Monograph:
Fung, Chi. F., Billings, S.A. and Luo, Wan. (1994) On-Line Supervised Adaptive Training Using Radial Basis Function Networks. Research Report. ACSE Research Report 554 . Department of Automatic Control and Systems Engineering

On-Line Supervised Adaptive Training Using Radial Basis Function Networks

CHI F. FUNG, STEVE A. BILLINGS AND WAN LUO

Department of Automatic Control and Systems Engineering
University of Sheffield, P. O. Box 600, Mappin Street, Sheffield S1 4DU
United Kingdom

Telephone: +44(0)742 768555
Facsimile: +44(0)742 731729

Research Report 554

2 December, 1994

On-Line Supervised Adaptive Training Using Radial Basis Function Networks

CHI F. FUNG, STEVE A. BILLINGS AND WAN LUO

Department of Automatic Control and Systems Engineering
University of Sheffield, P. O. Box 600, Mappin Street, Sheffield S1 4DU
United Kingdom

Telephone: +44(0)742 768555
Facsimile: +44(0)742 731729

2 December, 1994

Abstract A new recursive supervised training algorithm is derived for the radial basis neural network architecture. The new algorithm combines the procedures of on-line candidate regressor selection with the conventional Givens QR based recursive parameter estimator to provide efficient adaptive supervised network training. A new concise on-line correlation based performance monitoring scheme is also introduced as an auxiliary device to detect structural changes in temporal data processing applications. Practical and simulated examples are included to demonstrate the effectiveness of the new procedures.

Keywords: *radial basis function network, neural network learning algorithm, parameter estimation, adaptive filtering, system identification, dynamical system modelling, model selection, pattern recognition.*

1. Introduction

Neural networks are an exciting area of research with a range of potential applications which can broadly be categorised into pattern recognition [23] type problems and temporal data modelling or processing. Although many architectures and training algorithms are available radial basis function (RBF) networks [26] are becoming increasingly popular because of the distinctive properties of best approximation, simple network structure and efficient training procedures.

One of the most important issues associated with the use of RBF networks involves the selection of the network specification and the design of the network parameter updating strategy. The former concerns the choice of network input dimension, the selection of which RBF nonlinearity to employ and the number of centres in the hidden layer. The latter relates to the design of an effective algorithm to determine the centres, the connection weights, and if appropriate, the width parameters of the network.

For a RBF network with a static structure network learning can be achieved by training the centres and the weights separately in two consecutive stages. The centres are usually randomly initialised and subsequently adjusted iteration-by-iteration using some form of clustering method. Based on these updated centres, a standard parameter estimation routine can then be applied to compute the network weights [8][9]. With this network parameter updating strategy, the centres and weights are updated by unsupervised and supervised learning, respectively. The advantage in adopting this hybrid strategy is that the computational requirement is substantially reduced compared with the wholly supervised method. However, examination of the structure of the network model often suggests that the inclusion of all the hidden layer outputs in the weight updating procedure is neither efficient nor necessary.

Although numerous efficient algorithms have been developed specially dedicated to networks with static structure [9] the question of finding an optimal, or at least a sub-optimal, subset of centres from the full set of centres has rarely been addressed. From the point of view of system modelling a complete ignorance of this question implies an over-sized network may have been constructed. The consequence of this is not only a definite increase in the demand for processing power but, perhaps, more importantly, a dramatic deterioration in generalisation performance since this violates the principle of model parsimony. This is particularly relevant when an on-line scheme is applied to a large scale time-varying system where any procedures which avoid an over-fitted model with minimal computational complexity would certainly be welcome.

In the present study a new supervised training algorithm for RBF neural networks is derived. By observing the stepwise increments in variance that each individual regressor contributes at each time step the current structure of the network architecture or model is adaptively adjusted and a subset of centres is selected at a reasonable computational cost. The new algorithm combines the procedures of selecting candidate regressors on-line with a Givens QR based recursive least squares routine to provide

adaptive supervised on-line network learning that can be applied for both pattern recognition and time domain applications. Issues of performance monitoring are considered in the context of temporal data modelling and new concise correlation based test procedures are introduced to provide an on-line structure monitoring scheme and to detect inadequately trained networks. Both practical and simulated examples are included to demonstrate the new algorithm.

2. NARMAX Representation for Non-linear Systems

The study of non-linear systems plays an important role in data modelling because most physical phenomena encountered in practical applications are likely to exhibit some degree of non-linearity. An appropriate functional representation is therefore necessary in the modelling of non-linear systems. As a natural extension of the linear difference equation in non-linear system modelling the *non-linear autoregressive moving average model with exogenous inputs* or NARMAX model is often employed.

Given the $(mN_u + nN_y + nN_e)$-dimensional vector $\tilde{\mathbf{x}}(k)$ at time step k

$$\tilde{\mathbf{x}}(k) \triangleq \left[\mathbf{u}^{\mathrm{T}}(k-1) \;\cdots\; \mathbf{u}^{\mathrm{T}}(k-N_u) \;\; \mathbf{y}^{\mathrm{T}}(k-1) \;\cdots\; \mathbf{y}^{\mathrm{T}}(k-N_y) \;\; \mathbf{e}^{\mathrm{T}}(k-1) \;\cdots\; \mathbf{e}^{\mathrm{T}}(k-N_e)\right]^{\mathrm{T}}, \quad (1)$$

where $\mathbf{u}(k) \in \mathrm{R}^m$, $\mathbf{y}(k) \in \mathrm{R}^n$ and $\mathbf{e}(k) \in \mathrm{R}^n$ are the system input, system output and noise with maximum lags N_u, N_y, N_e at time step k, respectively, the m-input, n-output non-linear stochastic system described by the functional relationship

$$\mathbf{y}(k) = \mathbf{f}(\tilde{\mathbf{x}}(k)) + \mathbf{e}(k), \quad (2)$$

where $\mathbf{f}(\cdot) \in \mathrm{R}^n$ is a differentiable real vector valued function, is generally referred to as the NARMAX representation [7].

It is obvious that the model described in Equation (2) is, in fact, a non-linear regression model with the system output regressed on all the terms involved in the argument of the function $\mathbf{f}(\cdot)$ under the disturbance of the noise vector $\mathbf{e}(k)$. Clearly the choice of the form of the functional map $\mathbf{f}(\cdot)$ determines the behaviour of the system dynamics. In general, $\mathbf{f}(\cdot)$ is a highly sophisticated non-linear function although various forms may exist. Exact representation of $\mathbf{f}(\cdot)$ is usually difficult to obtain and an approximation is often used instead. In fact, the interpretation of the NARMAX model can be regarded as a generalisation of a wide class of some commonly employed non-linear models [21]. Examples such as the Hammerstein, Wiener, bilinear and Volterra models can all be shown to be special cases of the NARMAX model. In the present study, the RBF network will be used as an approximator $\hat{\mathbf{f}}(\cdot) \in \mathrm{R}^n$ to approximate the function $\mathbf{f}(\cdot)$ so that the predicted system output $\hat{\mathbf{y}}(k) \in \mathrm{R}^n$ at the RBF network output yields

$$\hat{\mathbf{y}}(k) = \hat{\mathbf{f}}(\mathbf{x}(k)) \quad (3)$$

where $\mathbf{x}(k) \in \mathrm{R}^{N_i}$ is the input vector given by

$$\mathbf{x}(k)\underline{\Delta}\left[\mathbf{u}^{\mathrm{T}}(k-1)\ \cdots\ \mathbf{u}^{\mathrm{T}}(k-N_u)\ \ \mathbf{y}^{\mathrm{T}}(k-1)\ \cdots\ \mathbf{y}^{\mathrm{T}}(k-N_y)\ \ \boldsymbol{\varepsilon}^{\mathrm{T}}(k-1)\ \cdots\ \boldsymbol{\varepsilon}^{\mathrm{T}}(k-N_\varepsilon)\right]^{\mathrm{T}} \quad (4)$$

and $\boldsymbol{\varepsilon}(k)\in\mathrm{R}^n$ is the innovation vector with maximum lag N_ε at time step k.

3. RBF Representation of The NARMAX Model

As depicted in Figure 1, a RBF network consists of only one hidden layer and one output layer. The entire hidden layer is composed of an array of non-linear processing units and, where appropriate, each of these units is associated with a positive scalar value known as the width parameter. For an N_i-input, n-output network with N_c hidden units, at time step k, the i-th $(1\le i\le N_c)$ hidden unit performs a non-linear functional map

$$\phi:\ \underbrace{\mathrm{R}^{N_i}}_{\text{network input}}\times\underbrace{\mathrm{R}^{N_i}}_{\text{centre}}\times\underbrace{\mathrm{R}_+}_{\text{width parameter}}\longrightarrow\underbrace{\mathrm{R}}_{\text{hidden layer output}} \quad (5)$$

which is characterised by the current value of the centre $\mathbf{c}_i(k)\in\mathrm{R}^{N_i}$ and the width parameter $\omega_i(k)\in\mathrm{R}$, on the N_i-dimensional input vector $\mathbf{x}(k)\in\mathrm{R}^{N_i}$ to produce the i-th hidden layer output

$$\varphi_i(k)=\phi\left(\left\|\mathbf{x}(k)-\mathbf{c}_i(k)\right\|,\omega_i(k)\right),\ 1\le i\le N_c \quad (6)$$

where $\|\cdot\|$ denotes the 2-norm in N_i-dimensional vector space [6]. The non-linearity $\phi(\cdot)$ generally has a radially symmetric response around the current centre. For the present study, we will be particularly concerned with the case where $\phi(\cdot)$ is specified to be the *thin-plate-spline* function which is defined as

$$\phi(x)\underline{\underline{\Delta}}\,x^2\log_{\mathrm{e}}x. \quad (7)$$

Although various alternative choices [25] of $\phi(\cdot)$ are possible the thin-plate-spline function was chosen because it does not have a width parameter which would inevitably demand extra processing effort but the richness of non-linearity in representing the underlying non-linear dynamics can still be preserved during network updating. With the RBF specified by $\phi(\cdot)$, Equation (6) can be simplified to

$$\varphi_i(k)=\phi\left(\left\|\mathbf{x}(k)-\mathbf{c}_i(k)\right\|\right),\ i=1,2,\cdots,N_c. \quad (8)$$

The output of each node in the output layer is simply a weighted sum of all the outputs from the hidden layer so that the j-th output node is associated with the weight vector estimated at time step k, $\boldsymbol{\theta}_j(k)\equiv\left\{\theta_{ij}(k)\right\}_{i=1}^{N_c}\in\mathrm{R}^{N_c}$ and the j-th network output $\hat{y}_j(k)$ can be expressed as

$$\hat{y}_j(k)=\boldsymbol{\varphi}^{\mathrm{T}}(k)\boldsymbol{\theta}_j(k),\ 1\le j\le n. \quad (9)$$

where $\boldsymbol{\varphi}(k)=\left\{\varphi_i(k)\right\}_{i=1}^{N_c}\in\mathrm{R}^{N_c}$. The response of n-dimensional network response $\hat{\mathbf{y}}(k)$ can be expressed as

$$\hat{\mathbf{y}}(k)=\left\{\boldsymbol{\varphi}^{\mathrm{T}}(k)\boldsymbol{\theta}_j(k)\right\}_{j=1}^{n}. \quad (10)$$

By defining the network output matrix

$$\hat{\mathbf{Y}}(k) \triangleq \begin{bmatrix} \hat{y}_1(k) & \cdots & \hat{y}_n(k) \end{bmatrix} \equiv \begin{bmatrix} \hat{\mathbf{y}}^{\mathrm{T}}(1) \\ \vdots \\ \hat{\mathbf{y}}^{\mathrm{T}}(k) \end{bmatrix} \in \mathrm{R}^{k \times n} \tag{11}$$

the data matrix

$$\Phi(k) \triangleq \begin{bmatrix} \varphi^{\mathrm{T}}(1) \\ \vdots \\ \varphi^{\mathrm{T}}(k) \end{bmatrix} \in \mathrm{R}^{k \times N_c}, \tag{12}$$

and the network weight matrix

$$\Theta(k) \triangleq \begin{bmatrix} \theta_1(k) & \cdots & \theta_n(k) \end{bmatrix} \equiv \begin{bmatrix} \theta_{11}(k) & \cdots & \theta_{1n}(k) \\ \vdots & & \vdots \\ \theta_{N_c 1}(k) & \cdots & \theta_{N_c n}(k) \end{bmatrix} \in \mathrm{R}^{N_c \times n} \tag{13}$$

the n-output network response can be concisely expressed as

$$\hat{\mathbf{Y}}(k) = \Phi(k)\Theta(k) \tag{14}$$

Recent studies [24] suggest that, the rate of convergence in network learning can be enhanced by modifying the network to allow direct contributions to the weighted sums at the network output nodes from direct linear links. In addition, it is advantageous to allow an adjustable dc value. These modifications which are illustrated in Figure 2, can easily be implemented by augmenting the input vector to the output layer $\varphi(k)$ to yield

$$\varphi'(k) = \begin{bmatrix} \varphi(k) \\ \hline \mathbf{x}(k) \\ \hline 1 \end{bmatrix} \begin{matrix} \}N_c \\ \}N_i \\ \}1 \end{matrix} \tag{15}$$

where the unity entry provides the dc link. For notional convenience $\varphi'(k)$ will be re-labelled as $\varphi(k)$ so that now

$$\varphi(k) = \begin{bmatrix} \varphi_1(k) \\ \vdots \\ \varphi_{N_c}(k) \\ \hline \varphi_{N_c+1}(k) \\ \vdots \\ \varphi_{N_c+N_i}(k) \\ \hline \varphi_{N_c+N_i+1}(k) \end{bmatrix} = \begin{bmatrix} \varphi_1(k) \\ \vdots \\ \varphi_{N_c}(k) \\ \hline x_1(k) \\ \vdots \\ x_{N_i}(k) \\ \hline 1 \end{bmatrix}. \tag{16}$$

The network connection weight vector must also be re-defined as

$$\theta_j(k) = \begin{bmatrix} \theta_{1j}(k) \\ \theta_{2j}(k) \\ \vdots \\ \theta_{N_c+N_i+1\,j}(k) \end{bmatrix}, \quad j = 1, 2, \cdots, n. \tag{17}$$

Notice that as a result of the re-definitions of $\varphi(k)$ and $\theta_j(k)$, the respective dimensions have been changed from N_c to

$$N_h \triangleq N_c + N_i + 1. \tag{18}$$

Based on the structural framework described above we now wish to train a RBF network with output response given by Equation (10) to represent the non-linear function $\mathbf{f}(\cdot)$ in the NARMAX model Equation (2). This can be accomplished by assigning the components of the network input node as all the components of the vector $\mathbf{x}(k)$ defined in Equation (4). With these network input node assignments, the input-output relationship of the network at time step k of the estimated model $\hat{\mathbf{f}}(\cdot)$ is therefore given by

$$\hat{\mathbf{y}}(k) = \left\{\varphi^{\mathrm{T}}(k)\theta_i(k)\right\}_{i=1}^{n} = \left\{\sum_{j=1}^{N_c} \phi\left(\left\|\mathbf{x}(k) - \mathbf{c}_j(k)\right\|\right)\theta_{ji}(k) + \sum_{j=1}^{N_i} x_j(k)\theta_{j+N_c,i}(k) + \theta_{N_h i}(k)\right\}_{i=1}^{n}. \tag{19}$$

4. Orthogonal Least Squares Estimator

One of the most commonly used techniques in data modelling is regression analysis based on the classical linear least squares method and variants such as generalised and weighted least squares. The properties of linear least squares have been extensively studied in the literature and the approach has been highly successful in modelling data governed by linear relationships. The desirable properties of linear least squares contributes considerably to the subsequent developments of various non-linear regression techniques such that non-linear systems can be accommodated. An alternative approach to classical least squares is the *orthogonal least squares* (OLS) method which was developed to circumvent numerical problems and to incorporate model structure detection by considering an equivalent auxiliary model in the setting of an orthogonal vector space [2][18].

Following the notation employed in above, the basic idea of the OLS method can be illustrated by considering the i-th $(1 \le i \le n)$ sub-system of an m-input, n-output system. Instead of performing parameter estimation directly on the regression model

$$\mathbf{y}_i(k) = \Phi(k)\theta_i(k) + \mathbf{e}_i(k) \tag{20}$$

the equivalent auxiliary model given by

$$\mathbf{y}_i(k) = \mathbf{W}(k)\mathbf{g}_i(k) + \mathbf{e}_i(k) \tag{21}$$

is considered where

$$\mathbf{W}(k) = \left[\mathbf{w}_1(k) \quad \mathbf{w}_2(k) \quad \cdots \quad \mathbf{w}_{N_h}(k)\right] \text{ with } \mathbf{w}_i^{\mathrm{T}}(k)\mathbf{w}_j(k) = L\delta_{ij},\ \forall i, j = 1,2,\cdots,N_h \tag{22}$$

$\mathbf{g}_i(k) \in \mathrm{R}^{N_h}$, δ_{ij} is the Kornecker delta and $L \in \mathrm{R}$ is a scaling constant. The estimate of the j-th component of the parameter vector $g_{ji}(k)$, $\hat{g}_{ji}(k)$ at time step k can be computed as

$$\hat{g}_{ji}(k) = \frac{\mathbf{y}_i^{\mathrm{T}}(k)\mathbf{w}_j(k)}{\left\|\mathbf{w}_j(k)\right\|^2},\ 1 \le j \le N_h. \tag{23}$$

Therefore, the set of vectors $\{\mathbf{w}_i(k): i = 1,2,\cdots,N_h\}$ effectively forms an orthogonal basis which spans the same space as the space spanned by the set of basis vectors $\{\boldsymbol{\varphi}_i(k): i = 1,2,\cdots,N_h\}$ in the original regression model expressed in Equation (20). Symbolically,

$$\text{span}\{\mathbf{w}_i(k)\}_{i=1}^{N_h} = \text{span}\{\boldsymbol{\varphi}_i(k)\}_{i=1}^{N_h} \tag{24}$$

In geometric terms, the j-th estimated coefficient $\hat{g}_{ji}(k)$ can be interpreted as the magnitude of the orthogonal projection of the observed vector $\mathbf{y}_i(k)$ onto the j-th co-ordinate in the new (orthogonal) vector space.

With the geometrical interpretation described above, the criterion of selecting the most important regressors from the full set of regressors to be included in the auxiliary model can be taken as the proportion of energy contributed by an individual regressor. Such a criterion can be expressed as

$$ERR_i^{(j)} \triangleq \frac{\hat{g}_{ji}\|\mathbf{w}_j(k)\|^2}{\|\mathbf{y}_i(k)\|^2} \times 100\% \tag{25}$$

and is called the *error-reduction-ratio* (ERR) [2] for the j-th component in the auxiliary model at time step k. It is obvious that a large ERR value, say close to 100%, indicates a significant contribution from an important regressor and should therefore be included in the model whereas a small ERR value indicates an unimportant regressor which would tend to make a negligible difference to the overall goodness-of-fit. By including only the most significant regressors in the auxiliary model the effect of multi-collinearity and numerical ill-conditioning will be minimised.

A closely related idea is principal component analysis [20] which is extensively used in multi-dimensional statistical data analysis where reduction in dimensionality of the data set to be analysed is required. In the context of neural networks the OLS learning procedure selects appropriate radial basis function centres as a subset of the training data vectors [15]. At each step of the selection procedure, the increment to the explained variance of the system output is maximised. In this way, the OLS learning procedure produces an RBF network whose hidden layer is smaller than that produced with randomly selected centres. Thus, the OLS learning procedure provides a powerful approach for the construction of a parsimonious RBF network with good numerical properties. By using the ERR value and observing the stepwise increments in variance that each individual regressor may contribute at each time step, the current structure of the network model can be adaptively adjusted and a subset of good centres can be selected at minimal computational cost.

For many real-time applications in parameter estimation, the volume of available data is unrestricted and batch processing based on the entire data set is not realistic. Such situations frequently arise, for instance, in many aspects of adaptive signal processing and control problems. Effective algorithms operating explicitly on a moving window of data in a recursive manner must be employed. Among the various recursive algorithms which are available the QR based method appears to be

remarkably appealing. One implementation of this is the Givens recursive least squares (RLS) procedure [17] which allows simple parallel architectural implementation [13] while preserving superior numerical properties over conventional least squares type estimators.

5. Network Learning Based on *k*-means Clustering and Givens RLS Algorithm

When the function approximation problem is reformulated in terms of RBF network learning, the learning task involves the determination of an appropriate set of RBF centres, width parameters and associated network weights [22]. To avoid non-linear learning, the candidate centres can be selected from the entire set of data. But for many applications the number of all the candidate centres or regressors is usually very large and the inclusion of all these candidates is hardly practical and often unnecessary since a much smaller subset of centres may be sufficient to construct an adequate network. In this respect, the centre selection problem can be regarded as a sub-model selection problem that selects only a small subset of the appropriate centres from the full set of candidate centres. It is apparent that the idea of OLS can easily be applied to select the centres as candidate regressors to be used the subsequent stage of weight calculations.

The procedure of network learning, described in detail below, will therefore consist of two separate stages, centre selection and weight updating. These two sub-tasks can be achieved by means of unsupervised and supervised learning in a recursive manner.

5.1. *k*-Means Centre Clustering

To update the centres, a *k*-means clustering [11] algorithm based on the non-hierarchical clustering methods is employed. In its simplest version, the algorithm directs each item towards the cluster having the nearest centroid. Given that the number of centres is specified, the *k*-means clustering algorithm operates on the network input vector at each time step and successively assigns the centres to the data clusters in the domain of the network input space. After a sufficient number of samples all these centres eventually converge to the data clusters.

Given that all the centres have been updated by a *k*-means clustering algorithm, the Givens forward selection algorithm treats all the hidden layer outputs as candidate centres or regressors in the least squares estimation of the network weights. Only a subset of the most significant hidden layer outputs are selected. Based on the successfully selected candidate centres, the algorithm minimises the mean-square-error between the desired output and the actual network output by performing a series of Givens orthogonal transformations. In this regard, the weight updating sub-task is considered as a supervised learning process [9].

To simplify the exposition of the learning algorithm, consider an N_i-input, single output network with N_c centres initialised to random values in the vicinity of the input space prior to starting the clustering process. Given an initial clustering gain $\kappa(0)$, the k-means clustering algorithm calculates the Euclidean distance $d_i(k)$ between the current network input vector $\mathbf{x}(k) \in \mathrm{R}^{N_i}$ and the i -th centre $\mathbf{c}_i(k) \in \mathrm{R}^{N_i}$ for all the N_c centres at time step k such that

$$d_i(k) = \|\mathbf{x}(k) - \mathbf{c}_i(k)\|, \; i = 1, 2, \cdots, N_c \tag{26}$$

The updating of a centre is based on the criterion of how far the current input vector is away from the previously updated centres measured by the Euclidean distance.

$$\mathbf{c}_j(k) = \mathbf{c}_j(k-1) + \kappa(k)\delta_{ij}\left[\mathbf{x}(k) - \mathbf{c}_j(k-1)\right] \tag{27}$$

where

$$j = \arg \min_{i=1,2,\cdots,N_c} \{d_i(k)\} \tag{28}$$

The clustering gain $\kappa(k) \in (0,1]$ is given by

$$\kappa(k) = \frac{\kappa(k-1)}{\sqrt{1 + \mathrm{int}\left(\frac{k}{N_h}\right)}} \tag{29}$$

where $\mathrm{int}(x)$ denotes the integral part of x. Other choices of updating rule for the clustering gain are possible providing $\kappa(0)$ monotonically decreases to zero. Experience suggests that the choice of initial clustering gain $\kappa(0)$ is not critical and good results are generally obtained with $\kappa(0) \approx 1$.

Based on the computed centre values $\{\mathbf{c}_j(k): j = 1, 2, \cdots, N_c\}$, the i-th hidden layer output $\varphi_i(k)$ can then be computed using Equation (8).

5.2. Givens Forward Selection Estimation Algorithm

Given a set of fixed centres determined by the k-means clustering algorithm, the most significant centres for the i-th $(1 \le i \le n)$ output of the network can be selected and the weight vector $\theta_i(k) \in \mathrm{R}^{N_h}$ can be computed recursively by the Givens algorithm [12] with column pivoting as described below.

The computations of $\theta_i(k)$ for time step k can be performed in two separate stages based on solving the normal equation

$$\left[\Lambda^{1/2}(k)\Phi(k)\right]^{\mathrm{T}}\left[\Lambda^{1/2}(k)\Phi(k)\right]\theta_i(k) = \left[\Lambda^{1/2}(k)\Phi(k)\right]^{\mathrm{T}}\mathbf{y}_i(k). \tag{30}$$

recursively, where

$$\Lambda(k) \triangleq \begin{bmatrix} \lambda^{k-1} & & \mathbf{O} \\ & \lambda^{k-2} & \\ & & \ddots \\ \mathbf{O} & & 1 \end{bmatrix} \equiv \left[\begin{array}{ccc|c} & \lambda\Lambda(k-1) & & \begin{matrix}0\\ \vdots \\ 0\end{matrix} \\ \hline 0 & \cdots & 0 & 1 \end{array}\right]; \tag{31}$$

$\lambda \in (0,1)$ is a forgetting factor in the recursive algorithm and the matrix $\Lambda^{1/2}(k)$ is a Cholskey factor of the matrix $\Lambda(k)$. The same convention of symbols will be used throughout. Detailed analysis (after some algebra) by Skinny decomposition [14] of the matrix $\Lambda^{1/2}(k)\Phi(k)$ into

$$\Lambda^{1/2}(k)\Phi(k) \equiv \mathbf{Q}(k)\mathbf{S}(k) \tag{32}$$

reveals that the system given by Equation (30) can be equivalently solved based on the triangular system

$$\mathbf{R}_i(k)\boldsymbol{\theta}_i(k) = \mathbf{v}_i(k) \tag{33}$$

where $\mathbf{Q}(k) \in \mathbf{R}^{k \times N_h}$ is a matrix whose columns are mutually orthogonal so that $\mathbf{Q}^{\mathrm{T}}(k)\mathbf{Q}(k) \in \mathbf{R}^{N_h \times N_h}$ is strictly diagonal; $\mathbf{S}(k) \in \mathbf{R}^{N_h \times N_h}$ is unit upper triangular;

$$\mathbf{R}_i(k) \underline{\Delta} \left[\mathbf{Q}^{\mathrm{T}}(k)\mathbf{Q}(k)\right]^{1/2}\mathbf{S}(k) \tag{34}$$

is an $N_h \times N_h$ upper triangular matrix and

$$\mathbf{v}_i(k) \equiv \left[v_{i1}(k) \quad v_{i2}(k) \quad \cdots \quad v_{iN_h}(k)\right]^{\mathrm{T}} \underline{\Delta} \left[\mathbf{Q}^{\mathrm{T}}(k)\mathbf{Q}(k)\right]^{-1/2}\mathbf{Q}^{\mathrm{T}}(k)\mathbf{y}_i(k) \tag{35}$$

is an N_h-component vector. Furthermore, Equation (30) can be re-configured into the following artificially augmented upper triangular matrix structure to facilitate concise matrix operations.

$$\left[\begin{array}{c|c} \mathbf{R}_i(k) & \mathbf{v}_i(k) \\ \hline 0 \quad \cdots \quad 0 & * \end{array}\right], \tag{36}$$

where * is a '*don't care*' entry.

Prior to starting the algorithm $\mathbf{R}_i(k)$ and $\mathbf{v}_i(k)$ can be initialised to a diagonal matrix with arbitrarily small identical positive entries $\delta \approx 0$ and an N_h-component zero vector, respectively, i.e.

$$\mathbf{R}_i(0) = \delta\mathbf{I} \tag{37}$$

where $\mathbf{I}$ is an identity matrix with appropriate dimension and

$$\mathbf{v}_i(0) = \underbrace{\left[0 \quad 0 \quad \cdots \quad 0\right]^{\mathrm{T}}}_{N_h} \tag{38}$$

To describe the process of determining the weight vector $\boldsymbol{\theta}_i(k)$ at time step k, it is assumed that the matrix structure has been formed after time step $(k-1)$ as follows:-

$$\left[\begin{array}{c|c} \mathbf{R}_i(k-1) & \mathbf{v}_i(k-1) \\ \hline 0 \quad \cdots \quad 0 & * \end{array}\right] \tag{39}$$

In the first stage of the procedure, subsequent to the determination of the input vector to the output layer $\boldsymbol{\varphi}(k)$, the matrix in (39) is modified to a non-triangular matrix as follows:-

$$\underbrace{\left[\begin{array}{c|c} \mathbf{R}_i(k-1) & \mathbf{v}_i(k-1) \\ \hline 0 \quad \cdots \quad 0 & * \end{array}\right]}_{\text{upper triangular}} \xrightarrow{\text{data arrival}} \underbrace{\left[\begin{array}{c|c} \sqrt{\lambda}\mathbf{R}_i(k-1) & \sqrt{\lambda}\mathbf{v}_i(k-1) \\ \hline \boldsymbol{\varphi}^{\mathrm{T}}(k) & y_i(k) \end{array}\right]}_{\text{non-upper triangular}}. \tag{40}$$

A series of Givens rotations is then applied to the matrix on the right hand side of (40) until an upper triangular matrix is obtained to yield

$$\underbrace{\left[\begin{array}{c|c} \sqrt{\lambda}\mathbf{R}_i(k-1) & \sqrt{\lambda}\mathbf{v}_i(k-1) \\ \hline \boldsymbol{\varphi}^{\mathrm{T}}(k) & y_i(k) \end{array}\right]}_{\text{non-upper triangular}} \xrightarrow{\text{Givens rotations}} \underbrace{\left[\begin{array}{c|c} \mathbf{R}_i^{(0)}(k) & \mathbf{v}_i(k) \\ \hline 0 \quad \cdots \quad 0 & * \end{array}\right]}_{\text{upper triangular}}. \tag{41}$$

The matrix given on the right hand side of (41) can be viewed as an interim stage in achieving the transformed system given in Equation (33) where the matrix $\mathbf{R}_i(k)$ will be completed after $n_h(k)(\le N_h)$ steps in the selection of candidate regressors as

$$\underbrace{\mathbf{R}_i^{(0)}(k)}_{\text{prior to selection}} \longrightarrow \underbrace{\mathbf{R}_i^{(1)}(k)}_{\text{after 1 selection}} \longrightarrow \cdots\cdots \longrightarrow \underbrace{\mathbf{R}_i^{(n_h(k))}(k) \rightarrow \mathbf{R}_i(k)}_{\text{after } n_h(k) \text{ selections}} \tag{42}$$

Based on the matrix on the right hand side of (41) the algorithm calculates $v_{ir}^2(k)$, $\forall r = 1,2,\cdots,N_h$. By examining the transformed system in Equation (33) it is obvious that these $v_{ir}^2(k)$ values are directly related to the relative strengths contributed by all the N_h candidate regressors. A large $v_{ir}^2(k)$ value indicates a significant contribution from the r-th regressor and vice versa. By monitoring the stepwise increments in the error-reduction ratio, defined in Equation (25), at each step of the selection procedure a regressor can be selected or discarded. For the transformed system given by the right hand side of (41), the error-reduction ratio achieved by the r-th regressor can equivalently be taken as

$$ERR_i^{(r)}(k) = \frac{v_{ir}^2(k)}{\|\mathbf{y}_i(k)\|^2}. \tag{43}$$

Define the N_h-component vector $\mathbf{p}_i^{(j)}(k)$, $(1 \le j \le n_h(k) \le N_h)$ as

$$\mathbf{p}_i^{(j)}(k) \triangleq \left[\begin{array}{cccc} p_{i1}^{(j)}(k) & p_{i2}^{(j)}(k) & \cdots & p_{iN_h}^{(j)}(k) \end{array}\right]^{\mathrm{T}} \text{ with } \mathbf{p}_i^{(0)}(k) = \left[\begin{array}{cccc} v_{i1}^2(k) & v_{i2}^2(k) & \cdots & v_{iN_h}^2(k) \end{array}\right]^{\mathrm{T}}. \tag{44}$$

At the j-th selection step, the j-th largest $v_r^2(k)$ $(1 \le r \le N_h)$ is identified in $\mathbf{p}_i^{(j)}(k)$ such that

$$l = \arg \max_{h=j,j+1,\cdots,N_h} \left\{ p_{ih}^{(j-1)}(k) \right\}. \tag{45}$$

The j-th and the l-th columns in the matrix $\mathbf{R}_i^{(j-1)}(k)$ are then exchanged. This can be implemented by post-multiplying by the permutation matrix

$$\boldsymbol{\Pi}_i^{(j)}(k) \triangleq \left[\begin{array}{ccccccc} \mathbf{1}_1 & \cdots & \underset{j\text{th}}{\mathbf{1}_l} & \cdots & \underset{l\text{th}}{\mathbf{1}_j} & \cdots & \mathbf{1}_{N_h} \end{array}\right] \tag{46}$$

where

$$\mathbf{1}_i \triangleq [0 \ \cdots \ 0 \ \underset{i\text{th}}{1} \ 0 \ \cdots \ 0]^{\mathrm{T}} \in \mathrm{R}^{N_h} \tag{47}$$

so that the matrix $\mathbf{R}_i^{(j)}(k)$ yields

$$\mathbf{R}_i^{(j)}(k) = \mathbf{R}_i^{(j-1)}(k)\Pi_i^{(j)}(k) = \mathbf{R}_i^{(0)}(k)\Pi_i^{(1)}(k)\Pi_i^{(1)}(k)\cdots\Pi_i^{(j)}(k). \tag{48}$$

Also, the vector $\mathbf{p}_i^{(j)}(k)$ is updated as

$$\mathbf{p}_i^{(j)}(k) = \Pi_i^{(j)^{\mathrm{T}}}(k)\mathbf{p}_i^{(j-1)}(k) = \Pi_i^{(j)^{\mathrm{T}}}(k)\Pi_i^{(j-1)^{\mathrm{T}}}(k)\cdots\Pi_i^{(1)^{\mathrm{T}}}(k)\mathbf{p}_i^{(0)}(k). \tag{49}$$

The effect of the permutation operation outlined in Equation (49) is to re-arrange the $v_r^2(k)$ values in the vector $\mathbf{p}_i^{(j)}(k)$ such that the first j largest components, whose corresponding regressors which should then be disqualified in the selection after the current selection step, appear in the first j leading positions of the vector $\mathbf{p}_i^{(j)}(k)$. With the permutation operations applied to $\mathbf{R}_i^{(j-1)}(k)$, the resulting matrix $\mathbf{R}_i^{(j)}(k)$ will no longer be upper triangular and a re-triangularization (re-orthogonalization) operation is required so that an equivalent upper triangular matrix $\overline{\mathbf{R}}_i^{(j)}(k)$ can be maintained. This, again, can be readily achieved by performing a series of Givens rotations as

$$\underbrace{\mathbf{R}_i^{(j)}(k)}_{\text{(non-upper triangular)}} \xrightarrow{\text{Givens rotations}} \underbrace{\overline{\mathbf{R}}_i^{(j)}(k)}_{\text{(upper triangular)}} . \tag{50}$$

To align with the rotations on $\mathbf{R}_i^{(j)}(k)$ the same sequence of rotations used in (50) should also be applied to the vector $\mathbf{v}_i^{(j)}(k)$ such that the rotated vector $\overline{\mathbf{v}}_i^{(j)}(k)$ yields

$$\mathbf{v}_i^{(j)}(k) \xrightarrow{\text{Givens rotations}} \overline{\mathbf{v}}_i^{(j)}(k). \tag{51}$$

It is worth noting that as a result of the near upper triangular structure in $\mathbf{R}_i^{(j)}(k)$ the transformation outlined in (50) involves only a sub-matrix of the last $(N_h - j + 1)$ rows and columns in $\mathbf{R}_i^{(j)}(k)$ after the first selection. As the selection process proceeds, the number of arithmetic operations actually required in the re-triangularization operation will therefore be monotonically reduced.

At each step of the selection process, the error-reduction-ratio for the current selection is calculated using Equation (43). To measure the goodness-of-fit for the i-th sub-system after j regressors have been selected, the accumulated error-reduction-ratio is updated to yield

$$\sum_{h=1}^{j} ERR_i^{(h)}(k) = ERR_i^{(j)}(k) + \sum_{h=1}^{j-1} ERR_i^{(h)}(k) \text{ with } ERR_i^{(0)}(k) = 0 \tag{52}$$

The accumulated energy of the i-th sub-system output $y_i(k)$, $\|\mathbf{y}_i(k)\|^2$ required in Equation (43) can be computed recursively by

$$\|\mathbf{y}_i(k)\|^2 = y_i^2(k) + \lambda\|\mathbf{y}_i(k-1)\|^2 \text{ with } \|\mathbf{y}_i(0)\| = 0. \tag{53}$$

In practice, a measure called the *normalised* (to $\|\mathbf{y}_i(k)\|^2$) *residual sum-of-squares* (NRSS) defined as

$$NRSS_i^{(j)}(k) \triangleq 1 - \sum_{h=1}^{j} ERR_i^{(h)}(k), \tag{54}$$

which is directly related to the accumulated error-reduction-ratio given in Equation (52) suggested by [19], is usually used as a criterion for terminating the selection process if either one of the following conditions is satisfied

$$j = N_h \tag{55}$$

or

$$NRSS_i^{(j)}(k) \geq \xi. \tag{56}$$

where $\xi \in (0,1)$ is a pre-specified cut-off value for the regressor selection algorithm. These two conditions correspond to the situation where all the candidate regressors have been encountered in Equation (55) (even without achieving the required NRSS) and the required NRSS has been achieved after j regressor selection steps in Equation (56). In either case, set

$$n_h(k) = j \tag{57}$$

to denote the total number of successfully selected candidate terms and set $NRSS_i(k)$ as

$$NRSS_i(k) \underline{\underline{\Delta}} NRSS_i^{(n_h(k))}(k), \tag{58}$$

at time step k to inhibit the same selection procedure being repeated by considering the next largest $v_r^2(k)$ value with j incremented by one.

With the obtained triangular system given by $\overline{\mathbf{R}}_i^{(n_h(k))}(k)$ in transformation (50) and $\overline{\mathbf{p}}_i^{(n_h(k))}(k)$ in transformation (51), the sub-system, formed by the first $n_h(k) \times n_h(k)$ entries of the matrix $\overline{\mathbf{R}}_i^{(n_h(k))}(k)$, $\left[\overline{\mathbf{R}}_i^{(n_h(k))}(k)\right]_{1:n_h(k),1:n_h(k)}$, and the first $n_h(k)$ elements of the vector $\overline{\mathbf{v}}_i^{(n_h(k))}(k)$, $\left[\overline{\mathbf{v}}_i^{(n_h(k))}(k)\right]_{1:n_h(k)}$, can be used to determine a permuted version of the estimated weight vector $\boldsymbol{\theta}_i^*(k) \in \mathbf{R}^{n_h(k)}$ whose components are only associated with the selected candidate centres, $\overline{\boldsymbol{\theta}}_i^*(k) \in \mathbf{R}^{n_h(k)}$. In other words, $\boldsymbol{\theta}_i^*(k)$ is nothing more than a subset of the full (least squares) estimated weight vector $\boldsymbol{\theta}_i(k)$ defined in Equation (30) with the components corresponding to the unselected regressors omitted. The permuted weight vector $\overline{\boldsymbol{\theta}}_i^*(k)$ can be solved by performing backward substitutions in the following vector-matrix equation

$$\left[\overline{\mathbf{R}}_i^{(n_h(k))}(k)\right]_{1:n_h(k),1:n_h(k)} \overline{\boldsymbol{\theta}}_i^*(k) = \left[\overline{\mathbf{v}}_i^{(n_h(k))}(k)\right]_{1:n_h(k)}. \tag{59}$$

The proper least squares estimate of the weight vector $\boldsymbol{\theta}_i^*(k)$ can be obtained by re-permuting $\overline{\boldsymbol{\theta}}_i^*(k)$ in Equation (59) to yield

$$\boldsymbol{\theta}_i^*(k) = \left\{ \left[\boldsymbol{\Pi}_i^{\mathrm{T}}(k)\right]_{r,1:n_h(k)} \overline{\boldsymbol{\theta}}_i^*(k) : \left[\boldsymbol{\Pi}_i(k)\right]_{1:n_h(k),r} \neq \underbrace{[0 \quad 0 \quad \cdots \quad 0]^{\mathrm{T}}}_{n_h(k)} \right\}_{r=1}^{N_h} \tag{60}$$

where

$$\boldsymbol{\Pi}_i(k) \underline{\underline{\Delta}} \boldsymbol{\Pi}_i^{(1)}(k) \boldsymbol{\Pi}_i^{(2)}(k) \cdots \boldsymbol{\Pi}_i^{(n_h(k))}(k), \tag{61}$$

$[\Pi_i(k)]_{1:n_h(k),r}$ or its transpose $\left[\Pi_i^{\mathrm{T}}(k)\right]_{r,1:n_h(k)}$ is the vector (sub-matrix) formed by the first $n_h(k)$ components in the r-th column of the permutation matrix $\Pi_i(k)$.

Similarly, by defining $\varphi_i^*(k) \in \mathrm{R}^{n_h(k)}$ as a subset of the full input vector to the output layer $\varphi(k)$ with the unselected regressors missing as

$$\varphi_i^*(k) \underline{\Delta} \left\{ \varphi_r(k): [\Pi_i(k)]_{1:n_h(k),r} \neq \underbrace{[0 \quad 0 \quad \cdots \quad 0]^{\mathrm{T}}}_{n_h(k)} \right\}_{r=1}^{N_h} \tag{62}$$

the network output $\hat{y}_i(k)$ at time step k can be computed as

$$\hat{y}_i(k) = \varphi_i^{*\mathrm{T}}(k)\theta_i^*(k) \tag{63}$$

If any innovation terms are to be included in constructing a noise model, the innovation $\varepsilon_i(k)$ should also be computed by

$$\varepsilon_i(k) \equiv y_i(k) - \hat{y}_i(k) \tag{64}$$

which is stored for processing in the next time step.

Finally, before the data in the next time step $(k+1)$ is processed, the matrix $\overline{\mathrm{R}}_i^{(n_h(k))}(k)$ should be re-permuted as

$$\mathrm{R}_i(k) = \overline{\mathrm{R}}_i^{(n_h(k))}(k)\Pi_i(k) \tag{65}$$

in order to align with the corresponding components in the vector $\varphi(k)$ defined in Equation (16) for the next time step.

5.3. Three-Phase Learning

The network learning described above can be broadly categorised into the three different stages of centre updating, centre selection and weight estimation. These sub-tasks, which form the major computational steps involved in the learning process should generally provide a satisfactory performance in most applications where reliable information on the range of network inputs can be determined so that reasonable initial settings can be specified. In circumstances where such *a priori* information is not available the algorithm can be further refined to reduce uncertainty in initialising the range of the network inputs. Furthermore, when an estimate of the noise model is required, an initial phase is necessary in order to establish a set of appropriate ranges for initialising the innovation components involved in the noise model.

In the first phase, the algorithm starts with (uniformly) randomly initialised RBF centres in arbitrary range in the $(mN_u + nN_y)$-dimensional space assuming that no information on the range of network inputs is available. In this initial learning phase, the network input vector $\mathbf{x}(k)$ is taken to be the composition of the lagged system inputs $\mathbf{u}(k-1), \mathbf{u}(k-2), \cdots, \mathbf{u}(k-N_u)$ and the lagged system

outputs $\mathbf{y}(k-1),\mathbf{y}(k-2),\cdots,\mathbf{y}(k-N_y)$ without any innovation terms involved. After a short transition period has elapsed, all the components in the network input vector, and any innovation terms for the noise model where appropriate, are monitored at each time step so that their ranges can be recorded. These estimated ranges of the network inputs facilitate finer adjustments on the assumed ranges set at the beginning of the first phase. Subsequent to the random initialisation of the RBF centres, all the centres are updated by *k*-means clustering followed by centre selection together with weight updating using the Givens procedure described above.

With the range information obtained from the first phase, the learning process in the second phase is based on the definition of the network input vector given in Equation (4). The algorithm is re-started with the initialised centres reset to the estimated ranges obtained in the previous step. As in the first phase, the centre updating and weight estimation are achieved by unsupervised and supervised learning using the *k*-means clustering and Givens procedure. After a short transition period a subset of all the network input vector sequences are stored. A typical arrangement is to store all the network input vectors in a reasonably long period after the transient. These stored network input vectors can be reliably considered as representative of data clusters in the network input space which then form the essential element for the third phase of learning.

The distinctive feature in the third phase of learning is that all the RBF centres are treated as constants. In this respect, the task of network learning is reduced to centre selection and weight updating. The learning process is therefore re-started and all parts of the algorithm are re-initialised accordingly. During the entire course of the third phase all the stored network input vectors in phase two are used as the fixed centres. The total number of these centres is typically taken between 100 and 200. Based on these fixed centres, a significant subset is selected and the associated weights are estimated using the Givens procedure. With this form of network structure, the only network parameters to be adjusted are the network weights.

6. Summary of The Network Learning Algorithm

The network learning strategy can be divided into three phases in order to gain maximum benefit in utilising the available information. For the sake clarity, the complete learning process can be summarised by the following computational steps.

☞ Phase 1 $(k=1,2,\cdots,N_1)$:

Phase 1 initialisation:

- Specify the maximum lags N_u, N_y and N_ε in the system input, output and innovations;
- Specify the number of RBF centres to be used in phase 1 and phase 2, N_c;

- Specify the cut-off ξ value for the selection routine;
- Set the initial network input vector $\mathbf{x}(0)$ to be a $N_i\left(=mN_u+nN_y\right)$-component zero vector;
- Set the initial clustering gain $\kappa(0)$ (typical values are from 0.5 to 0.9);
- Specify the forgetting factor λ (typical values are from 0.9 to 0.99).
- Randomly initialise all components of the N_c RBF centres $\left\{\mathbf{c}_j(0): j=1,2,\cdots,N_c\right\}$ in arbitrary ranges;
- Set the n matrices $\left\{\mathbf{R}_i(0): i=1,2,\cdots,n\right\}$ to be diagonal matrices with small positive numbers δ (typically $\delta \approx 10^{-3}$);
- Set the n vectors $\left\{\mathbf{v}_i(0): i=1,2,\cdots,n\right\}$ to a zero vector of $N_h\left(=N_c+N_i+1\right)$ components;
- Construct the matrix $\left[\begin{array}{c:c} \mathbf{R}_i(0) & \mathbf{v}_i(0) \\ \hdashline 0 \quad \cdots \quad 0 & * \end{array}\right]$;

Network learning procedure:

for $k=1$ **to** N_1

Network input node assignments and centre updating:

- Assign the network input vector as
  $\mathbf{x}(k)=\left[\mathbf{u}(k-1) \quad \cdots \quad \mathbf{u}(k-N_u) \quad \mathbf{y}(k-1) \quad \cdots \quad \mathbf{y}(k-N_y)\right]^{\mathrm{T}}$;
- Compute the clustering gain $\kappa(k)$ using Equation (29);
- Update all the N_c centres using Equation (27);
- Compute the input to output layer vector $\varphi(k)=\left\{\varphi_j(k)\right\}_{j=1}^{N_h}$ where $N_h=N_c+N_i+1$ as $\varphi_j(k)=\begin{cases}\phi\left(\left\|\mathbf{x}(k)-\mathbf{c}_j(k)\right\|\right) & \text{for } j=1,2,\cdots,N_c \\ x_{j-N_c}(k) & \text{for } j=N_c+1,N_c+2,\cdots,N_h-1; \\ 1 & \text{for } j=N_h\end{cases}$

Centre selection and weight updating:

for $i=1$ **to** n **do**

- Set up the matrix in the left hand side of expression (40);
- Perform the Givens transformation defined by expression (41);
- Compute the vector $\mathbf{p}_i^{(0)}(k)$ as $\mathbf{p}_i^{(0)}(k)=\left[v_{i1}^2(k) \quad v_{i2}^2(k) \quad \cdots \quad v_{iN_h}^2(k)\right]^{\mathrm{T}}$;
- Initialise the selection counter $j=0$, the set the initial NRSS, $NRSS_i^{(0)}(k)=1$;

$j=0$;

while $j \le N_h$ **and** $NRSS_i(k)>\xi$ **do**

$j=j+1$;

- Find the j-th largest $v_r^2(k)$ $(1 \le r \le N_h)$ such that $l = \arg \max_{h=j,j+1,\cdots,N_h} \left\{ p_{ih}^{(j-1)}(k) \right\}$;
- Find the permutation matrix $\Pi_i^{(j)}(k)$ defined in Equation (46);
- Permute the matrix $\mathbf{R}_i^{(j-1)}(k)$ using Equation (48);
- Permute the vector $\mathbf{p}_i^{(j-1)}(k)$ using Equation (49);
- Re-triangularize $\mathbf{R}_i^{(j)}(k)$ to $\overline{\mathbf{R}}_i^{(j)}(k)$ by Givens rotations;
- Rotate $\mathbf{v}_i^{(j)}(k)$ to $\overline{\mathbf{v}}_i^{(j)}(k)$ to align with the re-triangularization of $\mathbf{R}_i^{(j)}(k)$;
- Compute ERR using Equation (52);
- Compute accumulated energy in system output $\|\mathbf{y}_i(k)\|^2$ using Equation (53);
- Compute NRSS using Equation (54);
- Set $n_h(k) = j$;
- Set $NRSS_i(k) = NRSS_i^{(n_h(k))}(k)$

end

- Solve Equation (59) by backward substitution to obtain $\overline{\theta}_i^*(k)$;
- Re-permute $\overline{\theta}_i^*(k)$ to obtain $\theta_i^*(k)$ using Equation (60);
- Obtain $\varphi_i^*(k)$ using Equation (62);
- Compute network output $\hat{y}_i(k)$ using Equation (63);
- Compute the innovation $\varepsilon_i(k)$ using Equation (64);
- Update the range of the system input, output and innovation if $k > K$ where K is the length of the transient sequence (typically $K \approx 100$);
- Re-permute $\overline{\mathbf{R}}_i^{(n_h(k))}(k)$ to obtain $\mathbf{R}_i(k)$ using Equation (65) for the next iteration;

end

end

☛ Phase 2 $(k = N_1 + 1, N_1 + 2, \cdots, N_1 + N_2)$:

Phase 2 initialisation:

- Set the initial network input vector $\mathbf{x}(N_1)$ to be a $N_i(= mN_u + nN_y + nN_\varepsilon)$-component zero vector;
- Set the initial clustering gain $\kappa(N_1)$ (typical values are from 0.5 to 0.9);
- Randomly re-initialise all components of the N_c RBF centres $\{\mathbf{c}_j(N_1): j = 1,2,\cdots,N_c\}$ in the estimated ranges;
- Set the n matrices $\{\mathbf{R}_i(N_1): i = 1,2,\cdots,n\}$ to be diagonal matrices with small positive numbers δ (typically $\delta \approx 10^{-3}$);
- Set the n vectors $\{\mathbf{v}_i(N_1): i = 1,2,\cdots,n\}$ to zero vectors of $N_h(= N_c + N_i + 1)$ components.

Network learning procedure:

for $k = N_1 + 1$ **to** $N_1 + N_2$

Network input node assignments and centre updating:

- Assign the network input vector $\mathbf{x}(k)$ according to Equation (4);
- Store $\mathbf{x}(k)$ after a transient period, say K samples (typically $K \approx 100$);
- Compute the clustering gain $\kappa(k)$ according to $\kappa(k) = \kappa(k-1) \Big/ \sqrt{1 + \text{int}\left[(k - N_1)/N_c\right]}$;
- Update all the N_c centres using Equation (27);
- Compute the input to output layer vector $\varphi(k) = \left\{\varphi_j(k)\right\}_{j=1}^{N_h}$ where $N_h = N_c + N_i + 1$ as $\varphi_j(k) = \begin{cases} \phi\left(\left\|\mathbf{x}(k) - \mathbf{c}_j(k)\right\|\right) & \text{for } j = 1, 2, \cdots, N_c \\ x_{j-N_c}(k) & \text{for } j = N_c + 1, N_c + 2, \cdots, N_h - 1; \\ 1 & \text{for } j = N_h \end{cases}$

Centre selection and weight updating:

- Same procedures as phase 1;

end

Phase 3 $(k = N_1 + N_2 + 1, N_1 + N_2 + 2, \cdots, N_1 + N_2 + N_3)$:

Phase 3 initialisations:

- Use the stored network input vectors in phase 2 as candidate centres in phase 3, i.e. $\left\{\mathbf{c}_j(k) = \mathbf{x}(j + N_1 + K) : j = 1, 2, \cdots, N_2 - K\right\}$;
- Enlarge the dimension of the n matrices $\left\{\mathbf{R}_i(N_1 + N_2) : i = 1, 2, \cdots, n\right\}$ from $(N_c + N_i + 1) \times (N_c + N_i + 1)$ to $(N_2 - K + N_i + 1) \times (N_2 - K + N_i + 1)$;
- Enlarge the dimension of the n vectors $\left\{\mathbf{v}_i(N_1 + N_2) : i = 1, 2, \cdots, n\right\}$ from $(N_c + N_i + 1)$ to $(N_2 - K + N_i + 1)$;
- Set the n matrices $\left\{\mathbf{R}_i(N_1 + N_2) : i = 1, 2, \cdots, n\right\}$ to be diagonal matrices with small positive numbers δ (typically $\delta \approx 10^{-3}$);
- Set the n vectors $\left\{\mathbf{v}_i(N_1 + N_2) : i = 1, 2, \cdots, n\right\}$ to a zero vector of $N_h (= N_c + N_i + 1)$ components;

Network learning procedure:

for $k = N_1 + N_2 + 1$ **to** $N_1 + N_2 + N_3$ **do**

- Compute hidden layer outputs using Equation (8);
- Assign the network input vector $\mathbf{x}(k)$ according to Equation (4);

- Compute the input vector to output layer $\varphi(k)=\{\varphi_j(k)\}_{j=1}^{N_h}$ where $N_h=N_c+N_i+1$ as $\varphi_j(k)=\begin{cases}\phi\left(\|\mathbf{x}(k)-\mathbf{c}_j(k)\|\right) & \text{for } j=1,2,\cdots,N_c\\ x_{j-N_c}(k) & \text{for } j=N_c+1,N_c+2,\cdots,N_h-1;\\ 1 & \text{for } j=N_h\end{cases}$

Centre selection and weight updating:

for $i=1$ **to** n **do**

- Perform Givens transformation as shown in expression (41);
- Compute the vector $\mathbf{p}_i^{(0)}(k)$ as $\mathbf{p}_i^{(0)}(k)=\left[v_{i1}^2(k)\quad v_{i2}^2(k)\quad\cdots\quad v_{iN_h}^2(k)\right]^{\mathrm{T}}$;
- Initialise the selection counter $j=0$, the initial NRSS, $NRSS_i^{(0)}(k)=1$;

$j=0$;

while $j\le N_h$ **and** $NRSS_i(k)>\xi$ **do**

$j=j+1$;

- Find j-th largest $v_r^2(k)$ $(1\le r\le N_h)$ such that $l=\arg\max_{h=j,j+1,\cdots,N_h}\left\{p_{ih}^{(j-1)}(k)\right\}$;
- Find the permutation matrix $\Pi_i^{(j)}(k)$ defined in Equation (46);
- Permute the matrix $\mathbf{R}_i^{(j-1)}(k)$ using Equation (48);
- Permute the vector $\mathbf{p}_i^{(j-1)}(k)$ using Equation (49);
- Re-triangularize $\mathbf{R}_i^{(j)}(k)$ to $\overline{\mathbf{R}}_i^{(j)}(k)$ using Givens rotations;
- Rotate $\mathbf{v}_i^{(j)}(k)$ to $\overline{\mathbf{v}}_i^{(j)}(k)$ to align with the re-triangularization of $\mathbf{R}_i^{(j)}(k)$;
- Compute ERR using Equation (52);
- Compute the accumulated energy in the system output $\|\mathbf{y}_i(k)\|^2$ using Equation (53);
- Compute NRSS using Equation (54);
- Set $n_h(k)=j$;
- Set $NRSS_i(k)=NRSS_i^{(n_h(k))}(k)$

end

- Solve Equation (59) by backward substitution to obtain $\overline{\theta}_i^*(k)$;
- Re-permute $\overline{\theta}_i^*(k)$ to obtain $\theta_\mu^*(k)$ using Equation (60);
- Obtain $\varphi_i^*(k)$ using Equation (62);
- Compute the network output $\hat{y}_i(k)$ using Equation (63);
- Compute the innovation $\varepsilon_i(k)$ using Equation (64);
- Update the range of the system input, output and innovation;
- Re-permute $\overline{\mathbf{R}}_i^{(n_h(k))}(k)$ to obtain $\mathbf{R}_i(k)$ using Equation (65) for the next iteration;

end

end

7. On-Line Structural Monitoring

There are many structures where it would be desirable to have some mechanism which monitors on-line the performance of the learning strategy. Such a procedure should be independent of the learning algorithm but should be capable of detecting any effects which lead to an inadequate representation of the data [3][19]. In the current application for example if the structure of the underlying system changes the set of centres currently in use may no longer be representative and a re-clustering procedure may need to be initiated. On-line performance monitoring which can detect such effects will often be crucial and is particularly important for systems whose dynamics may change during the course of estimation.

7.1. Off-Line Formulations of Model Validity Tests

It has recently been suggested [5] that an estimated multi-input, multi-output model can be considered as an acceptable representation of the data set if the following correlation based validation tests are satisfied:-

$$\left| \frac{\sum_{k=1}^{N} \alpha(k-\tau)\gamma(k)}{\sqrt{\sum_{k=1}^{N} \alpha^2(k)} \cdot \sqrt{\sum_{k=1}^{N} \gamma^2(k)}} \right| < \frac{1.96}{\sqrt{N}} \tag{66}$$

and

$$\left| \frac{\sum_{k=1}^{N} \beta(k-\tau)\gamma(k)}{\sqrt{\sum_{k=1}^{N} \beta^2(k)} \cdot \sqrt{\sum_{k=1}^{N} \gamma^2(k)}} \right| < \frac{1.96}{\sqrt{N}} \tag{67}$$

where

$$\alpha(k) \triangleq \sum_{j=1}^{n} \frac{\varepsilon_j^2(k) - \frac{1}{N}\sum_{k=1}^{N} \varepsilon_j^2(k)}{\sqrt{\frac{1}{N}\sum_{k=1}^{N} \left[\varepsilon_j^2(k) - \frac{1}{N}\sum_{l=1}^{N} \varepsilon_j^2(l) \right]^2}}, \tag{68}$$

$$\beta(k) \triangleq \sum_{i=1}^{m} \frac{u_i^2(k) - \frac{1}{N}\sum_{k=1}^{N} u_i^2(k)}{\sqrt{\frac{1}{N}\sum_{k=1}^{N} \left[u_i^2(k) - \frac{1}{N}\sum_{l=1}^{N} u_i^2(l) \right]^2}}, \tag{69}$$

and

$$\gamma(k)\underline{\Delta}\sum_{j=1}^{n}\frac{z_j(k)\varepsilon_j(k)-\frac{1}{N}\sum_{k=1}^{N}z_j(k)\varepsilon_j(k)}{\sqrt{\frac{1}{N}\sum_{k=1}^{N}\left[z_j(k)\varepsilon_j(k)-\frac{1}{N}\sum_{l=1}^{N}z_j(l)\varepsilon_j(l)\right]^2}} \tag{70}$$

On-line versions of these tests can be used in the present application to monitor the performance of the network training.

7.2. Recursive Validation Tests for System Structure Monitoring

The above computations of Equations (66)-(70) require that all the data from $k=1$ to $k=N$ is available in batch form. However Equation (66) and (67) can be re-formulated to provide a recursive or on-line form which is suitable for the present application by defining

$$\rho_1(k)=\eta\rho_1(k-1)+(1-\eta)\left[\frac{1}{\tau_{max}+1}\sum_{\tau=0}^{\tau_{max}}\left|\frac{\alpha'(k-\tau)-\mu_{\alpha'}(k)}{\sigma_{\alpha'}(k)}\right|\right]\cdot\left|\frac{\gamma'(k)-\mu_{\gamma'}(k)}{\sigma_{\gamma'}(k)}\right| \text{ with } \rho_1(0)=0 \tag{71}$$

and

$$\rho_2(k)=\eta\rho_2(k-1)+(1-\eta)\left[\frac{1}{\tau_{max}+1}\sum_{\tau=0}^{\tau_{max}}\left|\frac{\beta'(k-\tau)-\mu_{\beta'}(k)}{\sigma_{\beta'}(k)}\right|\right]\cdot\left|\frac{\gamma'(k)-\mu_{\gamma'}(k)}{\sigma_{\gamma'}(k)}\right| \text{ with } \rho_2(0)=0 \tag{72}$$

where

$$\alpha'(k)\underline{\Delta}\sum_{j=1}^{n}\varepsilon_j^2(k) \tag{73}$$

$$\beta'(k)\underline{\Delta}\sum_{i=1}^{m}u_i^2(k) \tag{74}$$

$$\gamma'(k)\underline{\Delta}\sum_{j=1}^{n}z_j(k)\varepsilon_j(k) \tag{75}$$

$$\mu_{\alpha'}(k)=\eta\mu_{\alpha'}(k-1)+(1-\eta)\alpha'(k) \text{ with } \mu_{\alpha'}(0)=0 \tag{76}$$

$$\mu_{\beta'}(k)=\eta\mu_{\beta'}(k-1)+(1-\eta)\beta'(k) \text{ with } \mu_{\beta'}(0)=0 \tag{77}$$

$$\mu_{\gamma'}(k)=\eta\mu_{\gamma'}(k-1)+(1-\eta)\gamma'(k) \text{ with } \mu_{\gamma'}(0)=0 \tag{78}$$

$$\sigma_{\alpha'}(k)=\sqrt{\eta\sigma_{\alpha'}^2(k-1)+(1-\eta)\left[\alpha'(k)-\mu_{\alpha'}(k)\right]^2} \text{ with } \sigma_{\alpha'}(0)=\delta' \tag{79}$$

$$\sigma_{\beta'}(k)=\sqrt{\eta\sigma_{\beta'}^2(k-1)+(1-\eta)\left[\beta'(k)-\mu_{\beta'}(k)\right]^2} \text{ with } \sigma_{\beta'}(0)=\delta' \tag{80}$$

$$\sigma_{\gamma'}(k)=\sqrt{\eta\sigma_{\gamma'}^2(k-1)+(1-\eta)\left[\gamma'(k)-\mu_{\gamma'}(k)\right]^2} \text{ with } \sigma_{\gamma'}(0)=\delta' \tag{81}$$

where δ' is an arbitrarily small positive number and η is a scalar constant forgetting factor which controls the level of compromise between the effect of slow detection when η is large and the possibility of giving a false alarm when η is small. Such technique has also been used successfully in the application of on-line performance monitoring for the adaptive noise cancellation problem [4]. Any

abrupt changes observed in sequence(s) $\{\rho_1(k)\}$ and/or $\{\rho_2(k)\}$ can be used as an indication that a structural change may have occurred in the system or some other aspect of training is incorrect. Typical values for τ_{max}, η and δ' can be taken as $\tau_{max}=10$, $\eta=0.995$ and $\delta'=10^{-4}$, respectively. It is worth noting that, unlike the case of off-line validity tests where the level of significance can easily be specified using 95% confidence bands such threshold values cannot be used with the on-line tests because sudden changes in signal levels may incorrectly trigger a false alarm before any transitory effects vanish. Nevertheless, the on-line formulation does provide a simple and convenient means of monitoring [3] and acts as a warning device to detect potential problems that may arise during the estimation process while keeping the computational cost to a minimum.

8. Applications to Pattern Recognition

One of the potential applications of neural networks is pattern recognition where input patterns are to be divided into a finite number of classes. Such problems frequently arise in many disciplines such as biomedical data analysis, seismology, image pattern recognition, digital communication systems, electric power station fault detection and many others. The common scenario is that given a set of data obtained from sampled measurements, a mapping rule is required which will allocate a given data pattern to one of the several categories with minimal probability of mis-classification.

In the context of pattern recognition, the role of a neural network can be viewed as a rule extraction device which generalises a set of unknown pattern-to-category mapping rules through network learning. Since the problem is usually defined with a finite number of categories this implies the mapping rules are likely to be non-linear. A benchmark problem in pattern recognition is the exclusive OR example where linear inseparability [16] is encountered so that good classification rules can only be generated by systems that exhibit non-linearity.

Consider the exclusive OR example defined by the following truth table illustrated in Table 1. From this table it can be seen that the two categories mapped from the four possible input patterns lie in two non-connected regions in two-dimensional Euclidean space. It is therefore not possible to find a hyperplane which completely separates the regions in the two categories as shown in Figure 3. An optimal decision requires a non-linear boundary which can easily be generated by a well trained RBF network.

input 1	input 2	category
-1	-1	2
-1	1	1
1	-1	1
1	1	2

Table 1 XOR truth table

In an experiment to illustrate the new on-line training algorithm described above a sequence of 1000 pairs of input pattern-to-category training data were used based upon noisy inputs which were independently corrupted by additive white Gaussian noise $\mathcal{N}(0,10^{-2})$. During the training process, 30 RBF centres were initially randomly assigned in the region of $[-1,1]\times[-1,1]$. With the initial clustering gain $\kappa(0)=0.9$, forgetting factor $\lambda=0.97$, $\mathbf{R}(0)=10^{-3}\mathbf{I}$ and forward regressor selection cut-off $\xi=0.01$, 200 iterations were run in phase 1 training prior to re-starting the algorithm in phase 2 training with all 30 centres re-initialised randomly in the region $[-1.276,1.204]\times[-1.276,1.204]$. In phase 2 training, 200 iterations were used with the last 100 network input vectors stored for centre selections in phase 3 training. Thus, at any time step in phase 3 training, 103 candidates terms, which come from the 100 network input vectors in phase 2 together with the linear and dc links, were available. After running 600 iterations in phase 3 training with the first 100 iterations discarded to avoid start-up and transient effects a signal-to-noise ratio of 14.77 dB was achieved with a network complexity of just 6 selected terms on average. By using 7 selected terms from the final iteration decision region generated by the trained RBF network can be constructed as shown in Figure 4. This decision region, which is very similar to the ideal region depicted in Figure 3, demonstrates that the trained RBF network provides a good approximation to the classification rules given in truth Table 1 for the exclusive OR problem. The final network parameters are given in Table 2.

final centres		
first component	second component	final weights
-1.167	0.960	1.167
-0.983	0.959	2.429
1.161	-0.974	3.139
-1.090	1.068	-1.537
-1.005	0.831	-1.702
0.931	0.969	-4.168
-1.110	-1.002	-3.169

Table 2 Final network parameters after training for the XOR example

9. Application to Single-Input-Single-Output Time Domain Dynamic Data Modelling

Application of the new algorithm to non-linear system modelling will be illustrated by identifying a RBF model of a non-linear liquid level system.

The liquid level system consists of a water pump, which is driven by a dc voltage taken as the system input variable, feeding water into a conical flask which in turn feeds a square tank. By taking the water level in the conical flask as the system output variable a system with a non-linear input-output relationship is established. A detailed description of this process is given in [1].

In the first phase of the identification process, the following network input node assignment was used

$$\mathbf{x}(k)=\left[u(k-1) \quad \cdots \quad u(k-5) \quad y(k-1) \quad \cdots \quad y(k-3)\right]^{\mathrm{T}}. \tag{82}$$

With initial settings of the clustering gain $\kappa(0)=0.9$, forgetting factor $\lambda=0.97$, $\mathbf{R}(0)=10^{-3}\mathbf{I}$, 200 time steps were used to estimate the domain of the network input space. The algorithm started with 100 RBF centres which were all randomly initialised in the range of $[-1,1]$ for all the components. After 200 time steps innovation terms were augmented to the network input vector $\mathbf{x}(k)$ so that

$$\mathbf{x}(k)=\left[u(k-1) \quad \cdots \quad u(k-5) \quad y(k-1) \quad \cdots \quad y(k-3) \quad \varepsilon(k-1) \quad \varepsilon(k-2)\right]^{\mathrm{T}}. \tag{83}$$

Just prior to the starting time step 201 all 30 RBF centres were re-initialised so that all system input components were distributed as $[-0.742,0.898]$, all system output components were in $[-2.168,0.894]$ and all innovation components were in $[-0.132,0.195]$. A total of 300 time steps were used in phase 2 of training with the last 200 captured for centre selection in the subsequent phase. Starting from time step 501, after a re-initialisation with $\mathbf{R}(500)=10^{-3}\mathbf{I}$ and $\lambda=0.97$, phase 3 training took another 500 time steps to achieve a signal-to-noise ratio of 25.11 dB with a network complexity of just 14 selected terms out of a total number of 211 candidate terms on average using the selection cut-off $\xi=0.01$. The identification process pictured in Figure 5 shows that the network predicted output is highly consistent with the system output and confirms the performance of the new algorithm. Additional information on system structure is augmented by the on-line correlation sequences $\{\rho_1(k)\}$ and $\{\rho_2(k)\}$. These correlation sequences show that the underlying network was performing well in capturing the system dynamics except on a few occasions where the system experienced substantial variations in signal level which required significant re-adjustments in network structure in order to match the changing dynamics.

10. Application to Multi-Input-Multi-Output Time Domain Dynamic Data Modelling

To further explore the versatility of the modelling capability of the new algorithm, the identification of a 2-input, 2-output non-linear system will be studied.

Consider the 2-input, 2-output simulated system governed by the following system equations.

$$\left.\begin{aligned} y_1(k) &= 0.5y_1(k-1) + u_1(k-2) + 0.1y_2(k-1)u_1(k-1) + 0.5e_1(k-1) + 0.2y_1(k-2)e_1(k-2) + e_1(k) \\ y_2(k) &= 0.9y_2(k-2) + u_2(k-1) + 0.2y_2(k-1)u_2(k-2) + 0.5e_2(k-1) + 0.1y_2(k-1)e_2(k-2) + e_2(k) \end{aligned}\right\} \quad (84)$$

where $\{u_1(k)\}$ is a Gaussian sequence with zero mean and unit variance, $\{u_2(k)\}$ is uniformly distributed in the range $\left[-\sqrt{3},\sqrt{3}\right]$ and $\{e_1(k)\}$, $\{e_2(k)\}$ are independently identically distributed Gaussian sequences with zero means and variance 0.04. Notice that there are non-linear noise terms.

The identification process was executed in three training phases. In the first two phases, 200 and 300 time steps were iterated respectively based on a 2-input, 2-output network with 100 RBF centres. In the first phase, the network input vector was assigned to be

$$\mathbf{x}(k) = \left[u_1(k-1) \quad u_1(k-2) \quad u_2(k-1) \quad u_2(k-2) \quad y_1(k-1) \quad y_1(k-2) \quad y_2(k-1) \quad y_2(k-2)\right]^{\mathrm{T}}. \quad (85)$$

Prior to starting the algorithm all components of the 100 centres were randomly initialised in the range of $[-1,1]$. Using an initial clustering gain $\kappa(0) = 0.9$, forgetting factor $\lambda = 0.95$ and centre selection cut-off $\xi = 0.01$ an estimated range for the system inputs of $[-2.761, 2.400]$, system outputs of $[-1.673, 1.730]$ and innovation terms of $[-5.130, 2.607]$ were obtained after 200 iterations of phase 1 training. These estimated ranges were then used to re-initialise all the 100 RBF centres in phase 2 with a re-defined network input vector $\mathbf{x}(k)$ as

$$\begin{aligned} \mathbf{x}(k) = [&u_1(k-1), u_1(k-2), u_2(k-1), u_2(k-2), y_1(k-1), y_1(k-2), \\ &y_2(k-1), y_2(k-1), \varepsilon_1(k-1), \varepsilon_1(k-2), \varepsilon_2(k-1)]^{\mathrm{T}} \end{aligned} \quad .(86)$$

which includes innovation terms.

Performing 300 iterations in phase 2 and ignoring the first 100 iterations gave 200 stored network input vectors from iteration 401 to iteration 600 which were used as candidate centres in phase 3 training. As a result of the phase 3 training as depicted in Figure 6, from iteration 601 to iteration 1000, produced signal-to-noise ratio of 24.52 dB and 17.53 dB for sub-system output 1 and sub-system output 2 with a network complexity of 30 and 17 centres on average, respectively. Again, these results demonstrate the effectiveness of the learning algorithm in the selection of RBF centres. As in the previous example, this argument is well supported by the traces of on-line correlation sequences $\{\rho_1(k)\}$ and $\{\rho_2(k)\}$. The occasional discrepancies at approximately time step 200 and time step 750 correctly indicate that substantial adjustments in network structure were required in response to the noticeable variations in signal levels at the system output.

11. Conclusions

A new on-line learning algorithm has been developed based on the RBF network architecture which encompasses centre selection and weight parameter estimation in a unified framework. By employing a clustering method to obtain an initial representative set of candidate RBF centres the centre selection procedure prior to weight updating provides a systematic mechanism for constructing parsimonious radial basis function networks. The centre selection procedure offers advantages over the conventional static structure approach by reducing the amount of computation involved in the subsequent weight estimation, by removing insignificant RBF centres and associated links and by offering the potential for adaptive tracking of time varying effects. A new recursive structural monitoring scheme based on a re-formulation of correlation based model validity tests for batch mode identification has also been introduced to complement the network learning process and to detect changes in system structure. Examples have been included to demonstrate the effectiveness of the new algorithms in reducing network complexity for applications in both pattern recognition type problems and temporal system modelling.

Acknowledgements

One of the authors (SAB) gratefully acknowledges that part of this work was supported by EPSRC.

References

[1] Billings, S. A. and W. S. F. Voon (1986). A prediction-error and stepwise-regression estimation algorithm for non-linear systems. *International Journal of Control*, **44**(3), 803-822.

[2] Billings, S. A. and S. Chen (1989). Extended model set, global data and threshold model identification of severely non-linear systems. *International Journal of Control*, **50**(5), 1897-1923.

[3] Billings, S. A. and F. A. Alturki (1992). Performance monitoring in non-linear adaptive noise cancellation. *Journal of Sound and Vibration*, **157**(1), 161-175.

[4] Billings, S. A. and C. F. Fung (1993). Recurrent radial basis function networks for adaptive noise cancellation. *Neural Networks* (in press).

[5] Billings, S. A. and Q. M. Zhu (1994). Model validation tests for multivariable non-linear models including neural networks (submitted for publication).

[6] Broomhead, D. S. and D. Love (1988). Multivariable functional interpolation and adaptive networks, *Complex Systems*, **2**, 321-355.

[7] Chen, S. and S. A. Billings (1988). Representation of non-linear systems: the NARMAX model. *International Journal of Control*, **49**(3), 303-344.

[8] Chen, S., S. A. Billings, C. F. N. Cowan and P. M. Grant (1990). Non-linear system identification using radial basis functions. *International Journal of Systems Science*, **21**(12), 2513-2539.

[9] Chen, S., S. A. Billings and P. M Grant (1992). Recursive hybrid algorithm for non-linear system identification using radial basis function networks. *International Journal of Control*, **55**(5), 1051-1070.

[10] Chen, S., Billings, S. A. and W. Luo (1989). Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, **50**(5), 1873-1896.

[11] Duda, R. O. and P. E. Hart (1973). *Pattern classification and scene analysis*. John Wiley and Sons, New York.

[12] Gentleman, W. M. (1973). Least squares computation by Givens transformation without square roots. *Journal of Institute of Mathematics and Its Applications*, **12**, 329-336.

[13] Gentleman, W. M and H. T. Kung (1981). Matrix triangularization by systolic arrays. *Proceedings of SPIE, (Real-Time Signal Processing IV)*, 298-303.

[14] Golub, G. H. and C. V. Loan (1989). *Matrix computations*, 2nd edition. Johns Hopkins University Press, Baltimore.

[15] Haykin, S. S. (1994). *Neural networks, a comprehensive foundation*. Macmillan College Publishing Company.

[16] Hertz, J., A. Krogh, and R. G. Palmer (1991). *Introduction to the theory of neural computation*. Addison-Wesley Publishing Company.

[17] Ling, F. (1991). Givens rotation based least squares lattice and related algorithms. *IEEE Transactions on Signal Processing*, **39**(7), 1541-1551.

[18] Ling, F., D. Manolakis and J. G. Proakis (1986). A recursive modified Gram-Schmidt algorithm for least-squares estimation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **34**(4), 829-836.

[19] Luo, W. and S. A. Billings and K. M. Tsang. (1994). On-line structure detection and parameter estimation with exponential windowing for nonlinear systems (submitted for publication).

[20] MacQueen, J. B., (1967). *Some methods for classification and analysis of multivariate observations*. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, **1**, Berkeley, Calififormia, University of California Press, 281-297.

[21] Mathews, V. J. (1991). Adaptive polynomial filters. *IEEE Signal Processing Magazine*, **8**(3), 10-26.

[22] Moody, J. E. and C. J. Darken (1989). Fast learning in networks of locally tunned processing units, *Nueral Computation* **1**, 281-294.

[23] Pao, Y. H. (1989). *Adaptive pattern recognition and neural networks*. Addison-Wesley Publishing Company, Inc.

[24] Poggio, T and F. Girosi (1990). Network for approximation and learning. *Proceedings IEEE*, 78(9), 1481-1496.

[25] Powell, M. J. D. (1985). Radial basis functions for multivariable interpolation: A review. In Mason, J. C. and Cox, M. G. (Eds.), *Algorithm for Approximation*, Oxford University Press, Oxford, UK, 143-167.

[26] Powell, M. J. D. (1988). Radial basis function approximations to polynomials. *Proceedings 12th Biennial Numerical Analysis Conference*, Dundee, UK, 223-241.

Appendix List of Key Symbols

k Time step index.

$\mathbf{f}(\cdot)$ Multi-input, multi-output non-linear system representation.

$\hat{\mathbf{f}}(\cdot)$ Function approximation of $\mathbf{f}(\cdot)$.

m Dimension of system input.

n Dimension of system output.

$\mathbf{u}(k)$ System input vector at time step k.

$u_i(k)$ The i-th component of $\mathbf{u}(k)$.

$\mathbf{y}(k)$ System output vector at time step k.

$y_i(k)$ The i-th component of $\mathbf{y}(k)$.

$\hat{\mathbf{y}}(k)$ Predicted system output vector at time step k.

$\hat{\mathbf{y}}_i(k)$ The predicted output vector over data record of length k for the i-th sub-system.

$\hat{\mathbf{Y}}(k)$ Predicted system output matrix over data record of length k.

$\mathbf{e}(k)$ Noise vector at time step k.

$e_i(k)$ The i-th component of $\mathbf{e}(k)$.

$\boldsymbol{\varepsilon}(k)$ Innovation vector at time step k.

$\varepsilon_i(k)$ The i-th component of $\boldsymbol{\varepsilon}(k)$.

N_u Maximum lag in $\mathbf{u}(k)$ as regressed inputs.

N_y Maximum lag in $\mathbf{y}(k)$ as regressed inputs.

N_e Maximum lag in $\mathbf{e}(k)$ as regressed inputs (in NARMAX representation).

N_ε Maximum lag in $\boldsymbol{\varepsilon}(k)$ as regressed inputs.

$\tilde{\mathbf{x}}(k)$ Vector of explanatory variables in non-linear system (NARMAX) representation.

$\mathbf{x}(k)$ Network input vector at time step k.

$x_i(k)$ The i-th component of $\mathbf{x}(k)$.

$\phi(\cdot)$ RBF non-linearity.

$\mathbf{c}_j(k)$ The j-th centre (vector) value at time step k.

$\omega_j(k)$ Width parameter associated with the j-th centre at time step k.

$d_i(k)$ Euclidean distance between the network input vector and the i-th centre at time step k.

$\kappa(k)$ Clustering gain at time step k.

$\boldsymbol{\varphi}(k)$ Input vector to output layer at time step k.

$\varphi_i(k)$ The i-th component of $\boldsymbol{\varphi}(k)$.

$\boldsymbol{\varphi}_i^*(k)$ Vector of selected regressors for the i-th network output at time step k.

$\boldsymbol{\Phi}(k)$ Data matrix over a record length of k.

$\mathbf{g}_i(k)$ The parameter vector in the auxiliary system.

$g_{ji}(k)$ The j-th component of $\mathbf{g}_i(k)$.

$\hat{\mathbf{g}}_i(k)$ The estimate of $\mathbf{g}_i(k)$.

$\hat{g}_{ji}(k)$ The j-th component of $\hat{\mathbf{g}}_i(k)$.

$\mathbf{W}(k)$ The data matrix in the auxiliary system.

$\mathbf{w}_i(k)$ The i-th column of $\mathbf{W}(k)$.

N_i Dimension of $\mathbf{x}(k)$.

N_c Number of RBF centres.

N_h Number of components in $\varphi(k)$.

$n_h(k)$ Number of selected components in $\varphi(k)$ at time step k.

N_1 Length of training sequence in phase 1 training.

N_2 Length of training sequence in phase 2 training.

N_3 Length of training sequence in phase 3 training.

K Length of transient sequence in each of the three phases during training.

$ERR_i^{(j)}(k)$ Error-reduction-ratio for the i-th sub-system contributed by the j-th selected centre at time step k.

$NRSS_i^{(j)}(k)$ Normalised residual sum-of-squares for the i-th sub-system after the j-th centre selection at time step k.

$NRSS_i(k)$ Normalised residual sum-of-squares for the i-th sub-system on termination of the selection at time step k.

ξ Pre-specified cut-off value for the centre (regressor) selection algorithm.

λ Forgetting factor used in RLS algorithm.

$\Lambda(k)$ $\text{diag}\left(\lambda^{k-1} \quad \lambda^{k-2} \quad \cdots \quad 1\right)$

$\mathbf{Q}(k)$ Skinny factor of $\Lambda^{1/2}(k)\Phi(k)$.

$\mathbf{S}(k)$ Skinny factor of $\Lambda^{1/2}(k)\Phi(k)$.

$\mathbf{R}_i(k)$ $\left[\mathbf{Q}^{\mathrm{T}}(k)\mathbf{Q}(k)\right]^{1/2}\mathbf{S}(k)$ (an upper triangular matrix)

$\mathbf{R}_i^{(j)}(k)$ Transformed version of $\mathbf{R}_i(k)$ after j selection steps (non-upper triangular).

$\overline{\mathbf{R}}_i^{(j)}(k)$ Upper triangularized version of $\mathbf{R}_i^{(j)}(k)$.

δ Small positive number in initialising $\mathbf{R}_i(k)$.

$\mathbf{v}_i(k)$ $\left[\mathbf{Q}^{\mathrm{T}}(k)\mathbf{Q}(k)\right]^{-1/2}\mathbf{Q}^{\mathrm{T}}(k)\mathbf{y}_i(k)$.

$v_{ir}(k)$ The r-th component of $\mathbf{v}_i(k)$.

$\overline{\mathbf{v}}_i^{(j)}(k)$ Rotated version of $\mathbf{v}_i(k)$ after j selection steps.

$\mathbf{p}_i^{(j)}(k)$ Vector containing all the $v_r^2(k)$ values with the first j entries being the largest in descending order.

$\Pi_i^{(j)}(k)$ Permutation matrix used in the j-th selection for the i-th network output at time step k.

$\Pi_i(k)$ Permutation matrix formed upon completion of the selection process for the i-th network output at time step k.

$\theta_j(k)$ The weight vector associated with the j-th output in a RBF network.

$\theta_{ij}(k)$ The i-th component in $\theta_j(k)$.

$\overline{\theta}_j^*(k)$ Estimated eight vector whose components corresponding to the selected centres for the j-th network output at time step k.

$\theta_j^*(k)$ Properly aligned version of $\overline{\theta}_j^*(k)$.

$\Theta(k)$ Connection weight matrix at time step k.

$$\alpha(k) = \sum_{j=1}^{n} \frac{\varepsilon_j^2(k) - \frac{1}{N}\sum_{k=1}^{N}\varepsilon_j^2(k)}{\sqrt{\frac{1}{N}\sum_{k=1}^{N}\left[\varepsilon_j^2(k) - \frac{1}{N}\sum_{l=1}^{N}\varepsilon_j^2(l)\right]^2}}$$

$$\beta(k) = \sum_{i=1}^{m} \frac{u_i^2(k) - \frac{1}{N}\sum_{k=1}^{N}u_i^2(k)}{\sqrt{\frac{1}{N}\sum_{k=1}^{N}\left[u_i^2(k) - \frac{1}{N}\sum_{l=1}^{N}u_i^2(l)\right]^2}}$$

$$\gamma(k) = \sum_{j=1}^{n} \frac{z_j(k)\varepsilon_j(k) - \frac{1}{N}\sum_{k=1}^{N}z_j(k)\varepsilon_j(k)}{\sqrt{\frac{1}{N}\sum_{k=1}^{N}\left[z_j(k)\varepsilon_j(k) - \frac{1}{N}\sum_{l=1}^{N}z_j(l)\varepsilon_j(l)\right]^2}}$$

$\rho_1(k)$ On-line model validity test 1.

$\rho_2(k)$ On-line model validity test 2.

$$\alpha'(k) = \sum_{j=1}^{n}\varepsilon_j^2(k)$$

$$\beta'(k) = \sum_{i=1}^{m}u_i^2(k)$$

$$\gamma'(k) = \sum_{j=1}^{n}z_j(k)\varepsilon_j(k)$$

$\mu_{\alpha'}(k)$ Moving average of $\alpha'(k)$ at time step k.

$\mu_{\beta'}(k)$ Moving average of $\beta'(k)$ at time step k.

$\mu_{\gamma'}(k)$ Moving average of $\gamma'(k)$ at time step k.

$\sigma_{\alpha'}(k)$ Moving standard deviation of $\alpha'(k)$ at time step k.

$\sigma_{\beta'}(k)$ Moving standard deviation of $\beta'(k)$ at time step k.

$\sigma_{\gamma'}(k)$ Moving standard deviation of $\gamma'(k)$ at time step k.

τ_{max} Maximum lag used in on-line model validity tests.

η Forgetting factor used in on-line model validity tests.

δ Small positive number.

δ' Small positive number.

L Scalar constant.

$\mathbf{I}$ Identity matrix.

$\mathbf{1}_i$ Vector whose i-th component is one and zero elsewhere.

$\mathbf{R}$ Set of real numbers.

$\mathbf{R}_+$ Set of positive real numbers.

$\mathbf{Z}$ Set of integers.

$\|\cdot\|$ 2-norm.

$[\cdot]_{i:j}$ (i,j)-th element of the matrix in the brackets.

$[\cdot]_{i:j_1:j_2}$ Row vector formed by the elements from column j_1 to column j_2 $(1 \le j_1 \le j_2)$ in the i-th row of the matrix in the brackets.

$[\cdot]_{i_1:i_2:j}$ Column vector formed by the elements from row i_1 to row i_2 $(1 \le i_1 \le i_2)$ in the j-th column of the matrix in the brackets.

$[\cdot]_{i_1:i_2:j_1:j_2}$ Sub-matrix formed by the elements from row i_1 to row i_2 $(1 \le i_1 \le i_2)$ and from column j_1 to column j_2 $(1 \le j_1 \le j_2)$ of the matrix in the brackets.

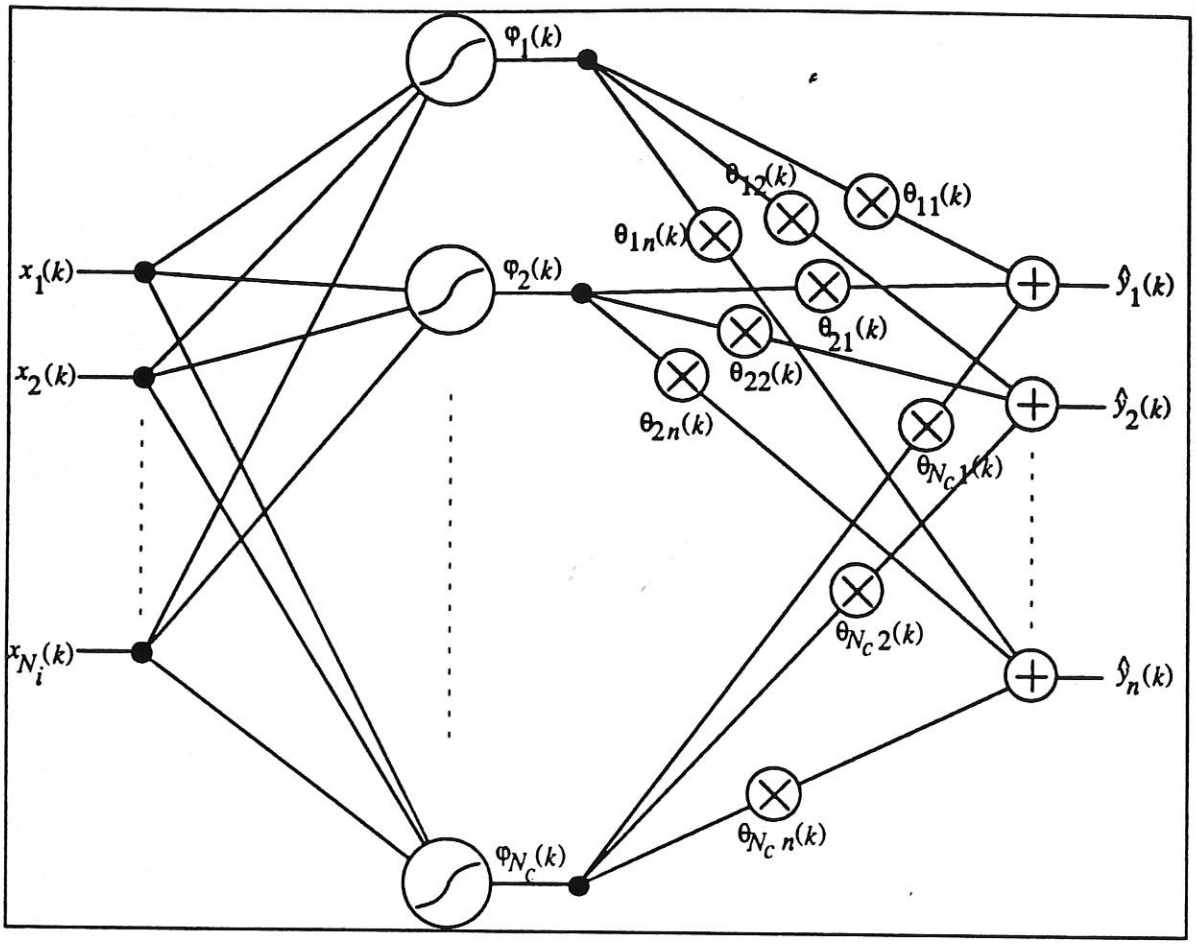

Figure 1 RBF network without linear and dc links.

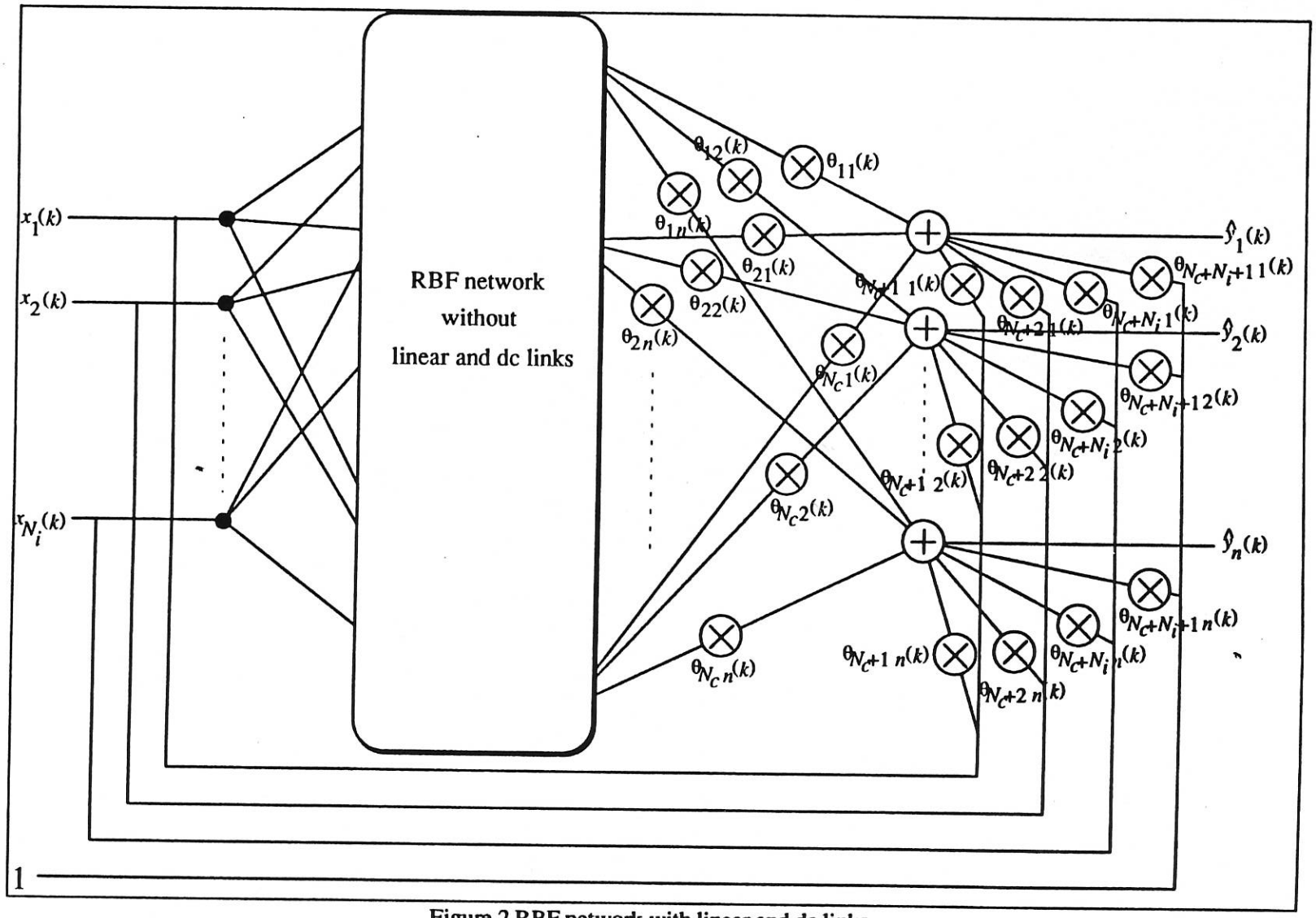

Figure 2 RBF network with linear and dc links.

Figure 3 Optimal decision region for the XOR example

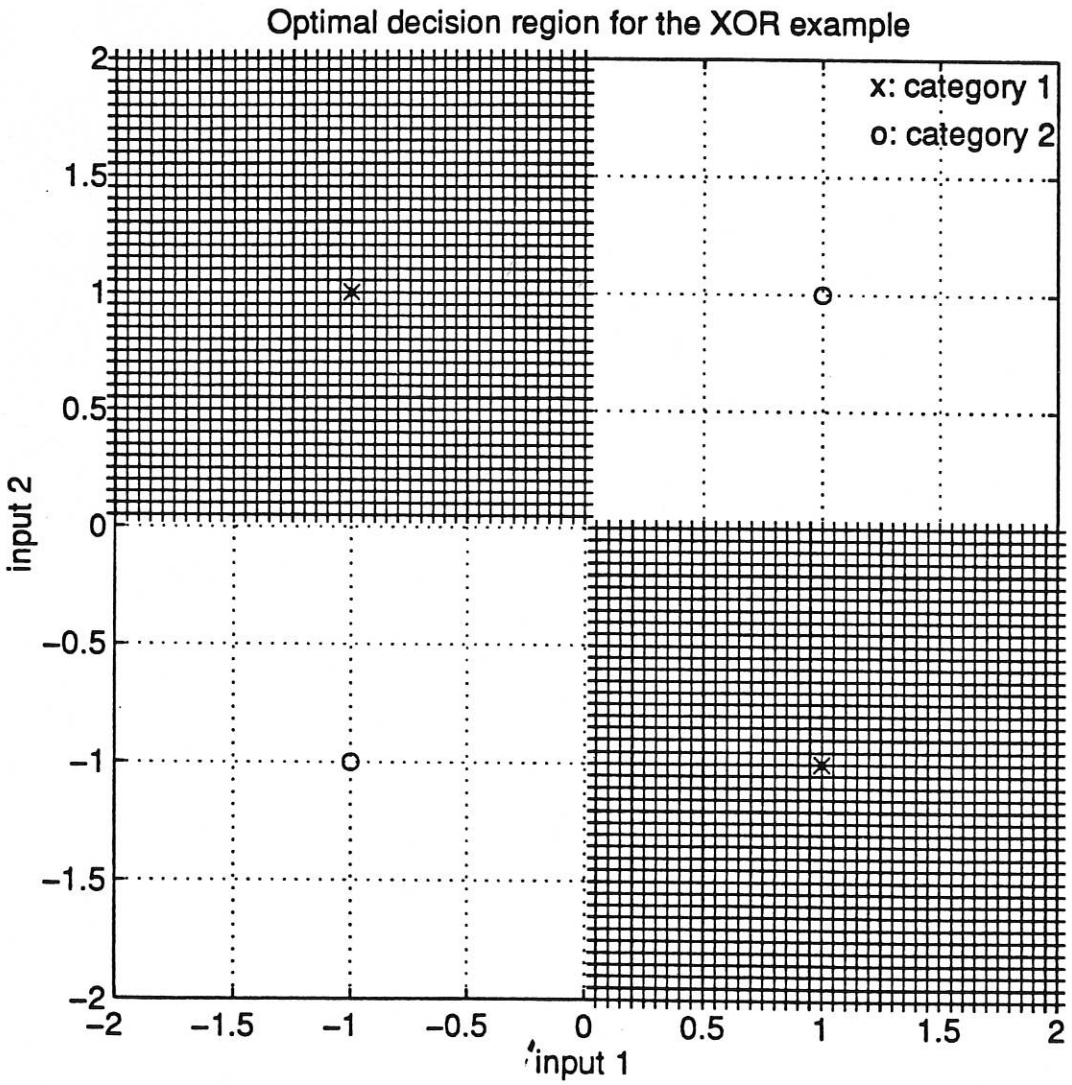

Figure 4 Decision region generated by the tained RBF network for the XOR example

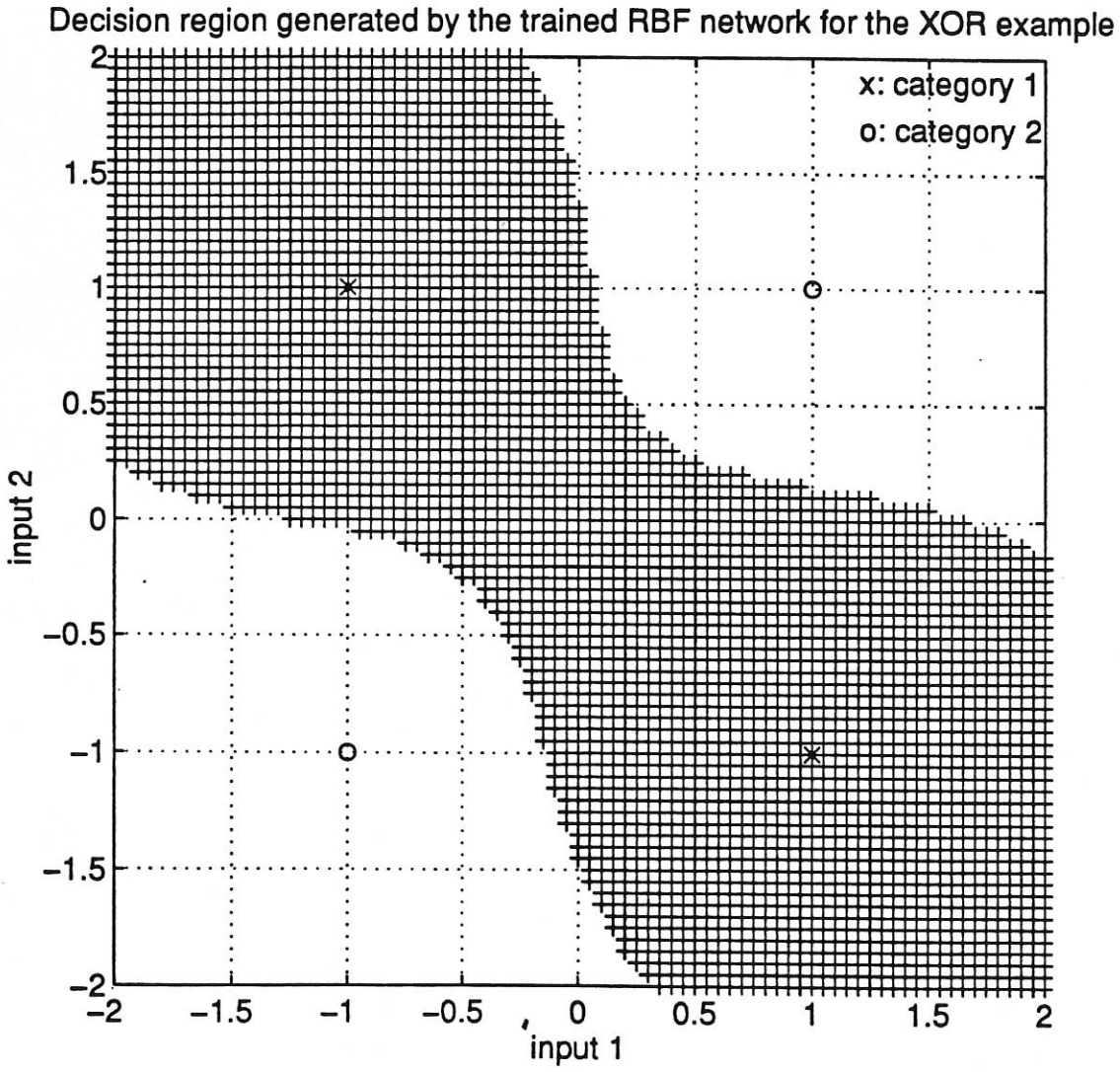

Figure 5 Tracking of the liquid level system example

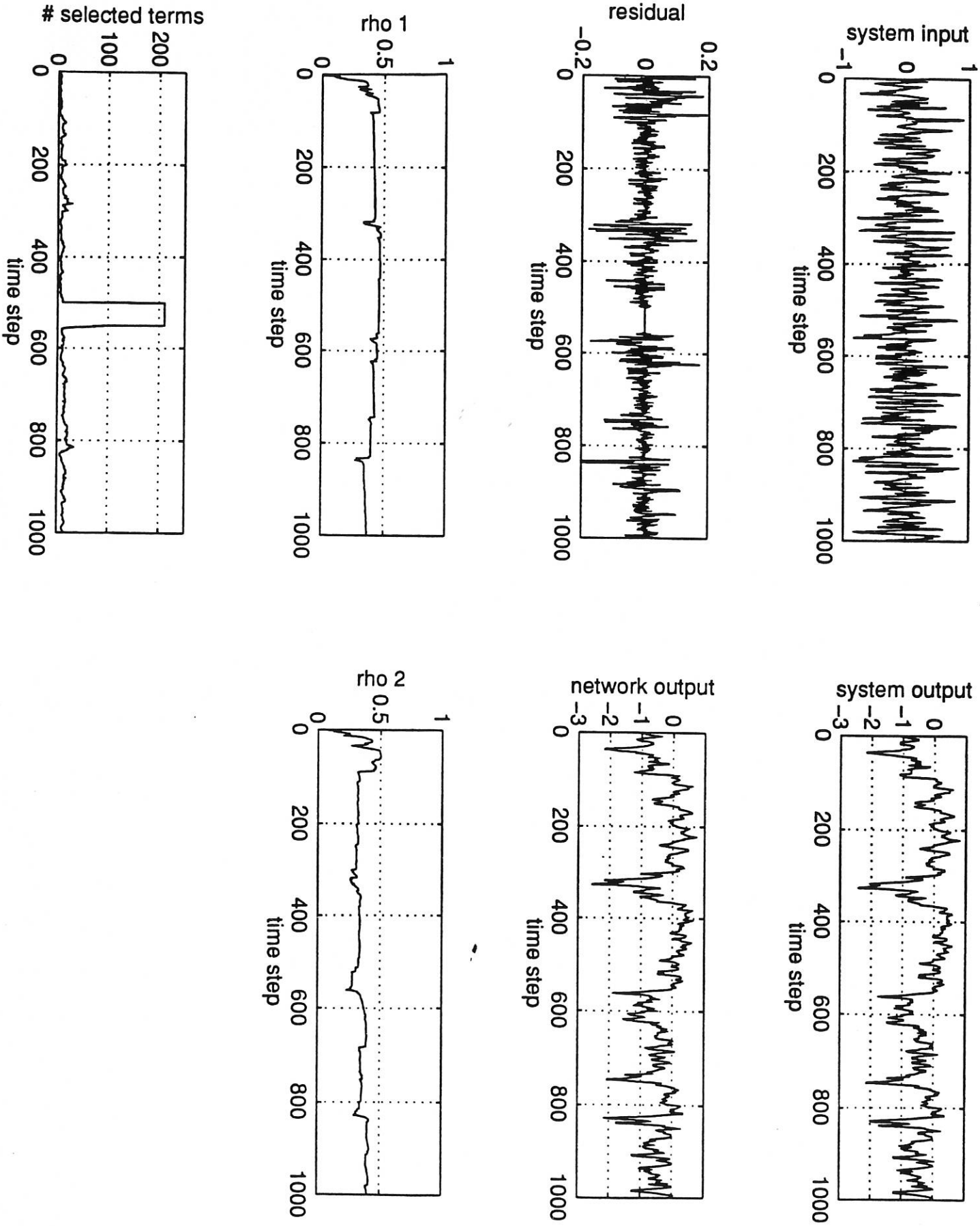

Figure 6 Tracking of the multi-input, multi-output system example