

# NEURAL NETWORK STRUCTURES FOR ON-LINE IDENTIFICATION AND OPTIMIZATION OF NONLINEAR PROCESSES

F.A. Cubillos<sup>(1)</sup>, G.P. Acuña<sup>(2)</sup>, and E.L. Lima<sup>(3)</sup>

(1) Depto. Ing. Química, (2) Depto. Ing. Informática, Universidad de Santiago de Chile.

Casilla 10233, Santiago, Chile. Fax:56-2-6817135, email: fcubillo@lauca.usach.cl

(3) Programa de Engenharia Química, COPPE, Universidade Federal do Rio de Janeiro. C.P.68502 CEP 21945-970, Rio de Janeiro, RJ, Brasil.

## ABSTRACT

This paper analyzes various formulations for the recursive training of neural networks that can be used for identifying and optimizing nonlinear processes on line. The study considers feedforward type networks (**FFNN**) adapted by three different methods. The study is completed using two network structures that are linear in the parameters: a radial basis network (**RBF**) and a principal components (**PCA**) network, both trained using a recursive least squares algorithm. The corresponding algorithms and a comparative test consisting of the on-line estimation of a reaction rate are detailed. The results indicate that all the structures were capable of converging satisfactorily in a few iteration cycles, FFNN type networks showing better prediction capacity than linear parameter networks, but the computational effort of the recursive algorithms is greater.

Submitted to **Brazilian Journal of Chemical Engineering**

## 1 INTRODUCTION

Neural networks are parallel-processing nonparametric structures that replicate on a small scale some of the operations seen in biological systems, such as learning and adaptation. The versatility of the use of neural networks is due to their potential to emulate a series of behaviors associated with human knowledge (Gupta and Rao, 1994). The attractive properties of these models, such as their ability to approximate nonlinear spaces, noise tolerance and ease of construction, among others, have contributed to a growing interest in both the academic and industrial fields. In recent years the use of neural networks as dynamic models of nonlinear processes has been studied extensively. Hussain (1999) has provided a complete review of this aspect.

Artificial neural networks, or simply neural networks, are made of a series of processing units, called "nodes" or "neurons," arranged in layers and densely connected to one another. The most important elements within a neural network are its nodes, which process the input information and deliver an output that is transmitted to other nodes in the network. Figure 1 shows a general scheme of a node that receives a vector of external stimuli  $\underline{x}$ . From the mathematical viewpoint, two basic operations can be distinguished in a node; the confluency operation and the activation operation.

The confluency operation ( $R^n \rightarrow R^1$ ) occurs between psynaptic weights (or simply weights) and external inputs to the node plus a bias term, and it is usually defined as the internal product, except in the so-called radial basis networks (**RBF**) in which the confluency is defined as the Euclidian distance. From an information processing viewpoint, the acquired experience or knowledge is stored in the psynaptic weights, while the process called the network's "learning" or "adaptation" consists in the modification of those weights, guided by a "training algorithm."

The activation operation ( $R^1 \rightarrow R^1$ ) is usually a nonlinear transformation on the confluency operation by means of an activation function  $f(\bullet)$  and it provides a limited output from the node, usually between [0,1] or between [-1,1]. Different kinds of activation functions are used in practice, such as unipolar sigmoid and bipolar (**Tanh**); linear with saturation; limit functions (hard limits); polynomials; orthogonal functions; etc.

In process identification, which involves the functional approximation task, the most frequently used structure is the feedforward neural network (**FFNN**) with a hidden layer, using sigmoids as activation functions, because it has been shown that it is capable of approximating any continuous nonlinear function (Hornik et al., 1989).

Traditionally, this kind of network is trained using the backpropagation method, which is coherent with the network's parallel structure. However, since it is a first order

method, it converges slowly, which makes it prohibitive for real time control applications. Narendra and Parthasarathy, 1990, Ydstie, 1990, Lee and Park, 1992, and Chen F and Khalil, 1992, studied neural controllers adapted recursively by backpropagation, achieving good results only after a significant number of adaptations. Wilson and Martínez (2003) provide foundations, scope and solutions for the training of networks by means of first order methods.

Because of the above, the development of more powerful training algorithms for feedforward type networks and the study of other more attractive neural structures is the subject of intensive studies. Among the former, the algorithms that use the Hessian matrix (second derivatives) or its approximations have given excellent results and have proved to be much faster with respect to training time, especially for the case of the approximation of continuous functions (Van der Smagt, 1994). Other methods based on intelligent search, such as genetic algorithms (Goldberg, 1989) –whose main advantage is to converge toward the global minimum– are being preferred for training networks with a significant number of parameters.

In relation to other structures, networks whose parameters are linear with respect to their outputs have been proposed, and they can be trained using robust recursive least squares (**RLS**) techniques.

In this paper we present a formulation for the recursive training of an **FFNN** neural network whose parameters can be adapted using three different methods: approximation of the inverse Hessian matrix using a **BFGS** (Broyden-Fletcher-Golfgarb-Shanno) algorithm; calculation of the inverse Hessian matrix in a Gauss-Newton recursive sequential (**NRS**) algorithm; and calculation of the inverse Hessian matrix in a Gauss-Newton recursive sequential algorithm implemented in parallel with respect to the nodes (**NRP**). We also complete the study using two network structures that are linear in the parameters: a radial basis (**RBF**) network and a principal component (**PCA**) network, both of them trained using a least squares recursive algorithm.

## 2 FEEDFORWARD TYPE NEURAL NETWORKS

### 2.1 STRUCTURE

Feedforward type neural networks have a structure ordered in layers whose nodes are connected in a direct sense with the nodes of the following layer, as shown in Figure 2. Given an input vector  $\underline{x}$  the network propagates and processes the information to generate an output vector  $\underline{y}$ . In the basic processing unit (node) the confluency operation is carried out based on the internal product given by

$$v_j = \sum_{i=1}^n \theta_{ji} * x_{ji} + \theta_{j0} \quad (1)$$

and the subsequent calculation of the output by means of a transformation given by  $\underline{y}_j = f(v_j)$  where  $f(.)$  is called the activation function. The most widely used activation functions are the sigmoids of type  $f(v) = 1/(1+e^{-v})$  and  $f(v) = \text{Tanh}(v)$ .

## 2.2 BACKPROPAGATION ALGORITHM

The most widely used training method for an **FFNN** neural network is the so-called backpropagation introduced by Rumelhart in 1986 (Rumelhart and McClelland, 1986), in which the weights of the network are adjusted based on the negative direction of the gradient of a cost function  $J(\mathbf{e})$  that quantifies the error between the output generated by network " $\underline{y}$ " and the desired output " $\underline{d}$ ". The cost function to be minimized is usually the mean quadratic error, given by

$$J(\mathbf{e}) = \frac{1}{2} \sum_{i=1}^m e_i^2 = \frac{1}{2} (\underline{y}-\underline{d})^t * (\underline{y}-\underline{d}) \quad (2)$$

The weight and bias vector of network  $\underline{\theta}$  is adapted according to:

$$\underline{\theta}(k+1) = \underline{\theta}(k) - \eta \underline{\nabla}(J) \quad (3)$$

where  $\eta$  is a constant parameter called the training rate.

Gradient  $\underline{\nabla}(J)$  can be obtained readily using the rules of differential calculus. An interesting way of showing the coherent parallelism between this method and the structure of the **FFNN** network was given by Narendra and Parthasarathy, 1988, by means of a diagrammatic representation of backpropagation. This representation, shown in Figure 3, makes it possible to see the flow of information needed to obtain  $\underline{\nabla}(J)$  and the possibility of integrating it in a parallel structure.

Figure 3 shows the feedforward direction of the information to generate the output vector  $\underline{y}$  from an input vector  $\underline{x}$ . On the other hand, it shows the backward way required to calculate the gradients with respect to the weights, using the information from the errors incurred ( $\mathbf{e}$ ), the confluencies calculated in the direct call ( $\mathbf{v}, \mathbf{v}'$ ), the derivative of the activation function ( $f'$ ), and the inputs external to the network and to the hidden layer ( $\underline{x}$  y  $\underline{x}'$ ).

Although the simplicity and the parallel nature of the backpropagation method have allowed its use in various applications, the need for a large number of iterations to achieve convergence has limited its use in real time adaptive applications. To improve this situation, minimization methods that include or approximate the information from the second derivatives (Hessian matrix), such as the quasi-Newton or Gauss-Newton, have been proposed by various authors (Leonard and Kramer 1990; Chen, S. et al., 1990;

Chen,G. and Ogmen, 1992; Van der Smagt, 1994). In these methods the parameters are adapted according to

$$\underline{\theta}(k+1) = \underline{\theta}(k) - [\underline{H}_{k+1}]^{-1} * \underline{\nabla}(J)_{k+1} \quad (4)$$

where  $[\underline{H}_{k+1}]$  is an approximation of the Hessian matrix ( $np \times np$ ).

### 2.3 ALGORITHMS FOR RECURSIVE ADAPTATION OF FFNN NETWORKS

With the purpose of adapting recursively the weights of a feedforward neural network, a cost function that weights the past information and the new information incorporated into the system must be defined conveniently. If we assume that we have sampled the process for k instants, we can define a weighted cost function between the values calculated by the neural network " $\underline{y}$ " and those desired " $\underline{d}$ " as follows:

$$\underline{J}_k = \frac{1}{2} \sum_{i=1}^k \beta^{(k-i)} * (\underline{y}-\underline{d})^t * (\underline{y}-\underline{d}) \quad (5)$$

where  $\beta \leq 1$  is a constant called the forgetting factor.

If we incorporate the next pair of data at the instant (k+1) we would have

$$\underline{J}_{k+1} = \beta * \underline{J}_k + \frac{1}{2} * (\underline{y}-\underline{d})^t * (\underline{y}-\underline{d})_{k+1} \quad (6)$$

From this result the following recursive relation can be obtained to calculate the gradient:

$$\underline{\nabla}(J)_{k+1} = \beta * \underline{\nabla}(J)_k - ([\Psi]^t * (\underline{d}-\underline{y}))_{k+1} \quad (7)$$

where  $[\Psi]$  is the Jacobian matrix of  $\underline{y}$  (network outputs) with respect to the parameters, which can be calculated by the backpropagation method. The way in which the inverse Hessian matrix is determined gives rise to a varied series of adaptation algorithms. In this work three different recursive type variants are studied:

#### i) Newton Recursive Sequential (NRS) Method:

In this method only some approximations are made to obtain analytically the Hessian matrix, which can be obtained recursively differentiating equation 7 with respect to the parameters:

$$[\mathbf{H}]_{k+1} = \beta * [\mathbf{H}]_k - ([\Psi]^t * (\mathbf{d} - \mathbf{y}))_{k+1} + ([\Psi]^t * [\Psi])_{k+1} \quad (8)$$

If the errors  $(\mathbf{d} - \mathbf{y})$  are considered to be small, the second term of equation (8) can be omitted. Therefore a recursive form for the Hessian matrix is:

$$[\mathbf{H}]_{k+1} = \beta * [\mathbf{H}]_k + ([\Psi]^t * [\Psi])_{k+1} \quad (9)$$

From this relation, the inverse Hessian matrix can be calculated using the following matrix inversion lemma:

$$\begin{aligned} \text{Lemma: if } & [\mathbf{X}]^{-1} = [\mathbf{Y}]^{-1} + [\mathbf{Z}]^t * [\mathbf{R}]^{-1} * [\mathbf{Z}] \\ \text{then } & [\mathbf{X}] = [\mathbf{Y}] - [\mathbf{Y}] * [\mathbf{Z}]^t * [\mathbf{Z}\mathbf{Y}\mathbf{Z}^t + \mathbf{R}]^{-1} * [\mathbf{Z}] * [\mathbf{Y}] \end{aligned}$$

By applying the lemma, a recursive formula for calculating  $[\mathbf{H}]^{-1}$  is deduced directly:

$$\beta * [\mathbf{H}]_{k+1}^{-1} = [\mathbf{H}]_k^{-1} - [\mathbf{H}]_k^{-1} * [\Psi]^t * ([\Psi] * [\mathbf{H}]_k^{-1} * [\Psi]^t + \beta * \mathbf{I})^{-1} * [\Psi] * [\mathbf{H}]_k^{-1} \quad (10)$$

For **MISO** (Multi Input Single Output) systems, the inverse term is scalar and the calculation of  $[\mathbf{H}]^{-1}$  does not require matrix inversion.

The recursive estimation algorithm is summarized in the following sequence:

- i) Initialization of  $\beta$ ,  $[\mathbf{H}]^{-1}$ ,  $\underline{\theta}$
- ii) Estimation:
  - Collect a pair of data
  - Calculate the estimated value  $\underline{y}$  (network output)
  - Calculate the error  $(\mathbf{d} - \mathbf{y})$
  - Calculate  $[\Psi]$  by backpropagation
  - Calculate  $\underline{\nabla}(\mathcal{J})$  in equation (7)
  - Calculate  $[\mathbf{H}]^{-1}$  in equation (10)
  - Update parameters, equation (4)
  - Repeat.

## ii) Newton Recursive Parallel (NRP) Method

The disadvantages of the previous method are the centralized structure, the complexity of the calculation of the inverse Hessian matrix, and the information storage requirements. To achieve a certain parallel structure in the algorithm and relieve the storage load, Chen S. et al, 1990, propose estimating an inverse Hessian matrix for each

of the network's nodes. In this way it is possible to divide the calculation of equation (10) into  $n$  sub-algorithms, where  $n$  is the number of nodes of the neural network

This algorithm uses  $n$  inverse Hessian sub-matrix that are adapted recursively using equation (10). The capacity needed to store  $n$  inverse Hessian sub-matrix is smaller than that required to store a single matrix in the centralized version. Another advantage lies in its parallel structure by nodes; however, it requires a greater effort to tune and initialize the parameters of each of the estimators. The estimation sequence is identical to that of the previous method, except that it is subdivided  $n$  times.

### iii) *Quasi-Newton Methods*

These methods (Van der Smagt,1994) provide an approximation of the inverse of the Hessian  $[A]$  when the following relation is applied at the limit  $k \rightarrow \infty$ :

$$\underline{\theta}_{k+1} - \underline{\theta}_k = [A]_{k+1}^{-1} (\underline{g}_{k+1} - \underline{g}_k) \quad (11)$$

with  $\underline{g} = -\nabla(J)$ .

Two widely used adaptation formulas, **BFGS** (Broyden-Fletcher-Golfarb-Shanno) and **DFP** (Davidon-Fletcher-Powell), can be derived from equation (11). In particular, in this paper we analyze the **BFGS** method, whose incremental adaptation form (Edgard and Himmelblau, 1988) is:

$$\Delta[A] = [A]_{k+1} - [A]_k = [(\Delta\underline{\theta} - [A]_k * \Delta\underline{g}) \Delta\underline{\theta}^t + \Delta\underline{\theta}^t * (\Delta\underline{\theta} - [A]_k * \Delta\underline{g})^t] / (\Delta\underline{g}^t * \Delta\underline{\theta}) - [(\Delta\underline{\theta} - [A]_k * \Delta\underline{g})^t * \Delta\underline{g} * \Delta\underline{\theta} * \Delta\underline{\theta}^t] / [(\Delta\underline{g}^t * \Delta\underline{\theta}) * (\Delta\underline{g}^t * \Delta\underline{\theta})] \quad (12)$$

Although this formulation requires storing  $[A]$ , it is computationally simpler than the previous algorithms, since it only requires calculating the gradients.

## 3 NETWORKS THAT USE PRINCIPAL COMPONENT ANALYSIS

### 3.1 STRUCTURE

The incorporation of principal component analysis (**PCA**) to neural networks of the multilayer type was introduced by Pell et al. in 1992. In their proposal they adopt a structure with a hidden layer in which the weights of the first layer are calculated in such a way that the input vector is transformed into principal components. The principal components analysis technique consists in decomposing the input vector  $\underline{x}$  (with a mean of zero and variance of 1) according to:

$$\underline{\mathbf{x}} = [\mathbf{U}] * [\Sigma^{1/2}] * [\mathbf{V}]^t \quad (13)$$

Such that:  $[\mathbf{V}]^t * [\mathbf{V}] = [\mathbf{V}] * [\mathbf{V}]^t = \mathbf{I}$  and  $[\mathbf{U}]^t * [\mathbf{U}] = [\mathbf{U}] * [\mathbf{U}]^t = \mathbf{I}$

The columns of matrix  $[\mathbf{V}]$  are known as the singular vectors of the right, and the diagonal matrix  $[\Sigma^{1/2}]$  contains the singular values  $\sigma$ , arranged from large to small. The decomposition described above is called singular value decomposition (**SVD**), with various robust methods described in linear algebra. The principal components of  $\underline{\mathbf{x}}$  generate an orthogonal base and are given by:

$$[\mathbf{P}] = \underline{\mathbf{x}} * [\mathbf{V}] \quad (14)$$

A data matrix consisting of  $m$  observations of  $n$  variables  $[\mathbf{X}]_{m \times n}$  has  $n$  principal component vectors  $\mathbf{p}_i$  ( $i = 1 \dots n$ ) given by the columns of  $[\mathbf{P}]$ . The variance of each vector  $\mathbf{p}_i$  is given by the corresponding singular value  $\sigma_i$ , so the first vector,  $\mathbf{p}_1$ , is the one that has the largest variation,  $\mathbf{p}_2$  the second largest, and so on.

The application of the **PCA** technique to a data matrix in a neural network has two great advantages:

- i) Inputs to the hidden layer are linearly independent of one another because they were transformed into principal components.
- ii) An analysis of the singular values ( $\sigma_i$ ) makes it possible to eliminate the nodes of the hidden layer that do not contribute significantly to the variations of the data (those that have small singular values).

With the purpose of conserving the nonlinear characteristics, other activation functions of the polynomial type may be used as an approximation to the sigmoidal functions used in feedforward type networks, i.e.,

$$f(v) = a_0 + a_1 * v + a_2 * v^2 + \dots + a_p * v^p = \sum a_i * v^i \quad (15)$$

Figure 4 illustrates the topology of the **PCA** network, where the weights between the inputs and the hidden layer are the singular vectors of the input matrix used in the training. The nodes of the hidden layer contain the polynomial functions of degree  $p$ , while the output is calculated as the confluency of all the nodes. When the number of principal components is equal to the dimension of the input vector, then the network is said to be "symmetric,"

It is convenient to define a parameters vector  $\underline{\omega}$  that contains the product between the weights of the second layer and the coefficients of the polynomials of the activation functions. In this way it can be shown that the network's output  $\underline{y}$  can be calculated by the following linear relation (Peel et al, 1992):

$$\underline{y}^t = [\psi]^* \underline{\omega} \quad (16)$$

where  $[\psi]$  is a matrix defined by

$$[\psi] = [1 \ p_1 \ p_1^2 \ p_1^3 \ \dots \ p_1^p : 1 \ p_2 \ p_2^2 \ \dots \ p_2^p : \dots : 1 \ p_n \ p_n^2 \ \dots \ p_n^p] \quad (17)$$

If the singular vectors are kept constant, the parameters vector  $\underline{\omega}$  can be calculated using the traditional regression techniques for linear systems.

### 3.2 TRAINING

Training of this kind of network may be summarized in the following steps:

- i) Obtain the singular vectors of the input matrix  $[\mathbf{X}]$ , previously scaled to a mean of zero and a variance of 1.
- ii) Calculate the corresponding principal components (equation 14)
- iii) Determine the matrix  $[\psi]$  as a function of the degree of the polynomial  $p$  and the number of principal components to be used
- iv) Calculate the parameters vector  $\underline{\omega}$  (equation 16) by some linear fitting method.

In this kind of network the structural parameters are the number of principal components (number of nodes in the intermediate layer) and the degree of the polynomial  $p$ . The best set of these parameters is the one that has the lowest prediction error over a different set used in the training. As a general rule it is convenient to choose initially a symmetric network (number of principal components equal to the number of input nodes) and change systematically the order of the polynomial.

#### Recursive Adaptation

Because the determination of the weights of these kinds of networks is reduced to a linear regression problem, this same structure can be adapted recursively using the robust techniques of recursive least squares, provided the topology of the network and its singular vectors, which are considered constant during the adaptation, are specified beforehand.

If a large variation in the structure of the input data is seen (sectorization or alteration of information content), a continuous updating of the singular vectors over a mobile data window can be implemented. This implementation can be developed as an unsupervised neural network, as shown by Oja, 1992. Therefore, this kind of network can have a

totally parallel character, with a first zone in charge of calculating the principal components and another in charge of generating the outputs.

## 4 NETWORKS WITH RADIAL BASIS FUNCTIONS (RBF)

### 4.1 TOPOLOGY

Networks with radial basis functions (**RBF**) are the predecessors of feedforward neural networks and are traditionally used as a method for the interpolation of multidimensional spaces. Currently, and in view of their structure, they are considered as a kind of neural network and have been used successfully for process identification and control by Narendra and Parthasarathy, 1991; Chen, S. et al., 1991; Pottmann and Seborg, 1992; and Thompson M. and Kramer, 1994. These networks are a good alternative to multilayer type networks for identifying processes, since the outputs of an **RBF** network are linear with respect to their parameters.

The **RBF** type networks consist of a layer of several nodes in which the activation function acts on the Euclidian norm of the difference between the input vector  $\underline{x}$  and a vector  $\underline{c}$  called the "center," which is specific for each node. The network's output is the result of a linear combination between the outputs of the nodes and a weight vector  $\underline{\theta}$ . Figure 5 shows a scheme of such a network.

The most widely used activation functions are:

- Gaussian  $f(v) = \exp(-v/\mu) ; \mu > 0$
- Multiquadratic  $f(v) = (v+\mu)^{1/2} ; \mu > 0$
- Reciprocal multiquadratic  $f(v) = (v+\mu)^{-1/2} ; \mu > 0$
- Degree n spline  $f(v) = v^n \log(v)$

where  $v$  is the result of the confluency operation, defined as the Euclidian distance between the input vector  $\underline{x}$  and the center of node  $i$ ,  $\underline{c}_i$ , and is given by

$$v_i = \|\underline{x} - \underline{c}_i\| = [(\underline{x}_1 - \underline{c}_{i1})^2 + (\underline{x}_2 - \underline{c}_{i2})^2 + \dots + (\underline{x}_n - \underline{c}_{in})^2]^{1/2} \quad (18)$$

The network's output is given by the weighted sum of the outputs of the corresponding basis according to

$$\underline{y} = \sum_{i=1}^{nb} f(v_i) * \theta_i \quad (19)$$

The greatest difficulty in using these structures lies in an adequate choice of their topology, which involves the choice and the number of centers as well as the kind of parameters of the activation function. In the field of the identification of chemical processes, Pottmann and Seborg, 1992, propose a methodology based on statistical tests to determine automatically the topology of the network and the centers. With examples for a simulated reactor and a pilot scale neutralization system, the authors show the method's effectiveness; however, it is not quite feasible to implement it in a recursive way.

Historically, the number of basis has been determined by statistical analysis or cross-correlation, while the centers have been chosen basically by random values or values obtained from the same training set. An interesting technique for choosing the centers was suggested by Moody and Darken, 1989, based on an algorithm that divides the set of data into  $n$  sectors, producing a set of  $n$  centers, minimizing the mean quadratic error incurred by representing the data set by the  $n$  centers. These centers can be part of the same data set or arbitrary values. One of the main characteristics of this way of choosing, called ***n-mean clustering***, is that it can be implemented recursively allowing a continuous adaptation of the centers.

## 4.2 RECURSIVE ADAPTATION OF RBF NETWORKS

Chen et al., 1991, present an algorithm for recursive training of **RBF** networks consisting of two sub-algorithms, one based on ***n-mean clustering*** to adapt the centers, and the other, a recursive least squares routine, to determine the network's output parameters.

The ***n-mean clustering*** recursive algorithm (**NMCR**) is not supervised and may be summarized in the following steps:

i) Start: Give random initial values to  $\underline{c}_n$  centers and specify an initial training speed  $\alpha(0)$ .

ii) Incorporate a new input vector  $\underline{x}(k)$ .

iii) Calculate distances and find the minimum distance sector.

$$v_i(k) = |\underline{x}(k) - \underline{c}_i(k-1)|, \quad 1 \leq i \leq n$$

$$c_{imin} = \arg [\min v_i(k)] \quad (20)$$

iv) Adaptation of minimum distance center

$$\underline{c}_i(k) = \underline{c}_i(k-1) ; \quad 1 \leq i \leq n ; \quad i \neq imin$$

$$\underline{c}_{imin}(k) = \underline{c}_{imin}(k-1) + \alpha(k) * [x(k) - \underline{c}_{imin}(k-1)] \quad (21)$$

v) Return to ii.

The training speed  $\alpha(k)$  must be less than 1 and should decrease slowly to zero. In systems with a dynamically variable input structure,  $\alpha(k)$  can be kept at a small and constant value to ensure a continuous adaptation of the centers. The recursive identification algorithm can be carried out sequentially by adapting the centers at every instant and then calculating the output weights  $\underline{\theta}$  by means of the *RLS/VFF* algorithm

## 5 APPLICATION EXAMPLE: Identification of a kinetic parameter.

### 5.1 DEFINITION OF THE PROBLEM

The various algorithms and structures given above are used below for the recursive training of neural structures that are capable of predicting a kinetic parameter as a function of input and output variables obtained in a nonisothermal highly linear reactor **CSTR** in which the first order reversible reaction  $\mathbf{A} \Leftrightarrow \mathbf{R}$  takes place. The final objective is to develop a predictive-adaptable controller based on a hybrid-neural model of the process (See Section VI for further details). The neural structure must predict the kinetic parameter  $\mathcal{Y}$  when the actual concentration of product  $\mathbf{R}$  and input temperature to the reactor  $\mathbf{Ti}$  are known for present and past instants, i.e.

$$\mathcal{Y} = \mathcal{NM}(\mathbf{R}_k, \mathbf{Ti}_k, \mathbf{Ti}_{k-1}) \quad (22)$$

The data set obtained by means of simulations using a phenomenological model of the process is shown in Figure 6, where  $\mathbf{Ti}$  and  $\mathcal{Y}$  are dimensionless between [-1 and 1].

A total of six recursively adapted neural structures were evaluated, whose main characteristics are given in Table 1. The structural parameters such as the number of nodes in the hidden layer, the order of the polynomial in the **PCA** network, and the number of basis in the **RBF** network were determined by the **AGTSE** cross-correlation criterion (Pollard et al., 1992) using another data set.

### 5.2 IMPLEMENTATION AND RESULTS

The **FFNN** networks were initialized with the following parameters:  $\beta=0.99$ ,  $[\mathbf{H}]^{-1}=10.000*\mathbf{I}$  and  $\underline{\theta}$  with random values between -1 and 1. In order to make the estimation more robust, the trace of the  $[\mathbf{H}]^{-1}$  matrix was upper bounded in the three algorithms analyzed, using the method of Salgado et al., 1988, given by

$$\mathbf{H}^{-1} = [\text{Tr} / \text{Trace}(\mathbf{H}^{-1})] * \mathbf{H}^{-1} ; \text{if } \text{Trace}(\mathbf{H}^{-1}) > \text{Tr} . \quad (23)$$

A value of  $\mathbf{Tr} = 10^4$  was used in the centralized algorithms, and a value of  $\mathbf{Tr}=50$  for each of the five matrixes of the parallel algorithm (corresponding to the five nodes of the network). In the **PCA** network, the singular vectors were extracted from a **DVS** of the first 30 data using the ortho-normalization method of Gram-Schmidt (**GSO**), and they were considered constant during the adaptation. The singular values were the following:  $\boldsymbol{\sigma} = [3.9713 \ 1.501 \ 0.8035]$ .

In the case of the **RBF** network the 12 centers were initialized with random values between -1 and 1 for  $\mathbf{Ti}$  and between 0.2 and 0.6 for  $\mathbf{R}$ . A recursive adjustment of the centers was implemented using the **NMCR** algorithm described earlier, with a heuristic for the training speed similar to that suggested by Chen et al., 1991, given by

$$\begin{aligned} \alpha(k) &= \alpha(k-1) / ( 1 + 0.05 * \mathit{int}(k/12) ) ; \quad \text{if } \alpha(k-1) > 0.05 \\ \alpha(k) &= 0.05 \quad ; \text{if } \alpha(k-1) \leq 0.05 \end{aligned} \quad (24)$$

Figure 7 illustrates the distribution of the centers obtained with the **NMCR** algorithm after one adaptation cycle (120 data). A distribution that is quite representative of the data can be seen.

In the **PCA** and **RBF** networks the output parameters  $\underline{\theta}$  were estimated with a robust algorithm of recursive least squares with variable forgetting factor **RLS/VFF**, The initial parameters,  $\underline{\theta}$ , were given random values close to zero (0 to 0.01), and the covariance matrix of the parameters a value of  $10^4 * \mathbf{I}$ . The information content parameter of this algorithm,  $\boldsymbol{\Sigma}$ , was set at 0.01, as recommended by practice (Kershenbaum and Perez-Correa,1989).

The information was submitted sequentially to the estimators from  $k=1$  to  $k=120$ , and at every instant an adaptation occurred. Once a parameter adaptation had taken place, they were kept constant and the structure was used to predict the whole data set. Later, the instantaneous quadratic error  $\hat{e}$  between the output and the prediction was calculated, and it served as an indicator to evaluate the modeling capacity and convergence speed of each structure. The instantaneous quadratic error is given by

$$\hat{e}(k) = \sum_{i=1}^{120} (\hat{r}(\theta_k) - r)_i^2 \quad (25)$$

where  $\mathcal{Y}(\theta_k)$  is the value predicted by the neural structure with its parameters adapted at instant  $k$ .

The error  $\hat{\epsilon}$  is basically an indicator of the state of the estimation. If it decreases gradually, the algorithm is in the convergence phase and the identification is in progress. Otherwise, the identification diverges due to a deficiency in the structure or in the estimator's parameters. When  $\hat{\epsilon}$  tends to a constant value, it is indicative of a final convergence, and  $\hat{\epsilon}$  may be seen as the residual or identification error.

The results obtained for each of the structures analyzed are shown in Figures 8 and 9, and it is seen that all of them converge satisfactorily. The residual modeling error and the robustness of the estimator were analyzed by presenting the training set repeatedly over a certain number of cycles. Figures 10 and 11 show the error in the last adaptation of a cycle, i.e.  $\hat{\epsilon}(k=120)$  as a function of the number of times in which the set of data is presented to the estimator.

## 6 CONCLUSIONS

From the results obtained it is seen that all the structures analyzed were capable of adapting satisfactorily to the proposed example, converging in less than one presentation cycle. Cyclic repetitions also showed that the algorithms are stable and tend to different residual errors, providing an indication of the modeling capacity of each structure in particular. In this sense, feedforward type networks show less residual error due to their great nonlinear adjustment flexibility, in contrast with the **PCA** and **RBF** networks, which are more rigid.

The results also indicate differences between the convergence speeds, which are greater in the **PCA** and **RBF** networks -which practically converge for  $k > 65$  as shown in Figure 9- compared with the feedforward networks -which converge for  $k > 100$  (except **SS/NRS**) as seen in Figure 8-, due to the **RLS** adaptation algorithm, which guarantees global optima, and to the smaller number of parameters used.

Among the different feedforward network alternatives studied there are differences in both the convergence speed and the residual error, due basically to the simplifications introduced in the algorithm for adapting the inverse Hessian matrix. In fact, **BFGS** and **NRP** are approximations of **NRS** that imply some deterioration of the previously mentioned indicators.

Other characteristics such as the computational load of the adaptation and the ease of the structured modeling can also be analyzed. In general, feedforward networks are simpler to specify and initialize than **PCA** and **RBF** networks, however the algorithms are computationally more costly in floating point operations and storage requirements. On the other hand, in **PCA** and **RBF** networks specification of the intermediate layer

requires an additional effort, mainly in the **RFB** networks, in which the number of centers and the activation function has a marked influence on their performance.

### ***SYMBOLS***

$[A]$	: Inverse Hessian matrix approximation.
$\underline{c}$	: Center of a radial basis.
$\hat{e}$	: Instantaneous quadratic error.
$e$	: Prediction error of the network.
$\underline{d}$	: Desired output vector for the neural network.
$f(.)$	: Node activation function.
$g$	: gradient of the error.
$[H]$	: Hessian matrix of the neural network.
$int(x)$	: Whole part of x.
$J$	: Objective function to be minimized in the training of the neural network.
$k$	: Discrete time.
$m$	: Extension of the training data set.
$n$	: Input vector dimension.
$nb$	: Number of radial basis.
$np$	: Number of parameters.
$[P]$	: Principal component matrix.
$r$	: Net kinetic rate of product formation in <b>CSTR</b> .
$R$	: Product output concentration in <b>CSTR</b> .
$T_i$	: Reagent input temperature in <b>CSTR</b> .
$Tr$	: Constant trace algorithm parameter.
$u$	: Manipulated process variable.
$[V]$	: Singular vector matrix.
$vi$	: Result of the confluency operation of node i.
$\underline{x}$	: Neural network input vector.
$[X]$	: Neural network input matrix.
$\underline{y}$	: Neural network output vector.
$y$	: Measured process output.
$\hat{y}$	: Prediction of process output.
$\alpha$	: Parameter in the <b>NMSR</b> method.
$\beta$	: Exponential discount factor (forgetting factor).
$\eta$	: Training parameter in the backpropagation method.
$\theta$	: Weights of the neural network.
$[\Psi]$	: Jacobian matrix of the neural network.
$s$	: Singular values vector.

### **ACKNOWLEDGEMENT**

The authors gratefully acknowledge the financial support of **Fondecyt** through Projects 1020041 – 7020041 - 1040208

## REFERENCES

- Chen G. and Ogmen H.**, "Supervised learning via modified Extended Kalman Filtering," Proc. ACC, 1992.
- Chen F. and Khalil H.**, "Adaptive control of nonlinear system using neural networks", Int.J. Control, 55, 6, 1992.
- Chen S. Cowan C. Billings S. and Grant P.**, "Parallel recursive prediction error algorithm for training layered neural networks," Int.J.Control,51,6,1990.
- Chen S. Billings S. and Grant P.**, "Recursive hybrid algorithm for non linear system identification using radial basis functions networks," Int.J.Control, 55,5,1991.
- Edgard T. and Himmelblau D.** "Optimization of Chemical Processes," McGraw Hill Book Co., New York,1988.
- Goldberg D.**, "Genetic Algorithms in search, optimization and machine learning," Addison Wesley, USA,1989.
- Gupta M. and Rao D.**, "On the principles of fuzzy neural networks, Fuzzy Sets and Systems" , 61,1-18,1994.
- Henrique H., Lima E., and Seborg D.**, " Model structure determination in neural network models", Chemical Engineering Science, 55,5457-5469,2000.
- Holcomb T. and Morari M.**, "PLS/neural networks," Computers Chem. Engng.,16,4,1992.
- Hornik K. Stinchcombe M. and White H.**, "Multilayer feedforward networks are universal approximators ," Neural Network,2, 359-366,1989 .
- Hussain M.**, "Review of the applications of neural networks in chemical process control simulation and online implementation" -Artificial Intelligence in Engineering V13, N° 1, 55-68. 1999.
- Kershbaum L. and Perez R.**, "An extended horizon feedback-feedforward self tuning controller," AIChE J., 35,11,1989.
- Lee S. and Park S.**, "A new schema combining neural feedforward control with model predictive control," AIChE J.,38,2,1992.
- Leonard J. and Kramer M.**, "Improvement of the backpropagation algorithm for training neural networks," Computers Chem. Engng.,14,3,1990.
- Moody J. and Darken C.**, "Fast learning in networks of locally tuned processing units," Neural Comput. 1,281,1989.

**Narendra K. and Parthasarathy K.**, "A diagrammatic representation of backpropagation," Technical Report # 8815, Center of System Science, Yale Univ.,1988.

**Narendra K. and Parthasarathy K.**,"Neural networks and dynamical system: (II) Identification , (III) Control, IV," Technical Report Center of System Science, Yale Univ.,1990.

**Narendra K. and Parthasarathy K.**, "Stable adaptive control of a class of discrete time nonlinear system using radial basis neural networks," Technical Report # 9103, Center of System Science, Yale Univ.,1991.

**Oja E.**, "Principal components, minor components and linear neural networks," Neural Networks,5,1992.

**Peel C. Willis M. and Tham M.**, "A fast procedure for training neural networks," J.Proc.Control, 2,4,1992.

**Pollard J. Broussard M. Garrison D. and San K.**, Process identification using neural networks," Computers Chem. Engng.,16,4,253-270;1992.

**Pottmann M. and Seborg D.**, "Identification of nonlinear processes using reciprocal multiquadratic functions," J.Proc.Control, 2,4,1992.

**Rumelhart D. and McClelland J.**, "Parallel distributed processing," MIT Press,USA,1986.

**Salgado M. Goodwin G. and Middleton R.**, "Modified least squares algorithm incorporating exponential resetting and forgetting," Int.J.Control, 47,477,1988.

**Thompson M. and Kramer M.**, "Modeling chemical processes using prior knowledge and neural networks," AIChE J.,40,132,1994.

**Van der Smagt P.**, "Minimisation methods for training Feedforward neural networks," Neural Networks, 7,1,1994.

**Wilson R and Martinez T;** "The general inefficiency of batch training for gradient descent learning Neural Networks", Neural Networks, 16, N° 10, Pgs 1429-1451, 2003.

**Ydstie B.**, " Forecasting and control using adaptive connectionist networks," Computers Chem. Engng.,14,4/5,1990.

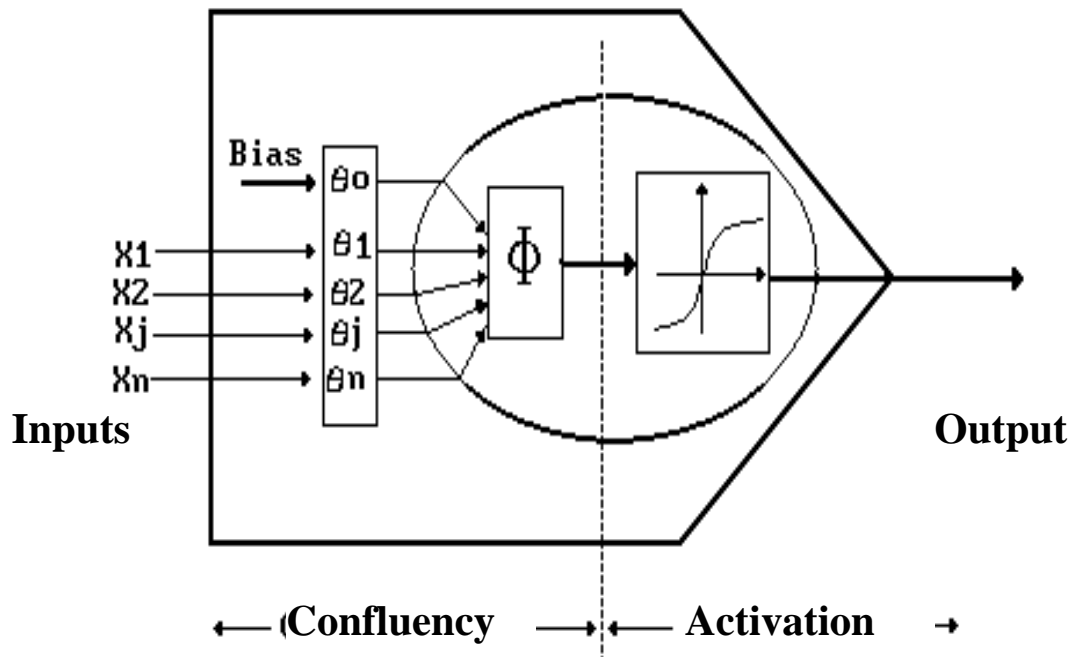


Figure 1. General scheme of a node ;  $\underline{x}$  is the input vector external to the node,  $\underline{\theta}$  are the psynaptic weights ,  $\Phi$  is the confluency operation between  $\underline{x}$  and  $\underline{\theta}$ .

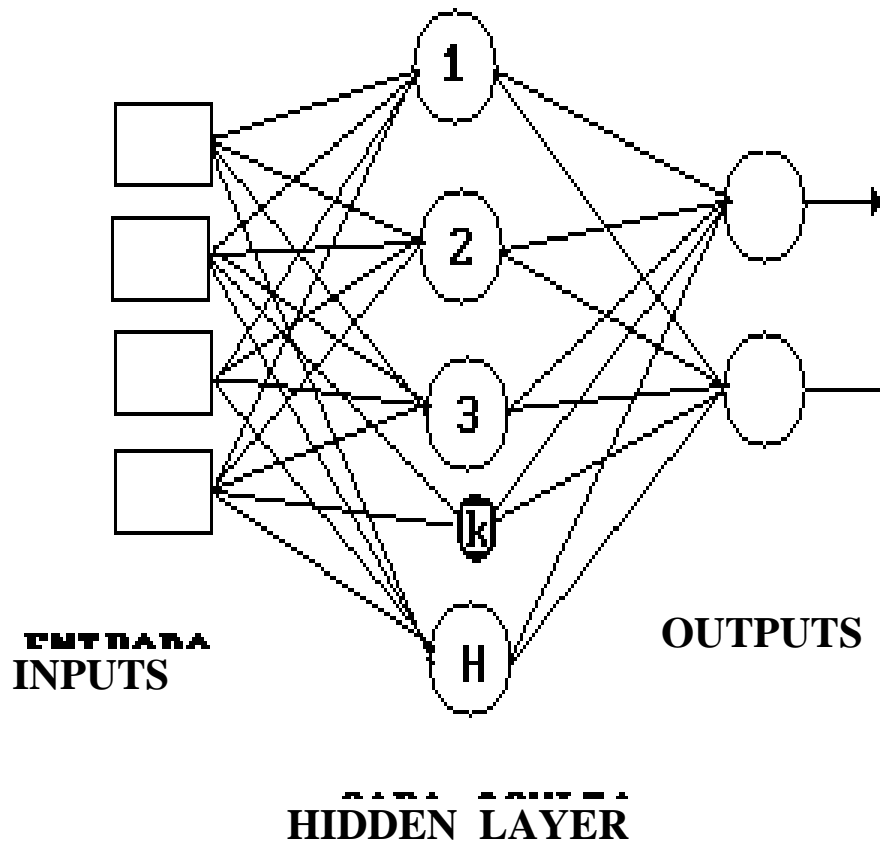


Figure 2 : Scheme of a feedforward type neural network



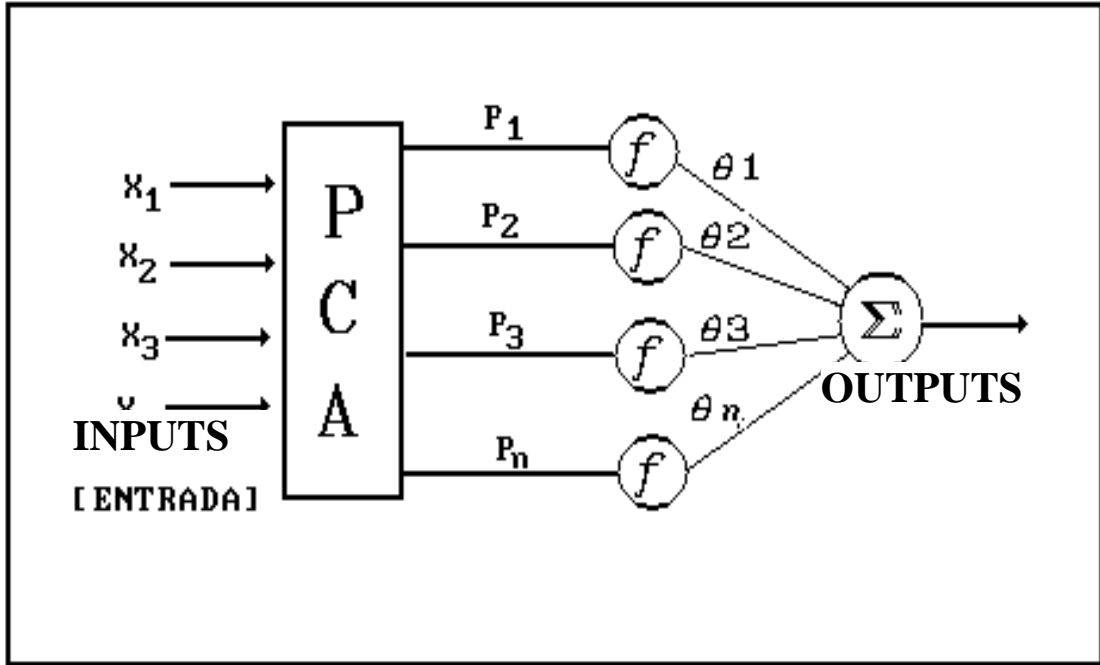


Figure 4: PCA network structure.

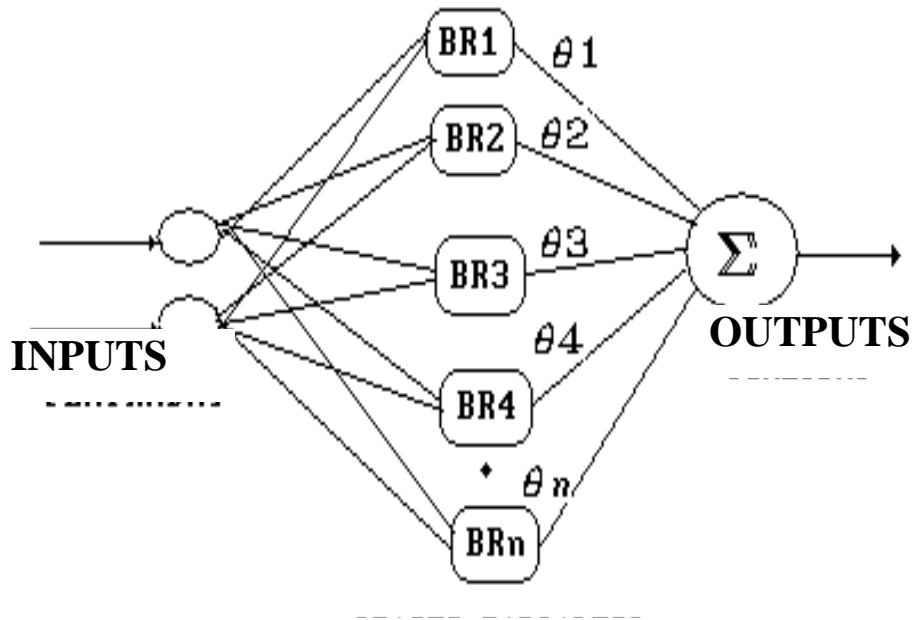
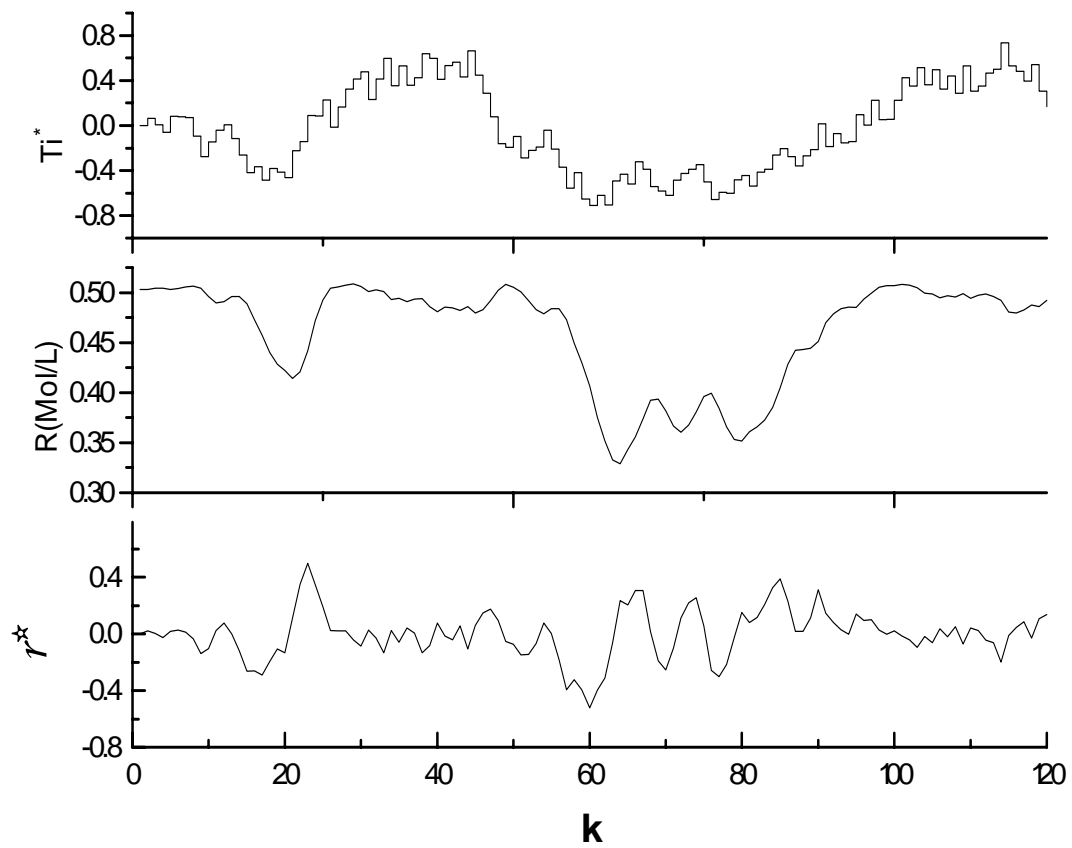
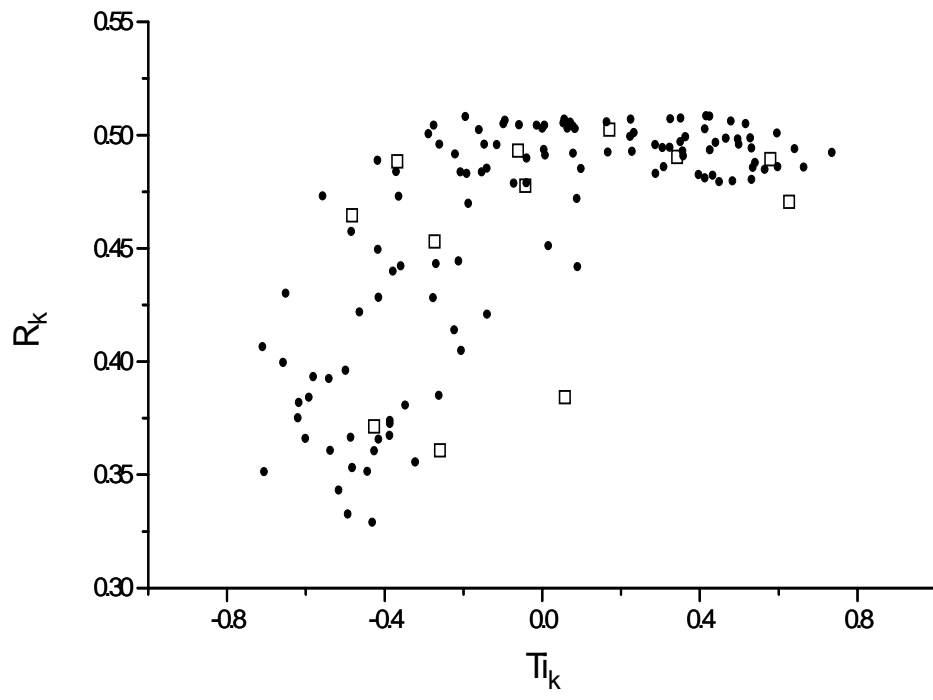


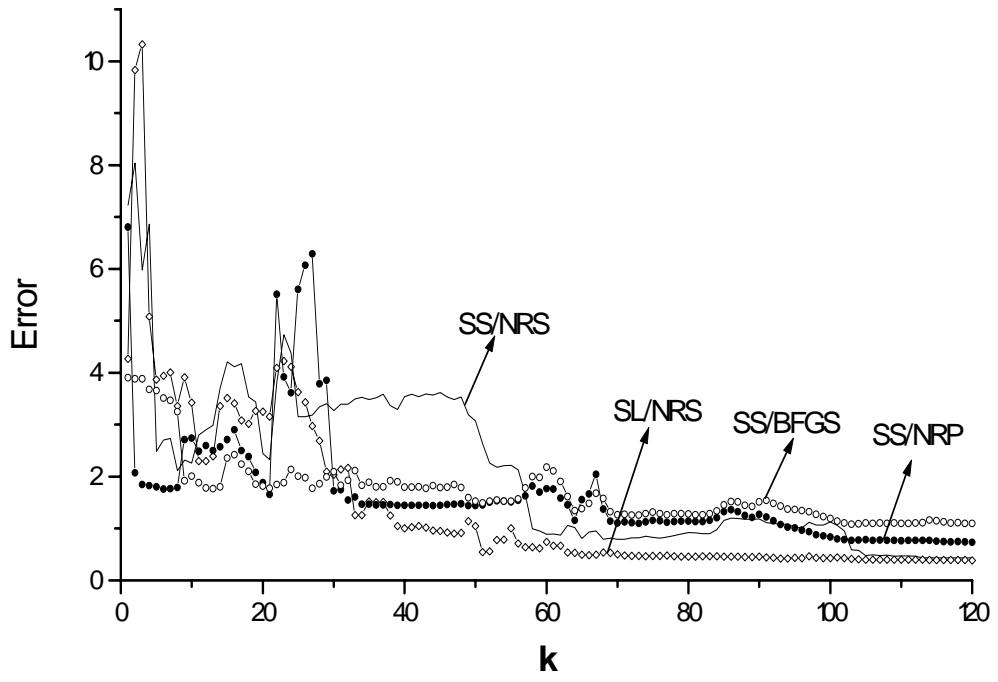
Figure 5 : RBF network structure.



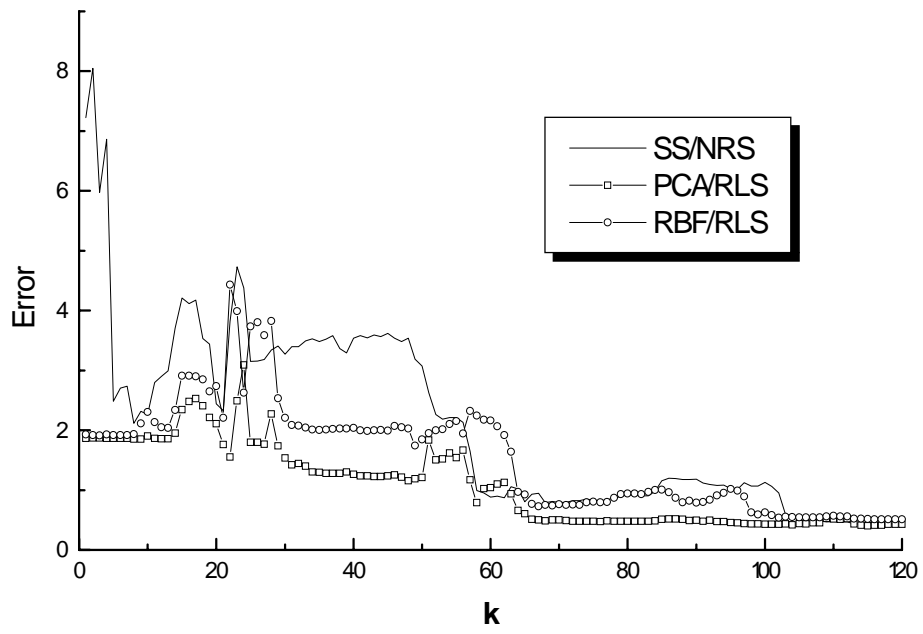
**Figure.6 :** Values of  $T_i$  (input var. of the process),  $R$  (output var. of the process) and  $Y$  (kinetic parameter to be estimated) obtained in a nonisothermal CSTR reactor with reaction  $A \rightleftharpoons R$ .



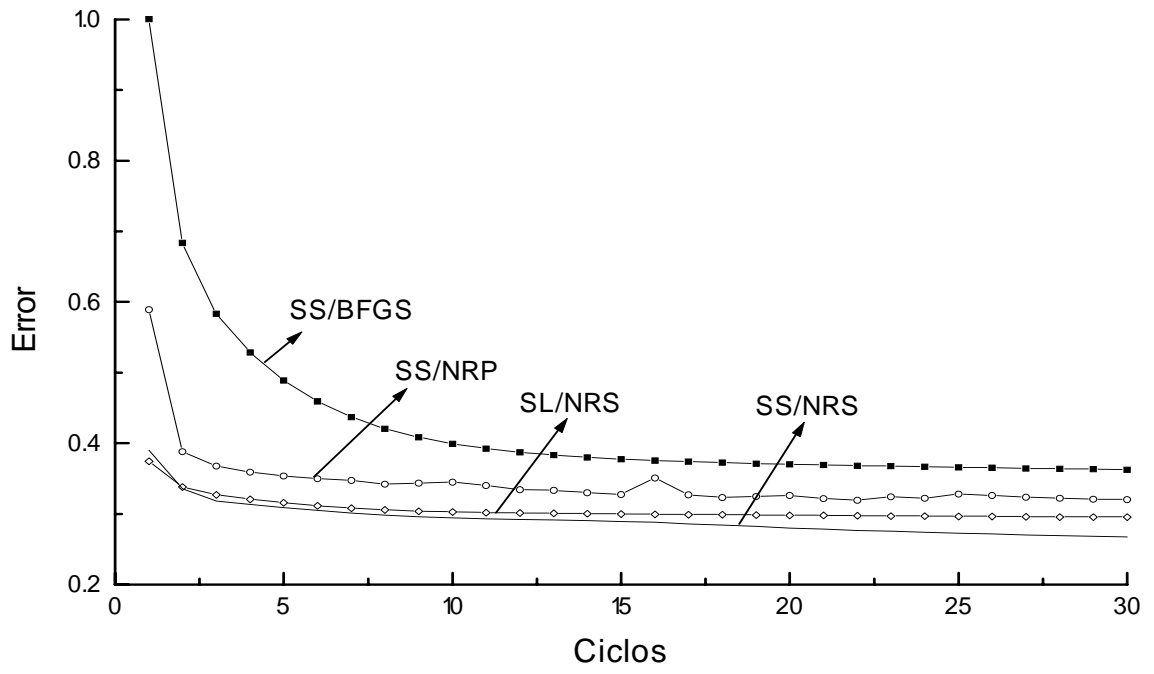
**Figure 7 : Distribution of centers (□) and data (●) after one adaptation cycle**



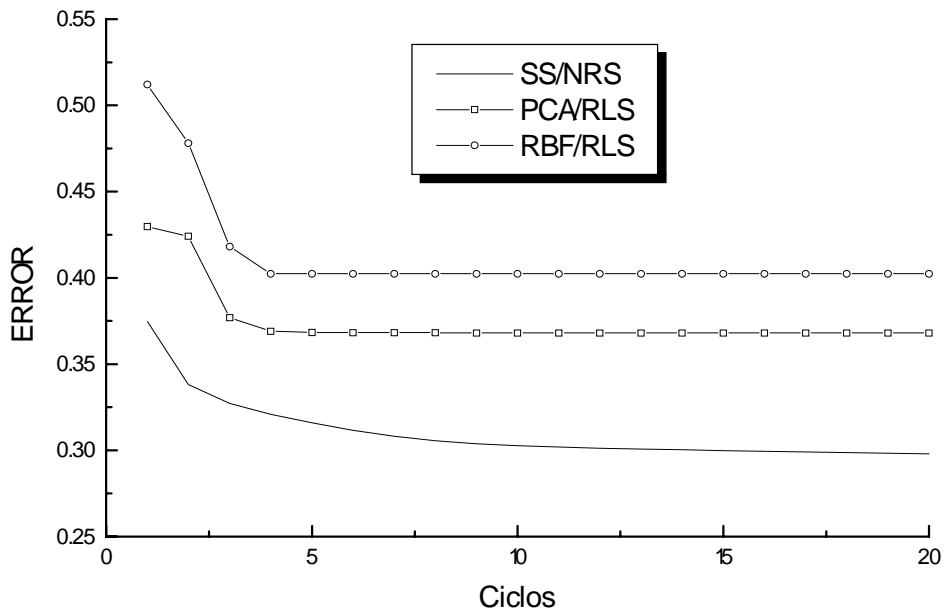
**Figure 8: Evolution of  $\hat{\epsilon}$  in the initial estimation cycle; *FFNN* networks.**



**Figure 9 : Evolution of  $\hat{\epsilon}$  in the initial estimation cycle; PCA and RBF networks.**



**Figure 10: Final estimation error for several estimation cycles, *FFNN* network.**



**Figure 11: Final estimation error for several estimation cycles, PCA and RBF networks.**

**Table 1 : Neural structures used**

<b>NETWORK</b>	<b>TYPE</b>	<b><math>f(v)</math></b>	<b>#Nodes c.o.</b>	<b>#Parameters</b>	<b>Algorithm</b>
<b>SS/NRS</b>	<b>FFNN</b>	Tanh/Tanh	4	21	<b>NRS</b>
<b>SS/NRP</b>	<b>FFNN</b>	Tanh/Tanh	4	21	<b>NRP</b>
<b>SS/BFGS</b>	<b>FFNN</b>	Tanh/Tanh	4	21	<b>BFGS</b>
<b>SL/NRS</b>	<b>FFNN</b>	Tanh/Lin.	4	20	<b>NRS</b>
<b>PCA</b>	<b>PCA</b>	Polyn. $P=3$	3	12	<b>GSO / RLS</b>
<b>RBF</b>	<b>RBF</b>	Spline n=2	12	12	<b>NMCR/RLS</b>