k\phi_k = \mathbf{0}$ or $\phi = b_1\phi_1 + b_2\phi_2 + \cdots + b_k\phi_k$. This contradicts the condition that rank$([\phi_1, \ldots, \phi_k, \phi]) = k + 1$. Therefore $\mathbf{R}_k\phi \neq \mathbf{0}$ should always hold.

ACKNOWLEDGMENT

The authors would like to thank the anonymous associate editor and reviewers for their constructive and helpful comments and suggestions.

REFERENCES

[1] J. Adams and S. Payandeh, "Methods for low-velocity friction compensation: Theory and experimental study," *J. Robot. Syst.*, vol. 13, no. 6, pp. 391–404, 1996.

[2] K. M. Adeney and M. J. Korenberg, "Iterative fast orthogonal search algorithm for MDL-based training of generalized single-layer network," *Neural Netw.*, vol. 13, pp. 787–799, 2000.

[3] H. Akaike, "New look at the statistical model identification," *IEEE Trans. Autom. Control*, vol. AC-19, no. 6, pp. 716–723, Dec. 1974.

[4] N. Ampazis and S. J. Perantonis, "Two highly efficient second-order algorithms for training feedforward networks," *IEEE Trans. Neural Netw.*, vol. 13, no. 3, pp. 1064–1074, May 2002.

[5] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Clarendon, 1995.

[6] F. C. Chen and H. K. Khalil, "Adaptive control of nonlinear systems using neural networks," *Int. J. Contr.*, vol. 45, no. 6, pp. 1299–1317, 1992.

[7] S. Chen and S. A. Billings, "Neural network for nonlinear dynamic system modelling and identification," *Int. J. Contr.*, vol. 56, pp. 319–346, 1992.

[8] S. Chen, S. A. Billings, and W. Luo, "Orthogonal least squares methods and their application to non-linear system identification," *Int. J. Contr.*, vol. 50, no. 5, pp. 1873–1896, 1989.

[9] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis functions," *IEEE Trans. Neural Netw.*, vol. 2, no. 2, pp. 302–309, Mar. 1991.

[10] S. Chen, X. X. Wang, and D. J. Brown, "Sparse incremental regression modeling using correlation criterion with boosting search," *IEEE Signal Process. Lett.*, vol. 12, no. 3, pp. 198–201, Mar. 2005.

[11] P. Connally, K. Li, and G. W. Irwin, "Two applications of eng-genes bases nonlinear identification," presented at the 16th IFAC World Congr., Prague, Czech Republic, Jul. 3–8, 2005.

[12] M. J. Er, W. Chen, and S. Wu, "High-speed face recognition based on discrete cosine transform and RBF neural networks," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 679–691, May 2005.

[13] S. S. Ge, C. C. Hang, T. H. Lee, and T. Zhang, *Stable Adaptive Neural Network Control*. Boston, MA: Kluwer, 2001.

[14] S. S. Ge, F. Hong, and T. H. Lee, "Adaptive neural control of nonlinear time-delay systems with unknown virtual control coefficients," *IEEE Trans. Syst., Man, Cybern. B*, vol. 34, no. 1, pp. 499–516, Feb. 2003.

[15] J. Gonzalez, I. Rojas, J. Ortega, H. Pomares, F. J. Fernandez, and A. F. Diaz, "Multiobjective evolutionary optimization of the size, shape, and position parameters of radial basis function networks for function approximation," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1478–1495, Nov. 2003.

[16] X. Hong and S. A. Billings, "Givens rotation based fast backward elimination algorithm for RBF neural network pruning," *Proc Inst. Elect. Eng. D, Contr. Theory Applicat.*, vol. 144, pp. 381–384, 1997.

[17] X. Hong, C. Harris, M. Brown, and S. Chen, "Backward elimination methods for associative memory network pruning," *Int. J. Hybrid Intell. Syst.*, vol. 1, no. 2, pp. 90–99, 2004.

[18] X. Hong, C. J. Harris, S. Chen, and P. M. Sharkey, "Robust nonlinear model identification methods using forward regression," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 33, no. 4, pp. 514–523, Jul. 2003.

[19] D. S. Huang, "The united adaptive learning algorithm for the link weights and the shape parameters in rbfn for pattern recognition," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 11, no. 6, pp. 873–888, 1997.

[20] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 57–67, Jan. 2005.

[21] N. B. Karayiannis, "Reformulated radial basis neural networks trained by gradient descent," *IEEE Trans. Neural Netw.*, vol. 10, no. 3, pp. 657–671, May 1999.

[22] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 79–88, Jan. 2003.

[23] K. Li, J. Peng, and E. W. Bai, "A two-stage algorithm for identification of nonlinear dynamic systems," *Automatica*, vol. 42, no. 7, pp. 1189–1197, 2006.

[24] K. Li, J. Peng, and G. W. Irwin, "A fast nonlinear model identification method," *IEEE Trans. Autom. Control*, vol. 50, no. 8, pp. 1211–1216, Aug. 2005.

[25] Y. Li, S. Qiang, X. Zhuang, and O. Kaynak, "Robust and adaptive backstepping control for nonlinear systems using RBF neural networks," *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 693–701, Sep. 2004.

[26] L. Ljung, *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[27] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *SIAM J. Appl. Math.*, vol. 11, pp. 431–441, 1963.

[28] S. McLoone, M. D. Brown, G. W. Irwin, and G. Lightbody, "A hybrid linear/nonlinear training algorithm for feedforward neural networks," *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 669–684, Jul. 1998.

[29] R. Neruda and P. Kudova, "Learning methods for radial basis function networks," *Future Gen. Comput. Syst.*, vol. 21, pp. 1131–1142, 2005.

[30] M. J. L. Orr, "Regularization on the selection of radial basis function centers," *Neural Comput.*, vol. 7, pp. 606–623, 1995.

[31] Y. J. Oyang, S. C. Hwang, Y. Y. Ou, C. Y. Chen, and Z. W. Chen, "Data classification with radial basis function networks based on a novel kernel density estimation algorithm," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 225–236, Jan. 2005.

[32] C. Panchapakesan, M. Palaniswami, D. Ralph, and C. Manzie, "Effects of moving the centers in an RBF network," *IEEE Trans. Neural Netw.*, vol. 13, no. 6, pp. 1299–1307, Nov. 2002.

[33] J. Park and I. Sandberg, "Universal approximation using radial-basis function networks," *Neural Comput.*, vol. 3, pp. 246–257, 1991.

[34] H. Peng, T. Ozaki, V. Haggan-Ozaki, and Y. Toyoda, "A parameter optimization method for radial basis function type models," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 432–438, Mar. 2003.

[35] M. M. Polycarpou, "Fault accommodation of a class of multivariable nonlinear dynamical systems using a learning approach," *IEEE Trans. Autom. Control*, vol. 46, no. 5, pp. 736–742, May 2001.

[36] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge, U.K.: Cambridge Univ. Press, 1992.

[37] B. Schlkopf, C. J. C. Burges, and A. J. Smola, *Advances in Kernel Methods—Support Vector Learning*. Cambridge, MA: MIT Press.

[38] E. L. Sutanto, J. D. Mason, and K. Warwick, "Mean-tracking clustering algorithm for radial basis function centre selection," *Int. J. Contr.*, vol. 67, pp. 961–977, 1997.

[39] M. Vogt, "Combination of radial basis function neural networks with optimized learning vector quantization," in *Proc. IEEE Int. Conf. Neural Networks (ICNN)*, San Francisco, CA, 1993, vol. 3, pp. 1841–1846.

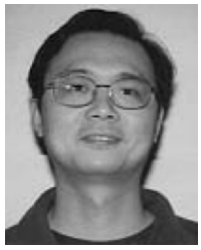
[40] N. Xie and H. Leung, "Blind equalization using a predictive radial basis function neural network," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 709–720, May 2005.

[41] X. Zhang, T. Parisini, and M. M. Polycarpou, "Adaptive fault-tolerant control of nonlinear uncertain systems: An information-based diagnostic approach," *IEEE Trans. Autom. Control*, vol. 49, no. 8, pp. 1259–1274, Aug. 2004.

[42] Q. M. Zhu and S. A. Billings, "Fast orthogonal identification of nonlinear stochastic models and radial basis function neural networks," *Int. J. Contr.*, vol. 64, no. 5, pp. 871–886, 1996.

Jian-Xun Peng received the B.Sc. and M.Sc. degrees from Northwestern Polytechnical University, Xi'an, China, in 1987 and 1992, respectively, and the Ph.D. degree from Queen's University, Belfast, U.K., in 2003.

He is currently a Research Fellow with Queen's University. His main research interests include nonlinear system modeling and identification using gray-box method, neural networks, and genetic algorithms. He is also interested in the development and application of embedded system, real-time computer control, and development of software platform for dynamical modeling and control.



Kang Li (M'05) received the B.Sc. degree from Xiangtan University, Hunan, China, in 1989, the M.Sc. degree from Harbin Institute of Technology, Harbin, China, in 1992, and the Ph.D. degree from Shanghai Jiaotong University, Shanghai, China, in 1995.

He is a Lecturer in intelligent systems and control at Queen's University, Belfast, U.K. He also held various research positions at Shanghai Jiaotong University, Delft University of Technology, Queen's University Belfast, and National University of Singapore. His current research interests are nonlinear system modeling and control using neural networks and genetic algorithms.

Dr. Li is a Chartered Engineer and a Member of InstMC. He is on the Editorial Board of *Neurocomputing* and *Transactions of the Institute of Measurement and Control.*

De-Shuang Huang (SM'98) received the B.Sc. degree from the Institute of Electronic Engineering, China, in 1986, the M.Sc. degree from the National Defense University of Science and Technology, China, in 1989, and the Ph.D. degree from Xidian University, China, in 1993.

During 1993–1997, he was a Postdoctoral Researcher with Beijing Institute of Technology and National Key Laboratory of Pattern Recognition, Chinese Academy of Sciences. In 2000, he joined the Institute of Intelligent Machines, Chinese Academy of Sciences, as Recipient of the "Hundred Talents Program of CAS." From 2000 to 2001, he was a Research Associate with Hong Kong Polytechnic University, China. From 2002 to 2003, he was Research Fellow with City University of Hong Kong. From August to September 2003, he was a Visiting Professor at George Washington University, Washington, DC. From October to December 2003, he was a Research Fellow with Hong Kong Polytechnic University. From July to December 2004, he was a University Fellow with Hong Kong Baptist University, China. He is currently a Visiting Professor at Queen's University of Belfast, U.K. He has published more than 240 papers and two monographs: "Systematic Theory of Neural Networks for Pattern Recognition" and "Intelligent Signal Processing Technique for High Resolution Radars."

Dr. Huang received the Second-Class Prize of the Eighth Excellent High Technology Books of China.