UKACC International Conference on Control 2012
Cardiff, UK, 3-5 September 2012

An Improved Conjugate Gradient Algorithm for Radial Basis Function (RBF) Networks Modelling

Long Zhang and Kang Li
School of Electronics, Electrical Engineering
and Computer Science,
Queen's University Belfast, UK
Email: lzhang14@qub.ac.uk,
k.li@qub.ac.uk

Shujuan Wang
School of Electrical Engineering and Automation,
Harbin Institute of Technology, China,
Email: wsj603@hit.edu.cn

***Abstract*—This paper proposes a new nonlinear optimization algorithm for the construction of radial basis function (RBF) networks in modelling nonlinear systems. The main objective is to speed up the learning convergence of the conventional conjugate gradient method. All the hidden layer parameters of RBF networks are simultaneously optimized by the conjugate gradient method while the output weights are adjusted accordingly using the orthogonal least squares (OLS) method. The derivatives used in the conjugate gradient algorithm are efficiently computed using a recursive sum squared error criterion. Numerical examples show that the new method converges faster than the previously proposed continuous forward algorithm (CFA).**

I. Introduction

The Radial Basis Function (RBF) networks are well known for strict interpolation for approximating scattered data in multi-dimensions [1]. It has been proved that a RBF network can approximate any multi-variate continuous function if a sufficient number of RBF nodes are provided [2]. The RBF networks have been successfully applied in nonlinear system identification [3], signal processing [4] and fault diagnosis [5]. A standard RBF network consists of a nonlinear hidden layer and a linear output layer. The training of such RBF networks involves the optimization of hidden layer parameters (centers and widths) and linear output weights, with the aim of minimizing the cost function like sum squared error (SSE) or Akaike information criterion (AIC) [6].

A variety of methods for training RBF networks have been proposed. Unsupervised clustering methods and the supervised least squares based subset selection methods are widely used. Though these methods, like k-means clustering [7], orthogonal least squares (OLS) [8] and the fast recursive algorithm (FRA) [9], are fast and powerful techniques, they may not build a compact model. To address this problem, methods combining the OLS with evolutionary algorithms [10], [11] have been proposed. However, they are often computationally expensive and suffer from slow convergence due to their random search nature [12].

Alternative solutions are gradient based methods, like conjugate gradient and Newton algorithms [13]. However, they treat all the hidden layer parameters and output weights separately without the consideration of the correlation between these two sets of parameters, therefore, they may converge slowly or not at all if the initial guess is far from the minimum. To speed up the convergence and simplify the computational complexity, the continuous forward algorithm (CFA) [14] has been proposed recently. This method employs the conjugate gradient algorithm to optimize the hidden layer parameters while the optimal output weights are transformed into a set of dependant parameters based on the least squares method, thus without explicit optimizing the output weights. The CFA can be considered as a stepwise algorithm as it optimizes one hidden node at a time. Its advantage is that the model parameters and model size can be determined simultaneously and the computational complexity is not high for each iteration of optimization. The disadvantage is that the learning convergency can be slow and the parameters are not necessarily optimal as the previously obtained parameters are fixed and become the constraints when optimizing the new hidden layer nodes. In other words, it does not optimize all the hidden layer parameters simultaneously.

In this paper, a conjugate gradient method combined with OLS approach is introduced to construct RBF networks, in which all the hidden layer parameters are optimized simultaneously by the conjugate gradient method while the output weights are adjusted accordingly using the OLS method in the continuous space, leading to an improved model performance with fast learning convergence. This is achieved effectively using a recursive sum squared error (SSE) and all the derivatives are updated recursively at each iteration. A numerical example confirms that the new method could converge faster than the CFA.

II. Problem formulation and preliminaries

A RBF network for modelling a nonlinear system can be formulated as [14]

$$y(t) = \sum_{i=1}^{m} \mathbf{p}_i(\mathbf{x}(t), d_i, \mathbf{s}_i)\theta_i + \xi(t) \tag{1}$$

where $\{\mathbf{x}(t), y(t)\}$ are the system input and output variables at time instant t. $\mathbf{x}(t)$ is of assumed known dimension of q, $t = 1, 2, \dots, N$, N being the size of the training data set. $\mathbf{p}_i(\mathbf{x}(t), d_i, \mathbf{s}_i)$ denotes the RBF network function of the ith hidden node with the width $d_i \in \Re^1$ and centers $\mathbf{s}_i \in \Re^q$, here $i = 1, 2, \dots, m$, m being the number of RBF nodes. θ_i is the weight parameter. $\xi(t)$ is a model residual sequence. For a RBF network, all adjustable parameters are the hidden layer parameters and output weights. More specifically, all the hidden layer parameters can be expressed as

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_m \end{bmatrix} = \begin{bmatrix} d_1 & s_{11} & s_{12} & \dots & s_{1q} \\ d_2 & s_{21} & s_{22} & \dots & s_{2q} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d_m & s_{m1} & s_{m2} & \dots & s_{mq} \end{bmatrix} \tag{2}$$

where the vector $\mathbf{v}_i$ includes the i_{th} hidden layer node parameters. d_i and the set $\{s_{i1}, ..s_{iq}\}$ are the width and centers of the i_{th} hidden layer node, respectively. For simplicity, $\mathbf{v}_i = [d_1, s_{i1}, \dots, v_{iq}]$ can be denoted as $\mathbf{v}_i = [v_{i1}, \dots, v_{ir}]$ where $r = q + 1$.

Equation (1) can be expressed in the matrix form as

$$\mathbf{y} = \mathbf{P}\boldsymbol{\Theta} + \boldsymbol{\Xi} \tag{3}$$

where $\mathbf{y} = [y(1), \dots, y(N)]^T$ is the output vector, $\boldsymbol{\Theta} = [\theta_1, \dots, \theta_m]^T$ is the unknown parameter vector, $\boldsymbol{\Xi} = [\xi(1), \dots, \xi(N)]^T$ is the residual vector, and $\mathbf{P} = [\mathbf{p}_1, \dots, \mathbf{p}_m]$ is a N-by-m matrix with $\mathbf{p}_j = [p_j(\mathbf{x}(1), \mathbf{v}_j), \dots, p_j(\mathbf{x}(N), \mathbf{v}_j)]^T$. The RBF network modeling aims to minimize the sum squared error (SSE)

$$\boldsymbol{\Xi}^T\boldsymbol{\Xi} = (\mathbf{y} - \mathbf{P}\boldsymbol{\Theta})^T(\mathbf{y} - \mathbf{P}\boldsymbol{\Theta}) \tag{4}$$

by optimizing d_i's, $\mathbf{s}_i$'s and θ_i's.

If all the hidden layer parameters are determined, the output weights can then be obtained by least squares methods. The orthogonal least squares (OLS) is the most popular algorithm for determining both the linear parameters and model structure. It computes the output weights using the orthogonal decomposition given by [15]

$$\mathbf{P} = \mathbf{W}\mathbf{A} \tag{5}$$

where $\mathbf{A}$ is a m-by-m triangular matrix,

$$\mathbf{A} = \begin{bmatrix} 1 & \alpha_{12} & \dots & \alpha_{1m} \\ 0 & 1 & \dots & \alpha_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

and $\mathbf{W} = [\mathbf{w}_1, ..., \mathbf{w}_m]$ is a N-by-m matrix with orthogonal columns $\mathbf{w}_i$,

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ w_{N1} & w_{N2} & \dots & w_{Nm} \end{bmatrix} \tag{7}$$

which satisfies

$$\mathbf{W}^T\mathbf{W} = diag\{\mathbf{w}_1^T\mathbf{w}_1, \dots, \mathbf{w}_m^T\mathbf{w}_m\}. \tag{8}$$

The orthogonal decomposition involved in OLS can be carried out by the classic Gram-Schimdt (CGS) method. This computes one column of $\mathbf{A}$ at a time and factorizes $\mathbf{P}$ as follows:

$$\left.\begin{array}{l} \mathbf{w}_1 = \mathbf{p}_1 \\ \left.\begin{array}{l} \alpha_{ik} = \dfrac{<\mathbf{w}_i, \mathbf{p}_k>}{<\mathbf{w}_i, \mathbf{w}_i>}, \ 1 \le i < k \\ \mathbf{w}_k = \mathbf{p}_k - \displaystyle\sum_{i=1}^{k-1} \alpha_{ik}\mathbf{w}_i \end{array}\right\} k = 2, \dots, m \end{array}\right\} \tag{9}$$

The model (3) can thus be expressed as

$$\mathbf{y} = (\mathbf{P}\mathbf{A}^{-1})(\mathbf{A}\boldsymbol{\Theta}) + \boldsymbol{\Xi} = \mathbf{W}\mathbf{g} + \boldsymbol{\Xi} \tag{10}$$

where $\mathbf{g} = [g_1, g_2, \dots, g_m]^T = \mathbf{A}\boldsymbol{\Theta}$ is the orthogonal weight vector. The original model weight vector $\mathbf{g}$ can then be calculated by [8]

$$\mathbf{g} = (\mathbf{W}^{\mathbf{T}}\mathbf{W})^{-1}\mathbf{W}^T\mathbf{y} - (\mathbf{W}^{\mathbf{T}}\mathbf{W})^{-1}\mathbf{W}^T\boldsymbol{\Xi}. \tag{11}$$

and the sum squares of $\mathbf{y}$ is given by

$$\mathbf{y}^T\mathbf{y} = \mathbf{g}^T\mathbf{W}^T\mathbf{W}\mathbf{g} + \boldsymbol{\Xi}^T\boldsymbol{\Xi} + \mathbf{g}^T\mathbf{W}^T\boldsymbol{\Xi} + \boldsymbol{\Xi}^T\mathbf{W}\mathbf{g} \tag{12}$$

If $\boldsymbol{\Xi}$ is a zero mean white sequence and is uncorrelated with $\mathbf{P}$ and all stochastic processes of interest are ergodic, then $\mathbf{W}^T\boldsymbol{\Xi} = 0$ [8]. Hence (11) and (12) can be further simplified to

$$\mathbf{g} = (\mathbf{W}^{\mathbf{T}}\mathbf{W})^{-1}\mathbf{W}^T\mathbf{y}, \tag{13}$$

and

$$\mathbf{y}^T\mathbf{y} = \mathbf{g}^T\mathbf{W}^T\mathbf{W}\mathbf{g} + \boldsymbol{\Xi}^T\boldsymbol{\Xi}, \tag{14}$$

respectively. Finally, the weight θ_i can be computed using backward elimination

$$\left.\begin{array}{l} \theta_m = g_m \\ \theta_i = g_i - \displaystyle\sum_{k=i+1}^{m} \alpha_{ik}\theta_k, i = m-1, \dots, 1 \end{array}\right\} \tag{15}$$

Substituting $\mathbf{g}$ in (10) and (14) using (13), we get

$$\boldsymbol{\Xi} = \mathbf{y} - \mathbf{W}(\mathbf{W}^{\mathbf{T}}\mathbf{W})^{-1}\mathbf{W}^T\mathbf{y}. \tag{16}$$

and

$$\boldsymbol{\Xi}^T\boldsymbol{\Xi} = \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{W}(\mathbf{W}^{\mathbf{T}}\mathbf{W})^{-1}\mathbf{W}^T\mathbf{y}. \tag{17}$$

Using the property in (8), the (17) is converted into a recursive form given by

$$\begin{aligned} \boldsymbol{\Xi}^T\boldsymbol{\Xi} &= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\{\sum_{j=1}^{m} \frac{\mathbf{w}_j\mathbf{w}_j^T}{\mathbf{w}_j^T\mathbf{w}_j}\}\mathbf{y} \\ &= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{R}\mathbf{y} \end{aligned} \tag{18}$$

where $\mathbf{R} = \sum_{j=1}^{m} \frac{\mathbf{w}_j\mathbf{w}_j^T}{\mathbf{w}_j^T\mathbf{w}_j}$ is a recursive matrix.

III. The proposed method

To address the problem of slow convergence in the conventional conjugate gradient method, the CFA [14] considers the coupling relationship between $\mathbf{V}$ and $\mathbf{\Theta}$. It converts $\mathbf{\Theta}$ and the estimation error sequence Ξ into $(\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T\mathbf{y}$ and $\mathbf{y}-\mathbf{P}(\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T\mathbf{y}$ using least-squares method respectively, and then computes the derivatives of the estimated error sequence with respect to $\mathbf{V}$. After the updated $\mathbf{V}$ are determined using derivative information, the output weights $\mathbf{\Theta}$ are recalculated by the least squares methods. However, It optimizes each vector parameter $\mathbf{v}_i, i=1,...,M$ at a time while other parameters $\mathbf{v}_j, j=1,...,M, j\neq i$ being fixed and become the constraints, hence the resultant model may be not optimal, though the computational complexity is significantly reduced in each iteration..

The proposed algorithm not only considers the coupling relationship between $\mathbf{V}$ and $\mathbf{\Theta}$ and but also computes all the hidden layer parameters $\mathbf{V}$ simultaneously. The proposed algorithm first optimizes all the hidden layer parameters in (2) using conjugate gradient algorithm and then adjusts the output weights $\mathbf{\Theta}$ by the OLS method.

The conjugate gradient involves computing the first derivative of the SSE with respective to the hidden layer parameters $\mathbf{V}$, which is given by [16]

$$\nabla(\Xi^T\Xi)=2\left(\frac{\partial\Xi}{\partial\mathbf{V}}\right)^T\Xi \tag{19}$$

where the first order derivative of the error sequence $\frac{\partial\Xi}{\partial\mathbf{V}}$ is also called the Jacobian matrix, given by

$$\left.\begin{aligned}\mathbf{J}=\frac{\partial\Xi}{\partial\mathbf{V}}&=-\left[\frac{\partial\mathbf{R}}{\partial\mathbf{v}_1}\mathbf{y},...,\frac{\partial\mathbf{R}}{\partial\mathbf{v}_m}\mathbf{y}\right]\\ \frac{\partial\mathbf{R}}{\partial\mathbf{v}_k}&=\left[\frac{\partial\mathbf{R}}{\partial v_{k1}},...,\frac{\partial\mathbf{R}}{\partial v_{kr}}\right],k=1,...,m\\ \frac{\partial\mathbf{R}}{\partial v_{ki}}&=\sum_{j=1}^{m}\frac{\partial\left\{\frac{\mathbf{w}_j\mathbf{w}_j^T}{\mathbf{w}_j^T\mathbf{w}_j}\right\}}{\partial v_{ki}},i=1,...,r\end{aligned}\right\} \tag{20}$$

where $\mathbf{v}_k=\{\mathbf{v}_{k1},...,\mathbf{v}_{kr}\}$.

The key step for the conjugate gradient algorithm is the computation of the Jacobian matrix. For each element in the Jacobian matrix shown in (20), the derivative of $\mathbf{w}_k$ with respect to $v_{ki}, k=1,...,m, i=1,...,r$ needs to be computed. The mapping between $\mathbf{w}_k$ and v_{ki} can be obtained from the relations among v_{ki}, $\mathbf{p}_k$ and $\mathbf{w}_k$. Each RBF term $\mathbf{p}_k$ is determined by a vector $\mathbf{v}_k=[v_{k1},...,v_{kr}]$. Further, (9) indicates $\mathbf{w}_k$ is dependant on $\mathbf{p}_1,...,\mathbf{p}_k$ and independent of $\mathbf{p}_{k+1},...,\mathbf{p}_m$, hence,

$$\frac{\partial\mathbf{w}_k}{\partial v_{ij}}=\begin{cases}0, & k<i, j=1,...,r\\ \dfrac{\partial\mathbf{w}_k}{\partial v_{ij}}, & k\geq i, j=1,...,r\end{cases} \tag{21}$$

and thus, the element in (20) is simplified as

$$\frac{\partial\mathbf{R}}{\partial v_{ki}}=\sum_{j=1}^{m}\frac{\partial\left\{\frac{\mathbf{w}_j\mathbf{w}_j^T}{\mathbf{w}_j^T\mathbf{w}_j}\right\}}{\partial v_{ki}}=\sum_{j=k}^{m}\frac{\partial\left\{\frac{\mathbf{w}_j\mathbf{w}_j^T}{\mathbf{w}_j^T\mathbf{w}_j}\right\}}{\partial v_{ki}},i=1,..,r \tag{22}$$

while $k=m$, (22) becomes

$$\frac{\partial\mathbf{R}}{\partial v_{mi}}=\frac{\partial\left\{\frac{\mathbf{w}_m\mathbf{w}_m^T}{\mathbf{w}_m^T\mathbf{w}_m}\right\}}{\partial v_{mi}},i=1,..,r \tag{23}$$

Compare (22) and (23), and it is clear that to compute the derivative with respect to the last term parameters $\mathbf{v}_m=[v_{m1},...,v_{mi}]$ only $\frac{\partial\mathbf{w}_m}{\partial\mathbf{w}_{mi}}, i=1,...,r$, is needed while for other derivatives, say $\mathbf{v}_k=[v_{k1},...,v_{ki}]$, only $\frac{\partial\mathbf{w}_k}{\partial\mathbf{v}_{ki}},...,\frac{\partial\mathbf{w}_m}{\partial\mathbf{w}_{ki}}, i=1,...,r$, need to be computed. To simplify the computational complexity for derivatives, each term except the last one is moved to the last position and its derivatives is then computed as proposed in [12]. If any term position is changed, the orthogonal decomposition procedure needs to be repeated. Suppose $\mathbf{p}_k, k=1,...,m-1$, is to be moved to the last position. This leads to shifting the columns $\mathbf{p}_{k+1},...,\mathbf{p}_m$, left by one place. This is given by

$$\{\mathbf{p}_1,...,\mathbf{p}_k,...,\mathbf{p}_m\}\rightarrow\{\mathbf{p}_1,...,\mathbf{p}_{k-1},\mathbf{p}_{k+1},...\mathbf{p}_m,\mathbf{p}_k\} \tag{24}$$

and consequently their corresponding orthogonal basis is changed. If $k=1$, then all the orthogonal terms $[\mathbf{w}_1',...,\mathbf{w}_m']$ need to be computed using (9). If $k>1$, the terms $[\mathbf{w}_1,...,\mathbf{w}_{k-1}]$ are left unchanged and only $[\mathbf{w}_k',...,\mathbf{w}_m']$ need to alter. In details, shift the position by

$$\left.\begin{aligned}\mathbf{p}_j'&=\mathbf{p}_{j+1},\ k\leq j<m\\ \mathbf{p}_m'&=\mathbf{p}_k\end{aligned}\right\} \tag{25}$$

and then re-decompose $\mathbf{p}_k',...,\mathbf{p}_m'$ by

$$\left.\begin{aligned}\alpha_{nj}'&=\frac{<\mathbf{w}_n,\mathbf{p}_k'>}{<\mathbf{w}_n,\mathbf{w}_n>},\ 1\leq n<k\\ \alpha_{nj}'&=\frac{<\mathbf{w}_n',\mathbf{p}_k'>}{<\mathbf{w}_n',\mathbf{w}_n'>},\ k\leq n<j\\ \mathbf{w}_j'&=\mathbf{p}_j'-\sum_{n=1}^{k-1}\alpha_{nj}'\mathbf{w}_n-\sum_{n=k}^{j-1}\alpha_{nj}'\mathbf{w}_n'\end{aligned}\right\} j=k,...,m. \tag{26}$$

The derivative with respect to the last term $\mathbf{p}_m$ in Jacobian matrix is calculated by

$$\left.\begin{aligned}\frac{\partial\mathbf{p}_m}{\partial v_{mi}}&=\left[\frac{\partial p_m(x(1))}{\partial v_{mi}},...,\frac{\partial p_m(x(N))}{\partial v_{mi}}\right]\\ \frac{\partial\mathbf{w}_m}{\partial v_{mi}}&=\frac{\partial\mathbf{p}_m}{\partial v_{mi}}-\sum_{j=1}^{j=m-1}\frac{\mathbf{w}_j^T\frac{\partial\mathbf{p}_m}{\partial v_{mi}}}{\mathbf{w}_j^T\mathbf{w}_j}\mathbf{w}_j\\ J_{mi}&=-\frac{\partial\mathbf{R}}{\partial v_{mi}}\mathbf{y}\\ &=-\frac{\frac{\partial\mathbf{w}_m}{\partial v_{mi}}(\mathbf{w}_m^T\mathbf{y})+\mathbf{w}_m(\frac{\partial\mathbf{w}_m^T}{\partial v_{mi}}\mathbf{y})}{\mathbf{w}_m^T\mathbf{w}_m}\\ &\quad+2\frac{\mathbf{w}_m(\mathbf{w}_m^T\mathbf{y})(\frac{\partial\mathbf{w}_m^T}{\partial v_{mi}}\mathbf{w}_m)}{(\mathbf{w}_m^T\mathbf{w}_m)^2}\end{aligned}\right\} i=1,...,r \tag{27}$$

Other terms $\mathbf{p}_k, k = 1, ..., m-1$, moved to the last position also use the formula (27) where $\frac{\partial \mathbf{p}'_m}{\partial \mathbf{v}'_{mi}}$ equals $\frac{\partial \mathbf{p}_k}{\partial \mathbf{v}_{ki}}$. To compute $\frac{\partial \mathbf{p}_k}{\partial \mathbf{v}_{ki}}$, the basis function $\mathbf{p}_k = [p_k(\mathbf{x}(1), \mathbf{v}_k), \ldots, p_k(\mathbf{x}(N), \mathbf{v}_k)]$ should be given. In this paper, the Gaussian function is considered

$$p_k(\mathbf{x}(t), \mathbf{v}_k) = p_k(\mathbf{x}(t), d_k, \mathbf{s}_k) = \exp(-\eta) \tag{28}$$

where $\eta = \sum_{i=1}^{i=q}(\frac{x_i - s_{ki}}{d_k})^2$, $t = 1, \ldots, N$ and $k = 1, \ldots, m$. The first-order partial derivatives with respect to widths and centers are

$$\frac{\partial p_k}{\partial d_k} = \frac{2}{d_k}\eta\exp(-\eta) \tag{29}$$

and

$$\frac{\partial p_k}{\partial s_{ki}} = \frac{2}{d_k^2}(x_i - s_{ki})\exp(-\eta) \tag{30}$$

respectively.

The procedure of the new method is summarized as follows:

Step 1: Initialize all the hidden layer parameters $\mathbf{V}$ randomly or using OLS based subset selection method [15], [17]. It should be pointed out that to choose appropriate values could speed up the model convergence.

Step 2: Compute Jacobian and Hessian matrices using (25)-(27).

Step 3: Update the $\mathbf{V} = \mathbf{V} + \Delta\mathbf{V}$ and construct the basis functions terms $\mathbf{P}$ in the form of (28) using the new parameters $\mathbf{V}$, and then compute the SSE using (9) and (18).

Step 4: Repeat step 3 until the SSE is reduced to some error target or the iteration number reaches a given number. After the hidden layer parameters $\mathbf{V}$ are determined, the output weights are calculated by back elimination (15).

IV. A Numerical Example

To verify the efficacy of the new method, a numerical examples is used to test its convergence rate. The comparison with CFA will be also discussed. All the tests were carried out using MATLAB R2010a on a desktop Intel E8400 PC with Windows XP system.

The following nonlinear system model [3]

$$\begin{aligned} y(t) = &-0.6377y(t-1) + 0.07298y(t-2) \\ &+ 0.03597u(t-1) + 0.06622u(t-2) \\ &+ 0.06568u(t-1)y(t-1) + 0.02357u^2(t-1) \\ &+ 0.05939 \end{aligned} \tag{31}$$

Here t denotes the time series and u and y represent the system input and output, respectively. A data sequence of length 500 was generated for training, with input u being uniformly distributed within $[-1, 1]$.

For comparison, the CFA was also used to model the nonlinear system. For a fair comparison, these two methods used the same initialization procedure. The iteration stops when the SSE reduction rate is less than 0.01. Fig. 1, Fig. 2 and Fig. 3 illustrate the training curves for the CFA and new methods with model size of 5, 6 and 7 separately. These results are averages over 100 different trials. It is shown that the new method converges faster than the CFA.

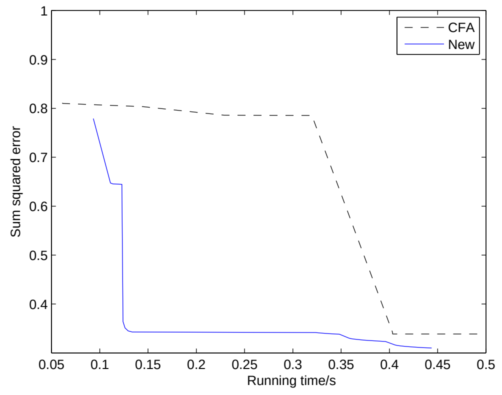

Fig. 1. Variation in training SSE for CFA and the new method with size 5

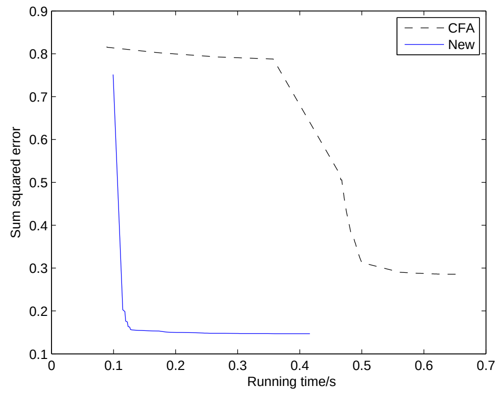

Fig. 2. Variation in training SSE for CFA and the new method with size 6

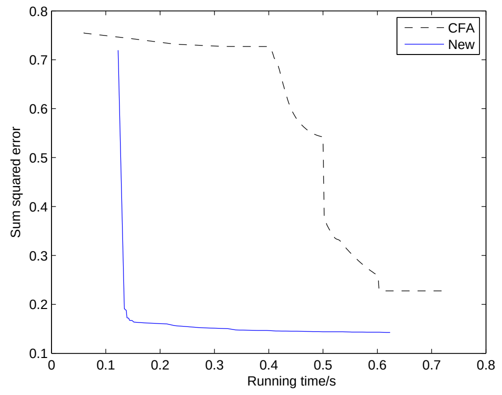

Fig. 3. Variation in training SSE for CFA and the new method with size 7

V. Conclusion

An improved conjugate gradient algorithm for constructing RBF models has been proposed. Unlike previously proposed CFA method which optimizes one hidden node parameters at a time, the new method optimizes all the RBF parameters simultaneously, leading to a faster convergence rate. A numerical example has confirmed the effectiveness of the proposed method.

Acknowledgment

This work was supported by the RCUK under grant EP/G042594/1 and EP/G059489/1, and partially supported by the Chinese Scholarship Council, National Natural Science Foundation of China under grant 51077022, and Science and Technology Commission of Shanghai Municipality under Grant 11ZR1413100.

References

[1] M. Buhmann, *Radial basis functions: theory and implementations*. Cambridge University Press, 2003.

[2] W. A. Light, "Some aspects of radial basis function approximation," *Approximation Theory, Spline Functions and Applications*, vol. 356, pp. 163–190, 1992.

[3] G. Zheng and S. Billings, "Radial basis function network configuration using mutual information and the orthogonal least squares algorithm," *Neural Networks*, vol. 9, no. 9, pp. 1619–1637, 1996.

[4] S. Chen, C. Cowan, and P. Grant, "Orthogonal least squares algorithm for radial basis funtion networks," *International Journal of Control*, vol. 2, no. 2, pp. 302–309, 1991.

[5] K. Narendra, V. Sood, K. Khorasani, and R. Patel, "Application of a radial basis function (rbf) neural network for fault diagnosis in a hvdc system," *Power Systems, IEEE Transactions on*, vol. 13, no. 1, pp. 177–183, 1998.

[6] H. Akaike, "A new look at the statistical model identification," *IEEE Transaction on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974.

[7] S. Billings, H. Wei, and M. Balikhin, "Generalized multiscale radial basis function networks," *Neural Networks*, vol. 20, no. 10, pp. 1081–1094, 2007.

[8] S. Billings, S. Chen, and M. Korenberg, "Identification of mimo non-linear systems using a forward-regression orthogonal estimatorn," *International Journal of Control*, vol. 49, no. 6, pp. 2157–2189, 1989.

[9] K. Li, J. Peng, and G. W. Irwin, "A fast nonlinear model identification method," *IEEE Transcations on Automatic Control*, vol. 8, no. 50, pp. 1211–1216, 2005.

[10] S. Billings and Y. Yang, "Identification of the neighborhood and ca rules from spatio-temporal ca patterns," *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, vol. 33, no. 2, pp. 332–339, 2003.

[11] K. Mao and S. Billings, "Algorithms for minimal model structure detection in nonlinear dynamic system identification," *IEEE Transcations on Neural Networks*, vol. 68, no. 2, pp. 311–330, 1997.

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

[13] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[14] J. Peng, K. Li, and G. W. Irwin, "A novel continous forward algorithm for rbf neural network," *IEEE Transactions on Automatic control*, vol. 52, no. 1, pp. 117–122, 2007.

[15] S. Chen, S. Billing, and W. Luo, "Orthogonal least squares methods and their application to non-linear system identification," *International Journal of Control*, vol. 50, no. 5, pp. 1873–1896, 1989.

[16] M. Hagan and M. Menhaj, "Training feedforward networks with the marquardt algorithm," *Neural Networks, IEEE Transactions on*, vol. 5, no. 6, pp. 989–993, 1994.

[17] J. Peng, K. Li, and D. Huang, "A hybrid forward algorithm for rbf neural network construction," *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1439–1451, 2006.