The Determination of Multivariable Nonlinear Models for Dynamic Systems Using Neural Networks

S A Billings* and S Chen**

* Department of Automatic Control and Systems Engineering
University of Sheffield, Mappin Street
Sheffield, S1 3JD, UK

** Department of Electrical and Electronic Engineering
University of Portsmouth, Anglesea Building
Portsmouth, PO1 3DJ, UK

Research Report No 629

June 1996

Invited Chapter in 'Neural Network Systems Techniques and Applications' in the Advances in Control and Dynamic Systems Series Edited by C T Leondes

The Determination of Multivariable Nonlinear Models for Dynamic Systems Using Neural Networks

S.A. Billings† and *S. Chen*‡

†Department of Automatic Control and Systems Engineering
University of Sheffield, Mappin Street
Sheffield S1 3JD, U.K.

‡Department of Electrical and Electronic Engineering
University of Portsmouth, Anglesea Building
Portsmouth PO1 3DJ, U.K.

1 Introduction

Modelling and identification is one of the major areas of control engineering, and the theory and practice of linear system identification is now well established [1],[2]. During the past decade, efforts have been focused on developing coherent and concise methods of nonlinear system modelling and identification [3]-[7], and more recently artificial neural networks have been applied to complex nonlinear dynamic systems [8]-[12]. There are two basic components in any system identification problem: determining the model structure and estimating or fitting the model parameters. These two tasks are critically influenced by the kind of model employed. Parameter estimation is relatively straightforward if the model structure is known a priori but this information is rarely available in practice and has to be learnt.

A general principle of system modelling is that the model should be no more complex than is required to capture the underlying system dynamics. This concept known as the *parsimonious* principle is particularly relevant in nonlinear model building because the size of a nonlinear model can easily become explosively large. An over complicated model may simply fit to the noise in the training data, resulting in *overfitting*. An overfitted model does not capture the underlying system structure well and will perform badly on new data. In neural network terminology, the model is said to have a poor generalisation property. The NARMAX (Nonlinear AutoRegressive Moving Average model with eXogenous inputs) methodology, developed by the authors and others, is based around the philosophy that determining the model structure is a critical part of the identification process. Many of the NARMAX techniques can readily be applied to neural network modelling.

During the resurgence in artificial neural network research in the 1980s, there was a tendency in neural network modelling to simply fit a large network model to data and to disregard structure information. Often models consisting of hundreds of parameters were used. It was thought that since neural networks had excellent representation capabilities and coded information into neuron connection weights, there was less need for physical insight into the underlying functional relationship and dependencies, and model structure determination became less critical. Soon people realized that a huge network model not only requires a very long training time and has poor generalisation performance but also has little value in system analysis and design. The parsimonious principle is therefore just as important in neural network modelling.

Many techniques have been developed to improve the generalisation properties of neural network models. These include: pruning to reduce the size of a complex network model [13]-[15], automatic construction algorithms to build parsimonious models for certain types of neural networks [16]-[19], regularisation [20]-[22], and combining regularisation with construction algorithms [18],[23]-[26]. Selection of the best and smallest nonlinear model from the set of all possible candidates is a *hard* optimisation problem which is generally intractable or computationally too expensive. Rather than insisting on the true optimal solution, a pragmatic approach is usually adopted in practice to find a suboptimal solution. In fact all of the successful construction algorithms attempt to find a sufficiently good suboptimal nonlinear model structure in this sense.

In a nonlinear model, the relationship between model outputs and model inputs is nonlinear by definition. But the relationship between the model outputs and the free adjustable model parameters can be either nonlinear or linear. Identification schemes can therefore be classified into two categories, nonlinear-in-the-parameters or linear-in-the-parameters. Neural networks are generally nonlinear-in-the-parameters. A typical example is the multilayer perceptron (MLP) model. Some basis function neural networks such as the radial basis function (RBF) network, B-spline networks and neural-fuzzy models can be configured in a linear-in-the-parameters structure.

The nonlinear-in-the-parameters neural network models are generally more compact and require less parameters compared with the linear-in-the-parameters models. In this sense the latter are often said to suffer from the "curse of dimensionality". But this problem arises largely because no attempt has been made to determine a parsimonious representation from the data. Learning for a nonlinear-in-the-parameters neural network is generally much more difficult than that for a linear-in-the-parameters neural network. More importantly, structure determination or the selection of a parsimonious model is extremely complex when the model is nonlinear-in-the-parameters and few results are available. Several powerful construction algorithms which automatically seek parsimonious models have been reported for basis function neural networks.

The remainder of this chapter is organized as follows. Section 2 introduces the NARMAX system representation which provides a framework for nonlinear system modelling and identification. Conventional or non-neural NARMAX modelling approaches are summarized in section 3, where the similarity between the conventional approach and neural network modelling is emphasized. Section 4 discusses a few popular neural network models employed in nonlinear system identification and classifies these models into the nonlinear-in-the-parameters and linear-in-the-parameters categories. Section 5 presents a gradient-based identification technique for neural network models that are nonlinear-in-the-parameters while section 6 discusses construction algorithms for building parsimonious neural network models that are linear-in-the-parameters. Issues of identifiability and the implications for nonlinear system modelling together with a scheme for constructing local models are treated in section 7. Some concluding remarks and further reserach topics are given in section 8. Throughout this chapter, simulated and real data are used to illustrate the concepts and techniques discussed.

2 The nonlinear system representation

For the class of discrete-time multivariable nonlinear dynamic systems, depicted in figure 1, the general input-output relationship can be written as

$$y(k) = f_s(y(k-1), \cdots, y(k-n_y), u(k-1), \cdots, u(k-n_u), e(k-1), \cdots, e(k-n_e)) + e(k) \quad (1)$$

where

$$y(k) = \begin{bmatrix} y_1(k) \\ \vdots \\ y_m(k) \end{bmatrix}, \quad u(k) = \begin{bmatrix} u_1(k) \\ \vdots \\ u_r(k) \end{bmatrix}, \quad e(k) = \begin{bmatrix} e_1(k) \\ \vdots \\ e_m(k) \end{bmatrix} \quad (2)$$

are the system m-dimensional output, r-dimensional input and m-dimensional noise vectors respectively; n_y, n_u and n_e are lags in the output, input and noise respectively; and $f_s(\cdot)$ is some m-dimensional vector-valued nonlinear function. The noise $e(k)$ is a zero mean independent sequence. This nonlinear system representation is known as the NARMAX model [3],[6]. Nonlinear time-series can be viewed as a specical case of this general nonlinear system and can be represented by the NARMA model [27]

$$y(k) = f_s(y(k-1), \cdots, y(k-n_y), e(k-1), \cdots, e(k-n_e)) + e(k) \quad (3)$$

The NARMAX model provides a general basis for the development of nonlinear system identification techniques.

The functional form $f_s(\cdot)$ for a real-world system is generally very complex and unknown. Any practical modelling must be based on a chosen model set of known functions. Obviously, this model set should be capable of approximating the underlying process to within an acceptable accuracy, and neural networks are just a class of such "functional approximators" [28]-[31]. Secondly, an efficient identification procedure must be developed for the selection of a parsimonious model structure because the dimension of a nonlinear model can easily become extremely large. Without efficient subset selection, the resulting model often has little practical value.

In general recurrent or feedback neural networks are required to model the general NARMAX system. In some approximations, it may be possible to reduce the NARMAX model to a simplified form

$$y(k) = f_s(y(k-1), \cdots, y(k-n_y), u(k-1), \cdots, u(k-n_u)) + e(k) \quad (4)$$

For autonomous systems or time series, the model (4) can be further reduced to

$$y(k) = f_s(y(k-1), \cdots, y(k-n_y)) + e(k) \quad (5)$$

The model in (4) is obviously less general than the model in (1). Identification schemes and system analysis are however simpler when based on the model (4), and many feedforward neural networks are sufficient to identify this simplified NARMAX model.

A main assumption for the NARMAX system is that the "system state space" has a finite dimension. This agrees with a basic result of dynamical systems theory [32], which states that if the attractor of a dynamical system is contained within a finite-dimensional manifold, then an embedding of the manifold can be constructed from time series observations of the dynamics on the manifold. The dynamical system induced by the embedding is differentiably equivalent to the one being observed. In fact, the lag n_y in (5) corresponds to the embedding vector dimension. Dynamical systems theory provides a firm basis for nonlinear time series modelling and many nonlinear signal processing applications, and it has strong connections with nonlinear system modelling and identification.

The choice of identification scheme is critically influenced by the chosen model form. Consider a general nonlinear model

$$\hat{y}(k) = f_m(\mathbf{x}(k); \Theta) \tag{6}$$

where Θ is the vector of adjustable model parameters. If the model is used to identify the NARMAX system (1), the model input vector $\mathbf{x}(k)$ is

$$\mathbf{x}(k) = [y^T(k-1) \cdots y^T(k-n_y)\ u^T(k-1) \cdots u^T(k-n_u)\ \epsilon^T(k-1) \cdots \epsilon^T(k-n_e)]^T \tag{7}$$

where

$$\epsilon(k) = y(k) - \hat{y}(k) \tag{8}$$

is the prediction error. If the model is used to identify the simplified NARMAX system (4),

$$\mathbf{x}(k) = [y^T(k-1) \cdots y^T(k-n_y)\ u^T(k-1) \cdots u^T(k-n_u)]^T \tag{9}$$

According to the relationship between the model output and adjustable model parameters, various models can be classified into two categories, namely nonlinear-in-the-parameters and linear-in-the-parameters models.

3 The conventional NARMAX methodology

The NARMAX modelling methodology, developed by the authors and others, is centred around the belief that the determination of model structure is a crucial part of the identification process. The NARMAX methodology attempts to break a system identification problem down into the following aspects:

Nonlinearity detection	determine whether a nonlinear model is needed
Structure determination	select which terms to include in the model
Parameter estimation	estimate the values of the model parameters
Model validation	test whether the model is adequate
Model application	use the model for prediction and system analysis

These components form an identification toolkit which allows the user to build a concise mathematical description of the system from data. A crucial component of this iteractive identification procedure is the determination of the model structure. This ensures that a parsimonious model is obtained.

Several conventional or non-neural nonlinear models have been developed for the NARMAX modelling approach. These typically include the polynomial model [7], the rational model [33] and the extended model set [34]. The polynomial model is derived by a polynomial expansion of $\mathbf{x}(k)$

$$\hat{y}_i(k) = p_i(\mathbf{x}(k); \Theta_i), \quad 1 \le i \le m \tag{10}$$

where $p_i(\cdot)$ is a polynomial function of $\mathbf{x}(k)$, and Θ_i denotes the parameter vector consisting of the coefficients to be identified. The rational model is given as the quotient of two polynomial expansions of $\mathbf{x}(k)$

$$\hat{y}_i(k) = \frac{p_{N_i}(\mathbf{x}(k); \Theta_{N_i})}{p_{D_i}(\mathbf{x}(k); \Theta_{D_i})}, \quad 1 \le i \le m \tag{11}$$

The extended model set refers to a general modelling approach which first forms some fixed nonlinear expansions to obtain various nonlinear model terms and then linearly combines the resulting terms. The same idea was given a neural network interpretation in [35] and the resulting model was called the functional-link network (FLN).

3.1 Structure determination and parameter estimation

When identifying nonlinear systems with unknown structure, it is important to avoid losing significant terms that must be included in the final model. Consequently, the experimenter is forced to start with a large model set. The key to the NARMAX modelling approach is an efficient procedure for subset model selection which aims to select only the significant model terms to form a parsimonious subset model. Which subset selection scheme to use will depend on the particular model form employed.

3.1.1 Nonlinear-in-the-parameters models

A class of parameter estimation algorithms widely used for nonlinear models is the prediction error algorithms [5],[33]. This is a class of gradient based algorithms that minimize the cost function

$$J_1(\Theta) = \log\left(\det\left(Q(\Theta)\right)\right) \tag{12}$$

where $\det(\cdot)$ denotes the determinant,

$$Q(\Theta) = \frac{1}{N}\sum_{k=1}^{N} \epsilon(k;\Theta)\epsilon^T(k;\Theta) \tag{13}$$

and N is the number of data samples. Here we have made the dependency of the prediction error on the model parameters explicit by using the notation $\epsilon(k;\Theta)$. Let the dimension of Θ be n_Θ, that is, $\Theta = [\theta_1 \cdots \theta_{n_\Theta}]^T$. The parameter estimate which minimizes the performance criterion $J_1(\Theta)$ can be obtained using the Gauss-Newton algorithm based on the gradient of $J_1(\Theta)$

$$\nabla J_1(\Theta) = \left[\frac{\partial J_1}{\partial \Theta}\right]^T = \left[\frac{\partial J_1}{\partial \theta_1} \cdots \frac{\partial J_1}{\partial \theta_{n_\Theta}}\right]^T \tag{14}$$

and the Hessian

$$H_1(\Theta) \approx \frac{\partial^2 J_1}{\partial \Theta^2} + \rho I = \frac{\partial}{\partial \Theta}\left[\frac{\partial J_1}{\partial \Theta}\right]^T + \rho I \tag{15}$$

where ρ is a small positive constant and I is the identity matrix of appropriate dimension. The addition of ρI to the Hessian matrix avoids the possibility of ill-conditioning. The gradient and the Hessian can be computed as follows [1]

$$\frac{\partial J_1}{\partial \theta_i} = \frac{2}{N}\sum_{k=1}^{N} \epsilon^T(k;\Theta)Q^{-1}(\Theta)\frac{\partial \epsilon(k;\Theta)}{\partial \theta_i}, \quad 1 \le i \le n_\Theta \tag{16}$$

$$\frac{\partial^2 J_1}{\partial \theta_i \partial \theta_j} \approx \frac{2}{N}\sum_{k=1}^{N} \frac{\partial \epsilon^T(k;\Theta)}{\partial \theta_i}Q^{-1}(\Theta)\frac{\partial \epsilon(k;\Theta)}{\partial \theta_j}, \quad 1 \le i,j \le n_\Theta \tag{17}$$

The Hessian $H_1(\Theta)$ also plays an important role in subset model selection. In fact, $H_1(\Theta)$ contains sufficient information regarding the significance of each parameter. Delecting a parameter corresponds to removing a row and a column in the Hessian matrix. This is because subset model selection can be formulated as an optimisation problem with the performance criterion [4]

$$C = N\log(\det(Q(\bar{\Theta}))) + n_{\bar{\Theta}}\chi_\alpha(1) \tag{18}$$

where $\bar{\Theta}$ is the parameter vector of the particular model and $n_{\bar{\Theta}}$ its dimension, $\chi_\alpha(1)$ is the critical value of the chi-squared distribution with 1 degree of freedom and a given significance level α. The best model selected from all the competing models is the one that minimizes the C-criterion (18). The first term in (18) indicates the model performance and the second term is a complexity measure that penalizes large models. An appropriate value for $\chi_\alpha(1)$ can be shown to be 4.0 [4]. The Hessian of the C-criterion (18), which is the same as $H_1(\Theta)$ except for a scalar N, is what is actually needed to efficiently evaluate the C-criterion value of a candidate model [4],[36].

For a full model size of n_Θ, the number of competing models is 2^{n_Θ}. The true optimal model that minimizes (18) must be selected from all these 2^{n_Θ} models. It is obvious that this is computationally impossible to do even for a moderate n_Θ. A practical method for selecting parsimonious models is based on backward elimination, which eliminates some parameters of a large model according to information provided by $H_1(\Theta)$. This method is similar to the stepwise backward elimination scheme in the statistical literature [37], and can generally find a suboptimal subset model. The details of a numerical algorithm, which implements this backward elimination algorithm using $H_1(\Theta)$ to compute the values of the C-criterion, can be found in [36]. We use the following simulated system taken from [33] to illustrate the combined approach of structure determination and parameter estimation.

Example 1.

The data was generated from the simulated single-input single-output system

$$y(k) = \frac{y(k-1) + u(k-1) + y(k-1)u(k-1) + y(k-1)e(k-1)}{1 + y^2(k-1) + u(k-1)e(k-1)} + e(k) \tag{19}$$

The input $u(k)$ was an independent and uniformly distributed sequence with mean zero and variance 1.0 and the noise $e(k)$ is a Gaussian white sequence with mean zero and variance 0.01. Notice that if the system is nonlinear the possibility of nonlinear noise terms has to be accommodated in the model. 600 points of the input-output data were used in the identification. A rational model with the lags $n_y = n_u = n_e = 1$ and a quadratic numerator and denominator were used. The full model had 20 terms shown in table 1.

The prediction error algorithm was first used to obtain the parameter estimate $\hat{\Theta}$ for the full model. The full model Hessian, evaluated at $\hat{\Theta}$, was then used in the backward elimination procedure to eliminate redundant parameters in the model. The elimination process is shown in table 2, where it is seen that the process stopped after stage 13 because the C-criterion reached a minimum value, giving rise to a subset model of 7 parameters. The prediction error algorithm was again used to fine tune the subset model parameters. The final subset model given in table 3 is obviously a good estimate.

The combined approach of structure determination and parameter estimation based on the prediction error estimation method, discussed above, is very general and can be applied to any nonlinear model. An alternative cost function for the prediction error estimation method is

$$J_2(\Theta) = \text{trace}(Q(\Theta)) = \frac{1}{N}\sum_{k=1}^{N} \epsilon^T(k;\Theta)\epsilon(k;\Theta) \qquad (20)$$

A similar algorithm which combines structure determination and parameter estimation can be derived based on this cost function. In the case of single-output systems, that is, $m = 1$, the two cost functions (12) and (20) become identical. A drawback of the approach discussed here is that, when the full model size n_Θ is very large, the computation of the full model Hessian can be expensive.

A multilayered neural network model is usually nonlinear-in-the-parameters. Therefore it is not surprising that the prediction error estimation method can readily be applied for constructing neural network models [9],[11],[38]. In fact, the famous backpropagation learning method [39] for neural networks can be viewed as a special case of the prediction error algorithm [38]. The importance of the parsimonious principle is now widely recognized in the neural network community. Weight elimination has been suggested as a method for reducing the size of large network models, and the process is known as the pruning. Approaches adopted in pruning often have their root in more traditional methods for subset model selection. For example, the so-called optimal brain damage method [13] uses the diagonal elements of the cost function Hessian in weight elimination.

3.1.2 Linear-in-the-parameters models

The polynomial model and the extended model set are examples of linear-in-the-parameters models. A linear-in-the-parameters model can generally be obtained by performing some *fixed* nonlinear functional transforms or expansions of the inputs before combining the resulting terms linearly. Specifically, a set of given functional expansions map the input space onto a new space of increased dimension n_L,

$$\mathbf{x}(k) \longrightarrow [\phi_1(\mathbf{x}(k)) \cdots \phi_{n_L}(\mathbf{x}(k))]^T \qquad (21)$$

The model outputs are obtained as linear combinations of the new *bases* $\phi_i(\mathbf{x}(k))$, $1 \le i \le n_L$,

$$\hat{y}_i(k) = \sum_{j=1}^{n_L} \theta_{j,i}\phi_j(\mathbf{x}(k)), \quad 1 \le i \le m \tag{22}$$

To qualify for a linear-in-the-parameters model, the value of each given basis function must depend only on the input $\mathbf{x}(k)$ so that $\phi_i(\mathbf{x}(k))$ contains no other adjustable parameters.

An advantage of the model (22) is that the standard least squares identification method can readily be applied to estimate the parameters $\theta_{j,i}$. In practice, the model dimension n_L can become excessively large. Consider, for example, the polynomial model, which uses the set of monomials of $\mathbf{x}(k)$ as bases. If the dimension of $\mathbf{x}(k)$ is 8, a degree-5 polynomial expansion will produce a model basis set of dimension $n_L = 1286$. Other choices of model bases [34] can also induce the problem of excessive model dimension. Subset selection is therefore essential, and an efficient subset selection procedure has been derived based on the orthogonal least squares (OLS) method [7]. Given the full set of n_L candidate bases, the OLS algorithm selects significant model bases one by one in a forward regression manner until an adequate subset model is constructed. The selection procedure is made simple and efficient by exploiting an orthogonal property.

Specifically, the cost function (20) is adopted for a combined subset model selection and parameter estimation. The system outputs, the model outputs and the prediction errors for $1 \le k \le N$ can be collected together in the matrix form

$$\mathbf{Y} = \mathbf{\Phi\Theta} + \mathbf{E} \tag{23}$$

where

$$\mathbf{Y} = [\mathbf{y}_1 \cdots \mathbf{y}_m] = \begin{bmatrix} y_1(1) & \cdots & y_m(1) \\ \vdots & \vdots & \vdots \\ y_1(N) & \cdots & y_m(N) \end{bmatrix} \tag{24}$$

$$\mathbf{E} = [\mathbf{e}_1 \cdots \mathbf{e}_m] = \begin{bmatrix} \epsilon_1(1) & \cdots & \epsilon_m(1) \\ \vdots & \vdots & \vdots \\ \epsilon_1(N) & \cdots & \epsilon_m(N) \end{bmatrix} \tag{25}$$

$$\mathbf{\Phi} = [\Phi_1 \cdots \Phi_{n_L}] = \begin{bmatrix} \phi_1(\mathbf{x}(1)) & \cdots & \phi_{n_L}(\mathbf{x}(1)) \\ \vdots & \vdots & \vdots \\ \phi_1(\mathbf{x}(N)) & \cdots & \phi_{n_L}(\mathbf{x}(N)) \end{bmatrix} \tag{26}$$

$$\mathbf{\Theta} = \begin{bmatrix} \theta_{1,1} & \cdots & \theta_{1,m} \\ \vdots & \vdots & \vdots \\ \theta_{n_L,1} & \cdots & \theta_{n_L,m} \end{bmatrix} \tag{27}$$

Let an orthogonal decomposition of the regression matrix $\mathbf{\Phi}$ be $\mathbf{\Phi} = \mathbf{WA}$. The system (23) can be rewritten as

$$\mathbf{Y} = \mathbf{WG} + \mathbf{E} \tag{28}$$

where

$$\mathbf{G} = \begin{bmatrix} g_{1,1} & \cdots & g_{1,m} \\ \vdots & \vdots & \vdots \\ g_{n_L,1} & \cdots & g_{n_L,m} \end{bmatrix} = \mathbf{A\Theta} \tag{29}$$

It can be shown that

$$NJ_2(\Theta) = \text{trace}\left(\mathbf{E}^T\mathbf{E}\right) = \text{trace}\left(\mathbf{Y}^T\mathbf{Y}\right) - \sum_{j=1}^{n_L}\left(\sum_{i=1}^{m} g_{j,i}^2\right)\mathbf{w}_j^T\mathbf{w}_j \tag{30}$$

where the new bases $\mathbf{w}_j$ are columns of $\mathbf{W}$. Define the error reduction ratio due to $\mathbf{w}_l$ as

$$[err]_l = \left(\sum_{i=1}^{m} g_{l,i}^2\right)\mathbf{w}_l^T\mathbf{w}_l/\text{trace}\left(\mathbf{Y}^T\mathbf{Y}\right) \tag{31}$$

Based on this ratio, significant model terms can be selected in a forward-selection procedure. At the l-th stage, a model term is selected among the $n_L - l + 1$ candidates if it produces the largest value of $[err]_l$ to add to the previously selected $(l-1)$-term model. The selection procedure can be terminated at the n_S stage when

$$1 - \sum_{l=1}^{n_S}[err]_l < \eta \tag{32}$$

where $0 < \eta < 1$ is a pre-set desired tolerance, giving rise to an n_S-term subset model. Alternatively, the procedure can be terminated when the criterion

$$N\log(J_2(\Theta)) + n_S\chi_\alpha(1) \tag{33}$$

reaches a minimum, where $\chi_\alpha(1)$ is as defined previously at (18).

The details of the OLS algorithm for forward subset selection can be found in [7]. It should be emphasized that the OLS algorithm is an efficient way of implementing forward subset selection and, like any forward subset selection method, it does not guarantee to find the best n_S-term model from the n_L-term full model. This however is not a serious deficiency because the subset model found is usually very good. Furthermore the optimal solution will certainly be too expensive to compute since n_L is often very large.

Notice that the aim here is to select a subset model consisting of a set of the *original* model bases, say, $\{\Phi_{j_l},\ 1 \le l \le n_S\}$. The set of the selected orthogonal bases $\{\mathbf{w}_l,\ 1 \le l \le n_S\}$ corresponds *precisely* to a subset of the original model bases $\{\Phi_{j_l},\ 1 \le l \le n_S\}$, that is, the subspace spanned by $\{\mathbf{w}_l,\ 1 \le l \le n_S\}$ is the same space spanned by $\{\Phi_{j_l},\ 1 \le l \le n_S\}$. This is reflected in the fact that $\mathbf{A}$ is an upper triangular matrix with unit diagonal elements. In many signal processing applications, the objective is to transform the original signal space $\mathbf{\Phi}$ onto a new space by some orthogonal transformation and to tackle the problem on a transformed subspace. This can be achieved, for example, using singular value decomposition (SVD) [40]. A subset of the orthonormal bases $\{\mathbf{v}_l,\ 1 \le l \le n_S\}$, which correspond to the first n_S largest eigenvalues, is selected to form the required subspace. Since each $\mathbf{v}_l$ is a linear combination of all the original bases Φ_j, $1 \le j \le n_L$, it is not known which subset $\{\Phi_{j_l},\ 1 \le l \le n_S\}$ represents exactly the subspace spanned by $\{\mathbf{v}_l,\ 1 \le l \le n_S\}$. When a subset model consisting of a subset of the original model bases is required, the OLS algorithm has clear advantages.

If the stopping criterion (32) is employed, the chosen tolerance η will influence both the modelling accuracy and the complexity of the final subset model. It is obvious that ideally η should be larger than but very close to the ratio trace$(\mathbf{E}^T\mathbf{E})$/trace$(\mathbf{Y}^T\mathbf{Y})$. Since trace$(\mathbf{E}^T\mathbf{E})$ is not known a priori, an appropriate value of η usually needs to be found, and this can be achieved by the following iterative learning procedure. An initial guess is assigned to η. Once a model is selected, an estimate for trace($\mathbf{E}^T\mathbf{E}$) can be computed and since trace($\mathbf{Y}^T\mathbf{Y}$) is known from the data, an improved η can be chosen. We use the following simulated system taken from [34] to illustrate this learning strategy and to demonstrate OLS subset selection.

Example 2.

The data was generated from the simulated single-input single-output system

$$y(k) = 0.5y(k-1)+u(k-2)+0.1u^2(k-1)+0.5e(k-1)+0.2u(k-1)e(k-2)+e(k) \quad (34)$$

where the system noise $e(k)$ was a Gaussian white noise with mean zero and variance $\sigma_e^2 = 0.04$ and the system input $u(k)$ was a uniformly distributed independent sequence with mean zero and variance 1.0. A data set of 500 input-output pairs was used in identification. A polynomial model with input

$$\mathbf{x}(k) = [y(k-1)\ y(k-2)\ y(k-3)\ u(k-1)\ u(k-2)\ u(k-3)\ \epsilon(k-1)\ \epsilon(k-2)\ \epsilon(k-3)]^T \quad (35)$$

and degree-3 polynomial expansion was used to fit the data. The full model set contained $n_L = 220$ terms. The iterative procedure for the subset model selection using the OLS algorithm is summarized in table 4.

Initially, we did not have the residual sequence and could not use the residual variance σ_ϵ^2 as an estimate of the noise variance σ_e^2. We also could not form any model terms containing $\epsilon(k-j)$. Thus, at the initial iteration, the model input was assumed to be

$$\mathbf{x}(k) = [y(k-1)\ y(k-2)\ y(k-3)\ u(k-1)\ u(k-2)\ u(k-3)]^T \quad (36)$$

An initial guess was given to the desired tolarence as $\eta = 0.032$, and the OLS algorithm selected a subset model with an estimate σ_ϵ^2. This initial iteration generated a residual sequence and allowed the assignment of a new tolarence $\eta = 0.03$. The iterative procedure was completed after the 5th iteration since the subset models selected at the 4th and 5th iterations were identical, and the variances of the resulting residual sequences were hardly changing.

For the class of linear-in-the-parameters models, automatic construction algorithms that are capable of building parsimonious models from data are crucial for overcoming the curse of dimensionality. The OLS algorithm described here is one such algorithm. On-line versions of these procedures are now available to provide adaptive model selection and estimation for nonlinear systems [41],[42]. Certain neural network learning problems can also be formulated within the linear-in-the-parameters framework. An example is the RBF network. When the centres and widths of a RBF network are fixed, learning becomes a linear-in-the-parameter problem [43]. The OLS algorithm can readily be applied to construct parsimonious RBF networks [16].

3.2 Model validation

Model validation is an important step in any identification or modelling procedure. For linear system identification, if the model structure and parameter values are correct, the residual $\epsilon(k)$ will be uncorrelated with past inputs and outputs. In the case of single-input single-output ($r = m = 1$) systems, therefore, an identified linear model is regarded as adequate if the autocorrelation function of $\epsilon(k)$ and the cross-correlation function of $\epsilon(k)$ and $u(k)$ satisfy

$$\left.\begin{array}{ll} \mathcal{R}_{\epsilon\epsilon}(\tau) = 0, & \tau \neq 0 \\ \mathcal{R}_{\epsilon u}(\tau) = 0, & \text{for all } \tau \end{array}\right\} \qquad (37)$$

For validating single-input single-output nonlinear models, (37) is clearly insufficient. The known principle of nonlinear model validation is to test if the residuals are unpredictable from all the linear and nonlinear combinations of past inputs and outputs. Three higher-order correlation tests have been suggested in addition to the tests (37) to validate whether a nonlinear model is correct [44],[33].

These correlation-based tests can also be applied to validate multi-input multi-output nonlinear models but a large number of correlation plots will be required. A more compact validation procedure has been developed for multivariable nonlinear models using the following tests [45]

$$\left.\begin{array}{ll} \mathcal{R}_{\beta\beta}(\tau) = 0, & \tau \neq 0 \\ \mathcal{R}_{\nu\beta}(\tau) = 0, & \text{for all } \tau \\ \mathcal{R}_{\gamma\omega}(\tau) = 0, & \tau \neq 0 \\ \mathcal{R}_{\zeta\omega}(\tau) = 0, & \text{for all } \tau \end{array}\right\} \qquad (38)$$

where

$$\left.\begin{array}{l} \beta(k) = \epsilon_1(k) + \epsilon_2(k) + \cdots + \epsilon_m(k) \\ \nu(k) = u_1(k) + u_2(k) + \cdots + u_r(k) \\ \gamma(k) = \epsilon_1^2(k) + \epsilon_2^2(k) + \cdots + \epsilon_m^2(k) \\ \omega(k) = y_1(k)\epsilon_1(k) + y_2(k)\epsilon_2(k) + \cdots + y_m(k)\epsilon_m(k) \\ \zeta(k) = u_1^2(k) + u_2^2(k) + \cdots + u_r^2(k) \end{array}\right\} \qquad (39)$$

Alternatively, model validation can be performed based on chi-squared tests [46],[4] but the correlation tests discussed here are much easier to implement.

In practice, the correlation tests are computed in normalized form and confidence limits are used to interpret if the tests have been violated or not. Experience has shown that if these tests are used in conjunction with a combined approach of structure determination and parameter estimation, they often give the experimenter a great deal of information regarding deficiencies in the fitted model and help to indicate which terms should be included in the model to improve the fit. Some examples of real-data identification coupled with the model validity tests using various models including neural networks can be found in [9],[33],[47], [48].

4 Neural network models

Generally, artificial neural networks refer to a computational paradigm in which a large number of simple computational units known as "neurons" or simply nodes, interconnected to form a network, perform complex computational tasks. Such a computational model was inspired by neurobiological systems. A key feature of neural networks is learning, and this refers to the fact that a neural network is trained to perform a task using examples of data. Such a learning process has clear connections with system identification and parameter estimation. A variety of neural network architectures or models have been employed for constructing representations of complex nonlinear systems from data and several of these models will be discussed below. From the viewpoint of information flow, there are two classes of neural networks, namely feedforward and recurrent networks. In the former case, the input signal is propagated forward through the network to produce the network output. In the latter case, node outputs can be fed back as node inputs. From the viewpoint of parameter estimation, neural networks can be classified as nonlinear-in-the-parameters or linear-in-the-parameters.

4.1 Multilayer perceptrons

A feedforward multilayer perceptron, MLP, is a layered network made up of one or more *hidden* layers between the input and output layers. Each layer consists of computing nodes, and the nodes in a layer are connected to the nodes in adjacent layers but there is no connection between the nodes within the same layer and no bridging layer connections. The input layer acts as an input data holder which distributes the inputs to the first hidden layer. The output from the first layer nodes then become inputs to the second layer and so on. The last layer acts as the network output layer. The architecture of an MLP can conveniently be summarized as $n_0 - n_1 - \cdots - n_l$, where n_0 is the number of the network inputs, n_l is the number of the network outputs and n_i, $1 \leq i \leq l-1$, are the numbers of nodes in the respective hidden layers. Figure 2(a) depicts the topology of a $4-5-6-3$ MLP.

The input-output relationship of a generic node is shown in figure 2(b). The node computes the weighted sum of the node inputs x_i, $1 \leq i \leq n$, adds a bias weight and passes the result through a nonlinear activation function f_a to produce the node output y_j

$$y_j = f_a\left(\sum_{i=0}^{n} w_{j,i}x_i\right) \tag{40}$$

where $w_{j,i}$ are the node weights and $x_0 = 1$ indicates that $w_{j,0}$ is a bias term. A typical choice of the activation function is the sigmoid function defined as

$$f_a(v) = \frac{1}{1+\exp(-v)} \tag{41}$$

For the purpose of function approximation, the output nodes usually do not contain a bias term and the associated activation functions are linear.

An MLP realizes an overall input-output mapping: $f_{MLP} : R^{n_0} \longrightarrow R^{n_l}$. The MLP is a general function approximator and theoretical works [28],[29] have shown that MLPs with one hidden layer are sufficient to approximate any continuous functions provided that there are enough hidden nodes. In applications to identifying nonlinear dynamic systems, the number of the output nodes is equal to the number of the system outputs and the number of the network input is equal to the dimension of $\mathbf{x}(k)$. Collect all the wieghts of the network model into a vector form Θ. The network input-output mapping

$f_{MLP}(\mathbf{x}(k);\Theta)$ is obviously highly nonlinear in Θ and training an MLP is equivalent to estimating this parameter vector. After learning, the network mapping can be used as a model of the system dynamics f_s.

4.2 Radial basis function networks

The RBF network is a processing structure consisting of an input layer, a hidden layer and an output layer. The hidden layer of a RBF network consists of an array of nodes and each node contains a parameter vector called a centre. The node calculates the Euclidean distance between the centre and the network input vector, and passes the result through a radially symmetric nonlinear function. The output layer is essentially a set of linear combiners. An example of the RBF network and the model of the Gaussian RBF node are shown in figure 3. The overall input-output response of an n_I-input n_O-output RBF network is a mapping $f_{RBF} : R^{n_I} \longrightarrow R^{n_O}$. Specifically,

$$f_{RBF_i} = \sum_{j=1}^{n_H} \theta_{j,i}\phi_j = \sum_{j=1}^{n_H} \theta_{j,i}\phi(\|\mathbf{x} - \mathbf{c}_j\|; \rho_j), \quad 1 \le i \le n_O \tag{42}$$

where $\theta_{j,i}$ are the weights of the linear combiners, $\|\cdot\|$ denotes the Euclidean norm, ρ_j are some positive scalars called the widths, $\mathbf{c}_j$ are the RBF centres, $\phi(\cdot;\rho)$ is a function from $R^+ \longrightarrow R$, and n_H is the number of hidden nodes.

The RBF method is a traditional technique for strict interpolation in multidimensional space [49]. On the other hand, using a feedforward neural network to model complex data can also be considered as performing a curve fitting operation in a multidimensional space. Broomhead and Lowe [43] adopted this viewpoint and revealed explicitly the connection between feedforward neural networks and RBF models. The topology of the RBF network is similar to that of the one-hidden-layer perceptron, and the difference lies in the characteristics of the hidden nodes. Two typical choices of $\phi(\cdot)$ for RBF networks are the Gaussian function

$$\phi(v/\rho) = \exp(-v^2/\rho) \tag{43}$$

and the thin-plate-spline function

$$\phi(v) = v^2 \log(v) \tag{44}$$

The Gaussian function represents a class of node nonlinearity having the property that $\phi(v/\rho) \longrightarrow 0$ as $v \longrightarrow \infty$. The thin-plate-spline function represents another class of node nonlinearity having the property that $\phi(v/\rho) \longrightarrow \infty$ as $v \longrightarrow \infty$. These two very different classes of nonlinearities may be referred to as class one and class two respectively.

Most of the applications of RBF networks can be included in the following framework: use the network input-output mapping f_{RBF} to learn or to approximate some nonlinear mapping from $R^{n_I} \longrightarrow R^{n_O}$. The RBF network is known to be a general function approximator and theoretical investigations have concluded that the choice of $\phi(\cdot)$ is not crucial for network approximation capabilities [50],[51],[30]. RBF models based on either class-one or class-two nonlinearities all have excellent approximation capabilities. Nevertherless, choosing the nonlinearity of the hidden nodes according to the application can often improve performance. Athough each hidden node may have a different width ρ_j, a uniform width is sufficient for universal approximation [30]. All the widths in the network can therefore be fixed to a value ρ and this ρ can be derived using some heuristic rules [52]. For classification applications, however, adjusting individual widths can often improve the generalisation properties [53]. Some choices of the nonlinearity such as (44) do not involve a width.

An extension to the standard RBF network is to replace the Euclidean distance by the Mahalanobis distance in hidden nodes [53],[54]. In this case, the hidden nodes become

$$\phi_j = \phi\left((\mathbf{x}-\mathbf{c}_j)^T V_j^{-1}(\mathbf{x}-\mathbf{c}_j)\right), \quad 1 \le j \le n_H \tag{45}$$

where V_j is an $n_I \times n_I$ positive definite matrix. Hartman and Keeler [55] proposed a Gaussian-bar network in which each input dimension is treated differently. In the multi-output case, the hidden nodes are defined by

$$\phi_{j,i} = \sum_{l=1}^{n_I} \theta_{l,j,i} \exp\left(-(x_l - c_{l,j})^2/\rho_{l,j}\right), \quad 1 \le j \le n_H, \quad 1 \le i \le n_O \tag{46}$$

The ith output node is the sum of $\phi_{j,i}$ for $1 \le j \le n_H$. Hartman and Keeler [55] reported better performance of this Gaussian-bar network over the standard Gaussian network.

In general, the RBF network is nonlinear in the parameters. However, learning can be performed in two phases. Firstly, some learning mechanism is employed to select a suitable set of RBF centres and widths. This effectively determines the hidden layer of the RBF network. Since the output layer is a set of linear combiners, learning the remaining output-layer weights becomes a linear problem. In this sense, the RBF network is often said to have a "linear-in-the-parameters" structure. This is only true after the hidden layer has been fixed separately. Because of this property, learning procedures for the RBF network are simple and reliable.

4.3 Fuzzy basis function networks

The architecture of a general n_I-input n_O-output fuzzy system is depicted in figure 4. A fuzzy system consists of four basic elements: a fuzzifier, a fuzzy rule base, a fuzzy inference engine and a defuzzifier. The fuzzifier maps the crisp input space onto the fuzzy sets defined in the input space. The fuzzy rule base consists of a set of n_F linguistic rules in the forms of IF–THEN. The fuzzy inference engine is a decision making logic which employs fuzzy rules from the rule base to determine a mapping from the fuzzy sets in the input space onto the fuzzy sets in the output space. The defuzzifier performs a mapping from the fuzzy sets in R^{n_O} to the crisp outputs $\mathbf{y} \in R^{n_O}$. A class of fuzzy systems commonly used in practice are constructed based on singleton fuzzification, product inference and centroid or weighted average defuzzification. Such fuzzy systems can be represented as series expansions of fuzzy basis functions known as fuzzy basis function (FBF) networks or models [31],[56].

An n_I-input n_O-output fuzzy system can be expressed as n_O single-output fuzzy subsystems. The rule base of the ith fuzzy subsystem consists of n_F rules in the following forms

$$RB_j^i : \text{ IF } x_1 \text{ is } A_{1,j} \text{ AND } \cdots \text{ AND } x_{n_I} \text{ is } A_{n_I,j} \text{ THEN } y_i \text{ is } B_j^i \tag{47}$$

where $1 \le j \le n_F$, x_l for $1 \le l \le n_I$ are the inputs to the fuzzy system, y_i is the ith output of the fuzzy system, $1 \le i \le n_O$, and $A_{l,j}$ and B_j^i are the fuzzy sets characterised by fuzzy membership functions $\mu_{A_{l,j}}(x_l)$ and $\mu_{B_j^i}(y_i)$, respectively. Under the assumptions of singleton fuzzifier, product inference and centroid defuzzifier, the input-output mapping of such a fuzzy system, $f_{FBF}: R^{n_I} \longrightarrow R^{n_O}$, can be shown to have the form [31],[56],[57]

$$f_{FBF_i}(\mathbf{x}) = \frac{\sum_{j=1}^{n_F} \bar{y}_j^i \left(\prod_{l=1}^{n_I} \mu_{A_{l,j}}(x_l)\right)}{\sum_{j=1}^{n_F} \left(\prod_{l=1}^{n_I} \mu_{A_{l,j}}(x_l)\right)}, \quad 1 \le i \le n_O \tag{48}$$

where $\bar{y}_j^i$ is the point at which $\mu_{B_j^i}(y_i)$ achieves its maximum value. Define

$$\phi_j(\mathbf{x}) = \frac{\prod_{l=1}^{n_I} \mu_{A_{l,j}}(x_l)}{\sum_{j=1}^{n_F} \left(\prod_{l=1}^{n_I} \mu_{A_{l,j}}(x_l)\right)}, \quad 1 \le j \le n_F \tag{49}$$

which are referred to as FBFs [31],[56]. Then the fuzzy system (48) is equivalent to an FBF expansion

$$f_{FBF_i}(\mathbf{x}) = \sum_{j=1}^{n_F} \phi_j(\mathbf{x})\theta_{j,i}, \quad 1 \le i \le n_O \tag{50}$$

where $\theta_{j,i}$ are coefficients or parameters of the FBF model.

Fuzzy systems are universal approximators and theoretical studies have proved that the FBF network (50) can approximate any continuous functions to within any degree of accuracy provided that a sufficient number of fuzzy rules are used [31],[56],[57]. Although the FBF network is derived within the framework of fuzzy logic, it obviously has many similarities to neural networks such as the RBF network. An advantage of the FBF network is that linguistic information from human experts in the form of the fuzzy IF–THEN rules can be directly incorporated into the model.

Currently fuzzy systems are mainly applied to engineering problems where the number of inputs and outputs is small. This is because the number of rules or FBFs grows exponentially as the number of inputs and outputs increases. This curse of "rule explosion" limits further applications of fuzzy systems to complex large systems. Research has been directed towards determining the best shape for fuzzy sets which will result in a significant reduction in the number of rules required [58]. Notice that the FBF network (50) has a linear-in-the-parameters structure once the rules have been specificied. Given observed input-output data, the problem of constructing a parsimonious FBF model can be formulated as one of subset model selection. This is the approach adopted in [31], where the OLS algorithm was used to select significant FBFs from a large set of candidate FBFs.

4.4 Recurrent neural networks

Architectures of recurrent neural networks are extremely rich and it is impossible to give a comprehensive discussion in a short section. For recent advances in the theory and applications of recurrent neural networks, see for example [59]. A straightforward way of obtaining a recurrent neural network is to feed back the network output to the network input. The structure of such feedback or recurrent networks is shown in figure 5. Notice that the feedback signal can consist of either the delayed network outputs $\hat{y}(k-1), \cdots, \hat{y}(k-n_d)$

or the delayed prediction errors $\epsilon(k-1), \cdots, \epsilon(k-n_d)$, where $\epsilon(k) = y(k) - \hat{y}(k)$ and $y(k)$ is the desired output. The "feedforward network" part of this structure can be a multilayer perceptron, a RBF network or a FBF network. Although this structure may appear to be similar to the feedforward network except for an "expanded" network input, the design and analysis of such a feedback network is considerably more complex than for feedforward networks. Advantages gained by using this structure are richer dynamic behaviours and improved representation capabilities. An example where this approach works well is in adaptive noise cancellation [60].

In a general recurrent network, node outputs can be fed back to node inputs. A class of 3-layer recurrent networks with internal feedback is illustrated in figure 6(a). As in the case of feedforward networks, the input layer simply distributes the network inputs to the hidden-layer nodes and the output layer is a set of linear combiners. The outputs of the hidden nodes are however fed back as part of the inputs to the hidden nodes, and the models of the two kinds of connections for hidden-layer neurons are depicted in figure 6(b). The topology of this 3-layer recurrent network can be summarised as $n_0 - n_1 - n_2$, where n_0 is the number of network inputs, n_1 the number of hidden nodes and n_2 the number of network outputs. The output of the jth hidden node is computed as

$$x_{O_j}(k) = f_a\left(\sum_{l=0}^{n_0} w_{F_{j,l}} x_{I_l}(k) + \sum_{l=1}^{n_1} w_{B_{j,l}} x_{O_l}(k-1)\right), \quad 1 \le j \le n_1 \tag{51}$$

where $x_{I_l}(k)$ for $1 \le l \le n_0$ are the network inputs, $x_{I_0}(k) = 1$ indicates that $w_{F_{j,0}}$ is a bias weight, $x_{O_j}(k)$ are the hidden layer outputs, $w_{F_{j,l}}$ and $w_{B_{j,l}}$ are the feedforward and feedback connection weights respectively. The single-hidden-layer feedforward network can be viewed as a special case of this recurrent network with all the feedback connection weights being zeros.

More generally, a recurrent network can include more than one hidden layer, and the network output can also be fed back as part of the network input. A smaller recurrent network with fewer hidden nodes can often achieve the same modelling accuracy as a large feedforward network. In this sense, recurrent networks have better representation capabilities. The learning and analysis for recurrent networks is however much more difficult. A recurrent network is a highly nonlinear-in-the-parameters and complex dynamic system. Theoretical investigation and practical application of recurrent networks will provide an interesting and challenging research area for many years to come.

5 Nonlinear-in-the-parameters approach

For artificial neural networks that are nonlinear in the parameters, learning must be based on nonlinear optimisation techniques. The optimisation criterion is typically chosen to be the mean square error defined in (20), and gradient learning methods remain as the most popular techniques for learning despite the possible pitfalls that learning can become trapped at a local minimum of the optimisation criterion. Global optimisation techniques, such as genetic algorithms [61] and simulated annealing [62], although capable of achieving a global minimum, require extensive computation. We describe an efficient gradient learning algorithm called the parallel prediction error algorithm (PPEA) [38].

5.1 Parallel prediction error algorithm

The prediction error algorithm [33] discussed previously in section 3 is a general parameter estimation algorithm and can readily be applied to neural networks [9]. Compared with the popular backpropagation algorithm (BPA) [39] which is a steepest-descent gradient algorithm, the prediction error algorithm achieves significantly better performance at a much faster convergence rate since the Hessian information is utilized in the search. The BPA however has computational advantages. Consider a generic neural network with a total of p neurons, and let the number of parameters of each node be n_{Θ_j}, where $1 \leq j \leq p$. The total number of parameters in the network is therefore

$$n_\Theta = \sum_{j=1}^{p} n_{\Theta_j} \tag{52}$$

The computational complexity of the BPA is of the order of n_Θ while the complexity of the prediction error algorithm can be shown to be of the order of $(n_\Theta)^2$. Another important advantage of the BPA is that the algorithm is a distributed learning procedure and weight updating is carried out locally. This is coherent with the massively distributed computing nature and parallel structure of neural networks.

The PPEA is also a distributed learning procedure that is a trade-off between the high performance of the full prediction error algorithm and the low complexity of the BPA. The recursive version of the PPEA is described here. For the generic network of p neurons and n_O outputs, local learning is achieved using

$$\left.\begin{aligned}
&\epsilon(k) = y(k) - \hat{y}(k) \\
&\Psi_j(k) = \left[\tfrac{\partial \hat{y}(k)}{\partial \Theta_j}\right]^T \\
&P_j(k) = \tfrac{1}{\lambda}\left[P_j(k-1) - P_j(k-1)\Psi_j(k)\left(\lambda I + \Psi_j^T(k)P_j(k-1)\Psi_j(k)\right)^{-1}\Psi_j^T(k)P_j(k-1)\right] \\
&\Theta_j(k) = \Theta_j(k-1) + P_j(k)\Psi_j(k)\epsilon(k)
\end{aligned}\right\} \tag{53}$$

where $1 \leq j \leq p$, $\hat{y}(k)$ is the network output vector computed given the previous parameter estimate, Θ_j is the parameter vector of the jth node, I is the $n_O \times n_O$ identity matrix, and λ is the forgetting factor. Essentially, the PPEA consists of many sub-algorithms, each one associated with a neuron in the network. The BPA can be derived from this PPEA by replacing all the P_j matrices with appropriate diagonal matrices [38].

By utilizing "local" or approximate Hessian information contained in the $P_j(k)$ matrices, the PPEA offers significant performance improvements over the BPA. Computational complexity of the PPEA is of the order of $\bar{n}_\Theta$ where

$$\bar{n}_\Theta = \sum_{j=1}^{p} \left(n_{\Theta_j}\right)^2 \tag{54}$$

Although this is more complex than the BPA, the increase in computational requirements per iteration can often be offset by a faster convergence rate so that overall the PPEA is computationally more efficient. We demonstrate better performance and faster convergence of the PPEA over the BPA by considering a real system identification application taken from [11].

Example 3.

The data was generated from a single-input single-output nonlinear liquid level system. The system consists of a DC pump feeding a conical flask which in turn feeds a square tank. The system input $u(k)$, the voltage to the pump motor, and the system output $y(k)$, the water level in the conical flask, are plotted in figure 7. A one-hidden-layer feedforward network with 5 hidden neurons was employed to model this real process and the network input vector was chosen to be

$$\mathbf{x}(k) = [y(k-1)\ y(k-2)\ y(k-3)\ u(k-1) \cdots u(k-5)]^T \tag{55}$$

The network structure was thus specified as 8-5-1, giving rise to $n_\Theta = 50$. The hidden node activation function was the sigmoid function (41) and initial weights were set randomly between -0.3 to 0.3. For the PPEA, all the P_j matrices were initialized to $100.0I$ with I being identity matrices of appropriate dimensions. A variable forgetting factor

$$\lambda(k) = \lambda_0 \lambda(k-1) + (1.0 - \lambda_0) \tag{56}$$

was used with $\lambda(0) = 0.95$ and $\lambda_0 = 0.99$. For the BPA, an adaptive gain of 0.01 and a momentum coefficient of 0.9 were found to be appropriate. The evolutions of the mean square error in dB's obtained by the BPA and the PPEA are depicted in figure 8.

5.2 Pruning oversized network models

Choosing a proper network size is critical for dynamic system modelling using neural networks. If the network is too small, it will not be able to capture the underlying structure of the dynamic system. On the other hand, if the network is too big, it may simply fit to the noise in the training data, resulting in a poor generalisation capability. According to the parsimonious principle, the smallest network that fits the data should be chosen in order to obtain good generalisation performance. If no prior knowledge is available, a proper network size has to be determined by trial and error. In the training of feedforward networks, one can start with a small network and gradually increase the size by adding more hidden and/or input nodes until the performance stops improving. Using a separate validation data set to test the model performance can often help to determine an appropriate network size.

Even when such a systematic approach is adopted, the final fully connected network model may still contain a large number of "redundant" weights which can be eliminated without affecting the network modelling capability. Prunning refers to the process of deleting redundant or insignificant weights in an oversized network model and is an effective technique to improve generalisation properties. The algorithm based on the C-criterion (18), discussed in section 3, is a powerful tool to prune oversized networks. The algorithm belongs to the approach utilizing complexity regularisation since (18) contains a measure of model complexity. A drawback of this algorithm is its computational cost. When the size of the full network is very large, the computation of the Hessian matrix is expensive and the stepwise elimination procedure further adds a considerable computational burden. A popular and computationally much simpler method for pruning is known as the "optimal brain damage" (OBD) method [13].

In the OBD method, pruning is based on the saliency measure of network weights. Let J be the cost function used for network learning and θ_i be a generic network weight. The saliency of θ_i defined by the OBD method is

$$S(\theta_i) \approx \frac{\partial^2 J}{\partial \theta_i^2}\,\theta_i^2 \tag{57}$$

Notice that $\frac{\partial^2 J}{\partial \theta_i^2}$ is a diagonal element of the Hessian matrix. The pruning process of the OBD method is as follows. The network is first trained and the saliencies of all the weights are computed using (57). The weights with the smallest saliency are then deleted and the reduced-size network is retrained. The procedure may need to be repeated several times before a final pruned network model is obtained.

If the standard BPA is used for training, the second derivations $\frac{\partial^2 J}{\partial \theta_i^2}$ need to be calculated separately in order to compute the saliencies of the weights. Notice however that if the batch version of the PPEA [38] is employed as the training algorithm, the diagonal elements of the (approximated) Hessian matrix are readily available and the computation of saliency (57) requires little extra cost. The detailed batch PPEA can be found in [11],[38] and will not be repeated here. We use the following example to demonstrate the combined approach of the batch PPEA training and pruning based on the saliency measure.

Example 4.

This is the same liquid level system which was studied in Example 3. The same 8-5-1 feedforward network was employed and all the 1000 data points were ued as the training data. This fully connected 8-5-1 network had a total of 50 weights or parameters. The batch PPEA [38] was used to train the network model with the performance criterion

$$J(\Theta) = \frac{1}{2}\log\left(Q(\Theta)\right), \quad Q(\Theta) = \frac{1}{N}\sum_{k=1}^{N} \epsilon^2(k;\Theta) \tag{58}$$

After the training, the mean square error $Q(\Theta)$ of the fully connected network model over the training data set was -26.74 dB. Table 5 summarizes the weight values and the corresponding saliencies of this fully connected network model.

By inspecting the saliency values given in table 5, it was concluded that any saliency smaller than 1.0 could obviously be regarded as insignificant and the corresponding weight could be deleted. The saliency threshold was therefore set to 1.0 and this resulted in 11 weights (underlined in table 5) being eliminated. After retraining, the mean square error of the pruned network model over the training data set was -26.67 dB. To compare the generalisation performance of the two network models, we examined the iterative network model output defined as

$$\hat{y}_d(k) = f_{MLP}(\mathbf{x}_d(k)) \tag{59}$$

where

$$\mathbf{x}_d(k) = [\hat{y}_d(k-1)\ \hat{y}_d(k-2)\ \hat{y}_d(k-3)\ u(k-1)\cdots u(k-5)]^T \tag{60}$$

Notice that during learning, the network input $\mathbf{x}(k)$ contained past system inputs and outputs. In generating $\hat{y}_d(k)$, the network was purely driven by the system inputs. If the trained model f_{MLP} really captured the system structure, $\hat{y}_d(k)$ should be able to follow the system output $y(k)$ closely. The iterative network outputs of the fully connected and pruned network models are plotted in figure 9. The results shown in figure 9 clearly demonstrate that the pruned smaller network has superior generalisation performance over the fully connected network model.

6 Linear-in-the-parameters approach

Neural networks are never truly linear-in-the-parameters. The class of basis-function neural networks, such as the RBF and FBF networks, only become linear-in-the-parameters after a separate mechanism has been used to fix or to select the basis functions. For this class of neural networks, learning can be "decomposed" into two phases. In the first phase, the hidden layer which is a set of basis functions is determined. Often this is done using some simple and efficient *unsupervised* method. The second phase of learning is a pure linear learning problem and can easily be solved. The linear-in-the-parameters approach for neural network learning really refers to this two-stage approach. The main advantage of this approach is that it avoids complicated nonlinear gradient-based learning. We will use the RBF network as an example to illustrate this learning approach. The techniques discussed can be applied to other basis-function networks such as the FBF network.

For the RBF network, there are two basic methods to avoid learning based on complex nonlinear optimisation. The first method chooses the RBF centres from the training inputs. Learning is formulated as a problem of linear subset model selection, and the OLS algorithm discussed in section 3 provides an efficient tool for constructing parsimonious RBF networks [16]. Other variants on this theme include the use of mutual information [63] and genetic algorithms [64] to configure the RBF netowrks. In the second method, RBF centres are first obtained using the unsupervised κ-means clustering algorithm and linear RBF weights are then learnt using the standard least squares or least mean square algorithm [52],[65]. If a width parameter is required, it is estimated using some heuristic rules [52]. We discuss these two methods here with an emphasis on recent improvements.

6.1 Regularised orthogonal least squares learning

Assume that we have a training set of N samples $\{y(k), \mathbf{x}(k)\}_{k=1}^{N}$. If each input vector $\mathbf{x}(k)$ is used as a RBF centre, we have a total of $n_H = N$ basis functions. The desired outputs $y(k)$, the RBF network model outputs and the errors can be collected together in the matrix form (23). The OLS algorithm can readily be applied to construct a parsimonious subset network model from this full model [16]. A well constructed parsimonious model often has better generalisation properties and suffers less from overfitting in a noisy environment. A technique to improve generalisation of over-parameterised neural networks is regularisation [21],[22]. For practical purposes, it is highly advantageous to combine regularisation techniques with parsimonious construction algorithms, and a regularised OLS (ROLS) algorithm has recently been derived [24],[25].

The usual least squares criterion in certain circumstances is prone to overfitting. When the data are highly noisy and the model size is large, the problem can be serious. The regularisation method improves generalisation by adding a penalty function to the cost function. Thus, the cost function $NJ_2(\Theta) = \text{trace}\left(\mathbf{E}^T\mathbf{E}\right)$ of (30) is modified into

$$J_R(\Theta;\xi) = \text{trace}\left(\mathbf{E}^T\mathbf{E}\right) + \xi(\text{penalty function}) \tag{61}$$

where ξ is a regularisation parameter. The simplest penalty function, known as the zero-order regularisation, is trace $\left(\Theta^T\Theta\right)$. The zero-order regularisation is a technique equivalent to the weight-decaying in gradient descent methods for the MLP [66]. It is also known as the ridge regression in the statistical literature [67].

In the derivation of the ROLS algorithm, the following zero-order regularised error criterion is actually used

$$J_R(\mathbf{G};\xi) = \text{trace}\left(\mathbf{E}^T\mathbf{E} + \xi\mathbf{G}^T\mathbf{G}\right) \tag{62}$$

where $\mathbf{G}$ is the orthogonal weight matrix related to Θ by (29). Notice that the use of trace $\left(\mathbf{G}^T\mathbf{G}\right)$ as the penalty is equivalent to the use of trace $\left(\Theta^T\Theta\right)$. The choice of the regularised error criterion (62) however has considerable computational advantages. This is because this cost function can be decomposed into

$$\text{trace}\left(\mathbf{E}^T\mathbf{E} + \xi\mathbf{G}^T\mathbf{G}\right) = \text{trace}\left(\mathbf{Y}^T\mathbf{Y}\right) - \sum_{j=1}^{n_H}\left(\sum_{i=1}^{m} g_{j,i}^2\right)\left(\mathbf{w}_j^T\mathbf{w}_j + \xi\right) \tag{63}$$

where $\mathbf{w}_j$ are the orthogonal basis vectors. Similar to the case of the OLS algorithm, we can define the regularised error reduction ratio due to $\mathbf{w}_l$ as

$$[rerr]_l = \left(\sum_{i=1}^{m} g_{l,i}^2\right)\left(\mathbf{w}_l^T\mathbf{w}_l + \xi\right)/\text{trace}\left(\mathbf{Y}^T\mathbf{Y}\right) \tag{64}$$

Based on this ratio, significant basis functions can be selected in a forward-selection procedure exactly as in the case of the OLS algorithm [24],[25].

The appropriate value of the regularisation parameter ξ depends on the underlying system that generates the training data and the choice of basis function $\phi(\cdot)$. How to choose a good value of ξ has been addressed in the statistical literature [67],[68]. The optimal value of ξ can be determined based on the Bayesian evidence procedure [23]. Applying this Bayesian approach to the ROLS algorithm results in the following iterative procedure for estimating ξ. Given an initial guess of ξ, the algorithm constructs a network model. This in turn allows an updating of ξ by the formula

$$\xi = \frac{\gamma}{N-\gamma}\,\frac{\text{trace}\left(\mathbf{E}^T\mathbf{E}\right)}{\text{trace}\left(\mathbf{G}^T\mathbf{G}\right)}, \tag{65}$$

where

$$\gamma = \sum_{i=1}^{n_S} \frac{\mathbf{w}_i^T\mathbf{w}_i}{\mathbf{w}_i^T\mathbf{w}_i + \xi} \tag{66}$$

is known as the number of good parameter measurements [23], and n_S is the number of basis functions in the selected network model. After a few iterations, an appropriate ξ value can be found. We use two examples taken from [25] to demonstrate the effectiveness of the ROLS algorithm.

Example 5.

This simple example was designed to illustrate the problem of overfitting and the power of the regularisation technique. In this example, the RBF network with Gaussian basis function and a width $\rho = 0.2$ was used to approximate the scalar function

$$f(x) = \sin(2\pi x),\ 0 \leq x \leq 1 \tag{67}$$

One hundred tràining data were generated from $f(x) + e$, where x was uniformly distributed in $(0,\ 1)$ and the noise e had a Gaussian distribution with zero mean and standard deviation 0.4. A separated test data set was also generated for $x = 0, 0.01, \cdots, 0.99, 1.00$. The training data and the function $f(x)$ are plotted in figure 10. The training data set was very noisy and highly ill-conditioned.

The ROLS algorithm selected 15 centres from the training set. Figure 11 depicts the mean square error as a function of $\log_{10}(\xi)$ for the both training and testing data sets. The optimal value of ξ for this example was approximately 1.0. However, for a large range of ξ values, the mean square error over the testing set was quite flat, indicating that the performance of the ROLS algorithm was fairly insensitive to the precise value of ξ in this large region. When the evidence formula (65) was used to estimate ξ, ξ converged approximately to 1.0. Figure 12 shows the network mapping constructed by the ROLS algorithm with $\lambda = 1.0$. As a comparison, the network mapping constructed by the OLS algorithm is given in figure 13, where overfitting can be clearly seen. This example demonstrates that subset selection alone is not immune to overfitting when data is highly noisy, and a combined regularisation and subset selection approach is often desirable.

Example 6.

This was the time series of annual sunspot numbers. The sunspot time series over the years 1700–1979 is depicted in figure 14. The data from 1700 to 1920 were used for training, and the multi-step preditions were then computed over the years 1921-1955 and the years 1921–1979 respectively. In a previous study [69], the OLS algorithm was used to construct a RBF network predictor with thin-plate-spline basis function (44). The algorithm selected 25 centres, and the predictive accuracy of the resulting RBF model is shown in figure 15. It should be emphasized that the performance of this RBF network constructed using the OLS algorithm is better than some other nonlinear models fitted to the time series [27],[70].

To demonstrate that regularisation can further improve performance, the ROLS algorithm was used to construct a RBF network of 25 centres based on the same full network model with $\xi = 10^7$. This value of regularisation parameter was found using the evidence procedure. The predictive accuracy of the network model obtained using the ROLS algorithm is also shown in figure 15, where it is clearly seen that the ROLS algorithm has better generalisation properties.

6.2 Enhanced clustering and least squares learning

A popular learning method for RBF networks is the clustering and least squares learning [52],[65]. The RBF centres are obtained by means of a κ-means clustering algorithm while the network weights are learnt using the least squares algorithm. The κ-means clustering algorithm is an unsupervised learning method based only on input training samples. It partitions the input data set into n_H clusters and obtains the cluster centres by attempting to minimize the total squared error incurred in representing the data set by the n_H cluster centres [71]. The traditional κ-means clustering algorithm can only achieve a local optimal solution, which depends on the initial locations of cluster centres. A consequence of this local optimality is that some initial centres can get stuck in regions of the input domain with few or no input patterns, and never move to where they are needed. This wastes resources and results in an unnecessarily large network.

An improved κ-means clustering algorithm was recently proposed [72], which overcomes the above-mentioned drawback. By using a cluster variation-weighted measure, the enhanced κ-means partitioning process always achieves an optimal centre configuration in the sense that after convergence all clusters have an equal cluster variance. This property ensures that centre resources are not wasted and performance does not depend on the initial centre locations. This enhanced κ-means clustering algorithm can readily be combined with the least squares algorithm to provide a powerful learning method for constructing RBF networks [73]. We describe this combined learning method here.

The RBF network structure considered is the normalized Gaussian RBF network

$$f_{RBF_i}(\mathbf{x}(k)) = \sum_{j=1}^{n_H} \theta_{j,i}\phi_j(k), \quad 1 \le i \le n_O \tag{68}$$

where

$$\phi_j(k) = \frac{\exp(-\|\mathbf{x}(k) - \mathbf{c}_j\|^2/\sigma_j^2)}{\sum_{l=1}^{n_H} \exp(-\|\mathbf{x}(k) - \mathbf{c}_l\|^2/\sigma_l^2)} \tag{69}$$

A normalized Gaussian basis function features either localized behaviour similar to that of a Gaussian function or nonlocalized behaviour similar to that of a sigmoid function, depending on the location of the centre [74]. This is often a desired property.

The RBF centres are learnt using the following enhanced κ-means clustering algorithm

$$\mathbf{c}_j(k+1) = \mathbf{c}_j(k) + M_j(\mathbf{x}(k))\left(\eta_c(\mathbf{x}(k) - \mathbf{c}_j(k))\right) \tag{70}$$

where $0 < \eta_c < 1.0$ is a learning rate, the membership function $M_j(\mathbf{x}(k))$ is defined as

$$M_j(\mathbf{x}) = \begin{cases} 1, & \text{if } v_j\|\mathbf{x} - \mathbf{c}_j\|^2 \le v_l\|\mathbf{x} - \mathbf{c}_l\|^2 \text{ for all } l \ne j \\ 0, & \text{otherwise} \end{cases} \tag{71}$$

and v_j is the variation or "variance" of the jth cluster. To estimate variation v_j, the following updating rule is used

$$v_j(k+1) = \alpha v_j(k) + (1-\alpha)\left(M_j(\mathbf{x}(k))\|\mathbf{x}(k) - \mathbf{c}_j(k)\|^2\right) \tag{72}$$

The initial variations, $v_j(0)$, $1 \le j \le n_H$, are set to the same small number, and α is a constant slightly less than 1.0.

The learning rate η_c can either be fixed to a small constant or self-adjusting based on an "entropy" formula [72]

$$\eta_c = 1 - H(\bar{v}_1, \cdots, \bar{v}_{n_H})/\log(n_H) \tag{73}$$

where

$$H(\bar{v}_1, \cdots, \bar{v}_{n_H}) = \sum_{j=1}^{n_H} -\bar{v}_j \log(\bar{v}_j) \quad \text{with} \quad \bar{v}_j = v_j / \sum_{l=1}^{n_H} v_l \tag{74}$$

The widths, σ_j^2, $1 \leq j \leq n_H$, can be calculated, after the clustering process has converged, from the variances of the clusters. Since the optimal κ-means clustering distributes the total variation equally among the clusters, a universal width can be used for all the nodes. The network weights, $\theta_{j,i}$, are then learnt using the usual least squares algorithm.

Example 7.

This was a simulated two-dimensional autonomous system taken from [73]. The data were generated using

$$\begin{aligned} y(k) &= \Big(0.8 - 0.5\exp(-y^2(k-1))\Big)\, y(k-1) - \Big(0.3 + 0.9\exp(-y^2(k-1))\Big)\, y(k-2) \\ &\quad + 0.1\sin(\pi y(k-1)) + e(k) \end{aligned} \tag{75}$$

where the Gaussian white noise $e(k)$ had a zero mean and variance 0.01. 2000 samples of the time series are depicted in figure 16. The first 1000 points were used as the training set and the last 1000 points as the test set. The enhanced clustering and recursive least squares algorithm was used to construct a RBF network model. Figure 17 shows the mean square error as a function of the centre number.

The results of figure 17 indicated that 8 centres were sufficient to model this time series. The noise-free system is a limit cycle depicted in figure 18, where the 8 centre locations obtained by the enhanced κ-means clustering algorithm from noisy data are also plotted. From figure 18, it can be seen that an optimal centre configuration was obtained. When the network model output was fed back to the input, the iterative network model output

$$\hat{y}_d(k) = f_{RBF}(\hat{y}_d(k-1), \hat{y}_d(k-2)) \tag{76}$$

generated a limit cycle which was indistinguishable from the system limit cycle. The errors between the noise-free system outputs and the iterative network model outputs are given in figure 19. The results obtained using the enhanced clustering and least squares algorithm are better than some previous results [65],[75]. Furthermore, the present method requires a smaller network size.

There are several alternatives to the κ-means based clustering algorithms including the mean-tracking algorithm [76] and fuzzy clustering schemes [77]. All of these provide excellent results and avoid the disadvantages of the basic κ-means method.

6.3 Adaptive on-line learning

Recursive versions of the OLS algorithm have been derived which update both the model structures and the parameters of nonlinear models on-line. These have been applied to both NARMAX model identification and the training of RBF networks [41],[42],[78],[79]. The new algorithms operate on a window of data and employ a Givens routine to minimise the loss function at every selection step by selecting significant regression variables and computing the parameter estimates in a way which maintains the orthogonality of the vector space. These algorithms can be used as part of an efficient adaptive procedure to track the variations of both model structure [41],[79] or network topology [42],[79] and update the associated parameters or weights on-line.

7 Identifiability and local model fitting

Experiment design for linear system identification is well established [1]. Basically, the input signal chosen for a linear identification experiments should be persistently exciting to ensure identifiability. Persistent excitation in the context of linear system identification means that the input should excite all the frequencies of interest in the system, and this can be shown to relate to the second-order statistics of the input signal. For nonlinear system identification, however, the second-order statistics of the input are no longer sufficient to determine identifiability and in general the input probability density function or higher-order statistics are needed. Thus the design of inputs for nonlinear system identification is a very complex problem.

Some useful results have been given in [80]. Roughly speaking, the definition of persistent excitation in the context of nonlinear system identification should be modified as: an input signal is persistently exciting if the input excites all the frequencies of interest in the system and also excites the system over the whole amplitude range of operation. We use a simple example taken from [81] to illustrate the relationship between persistent excitation and identifiability.

Example 8.

A nonlinear digital communication channel can be represented by

$$y(k) = f_s(u(k), \cdots, u(k-n_u)) + e(k) \tag{77}$$

where the input $u(k)$ is a white sequence taking values from the set $\{\pm 1\}$, and the noise $e(k)$ is uncorrelated with $u(k)$. Since $u(k)$ is white, it contains all frequency components, and is an ideal input signal for identifying the linear model of any order n_u

$$\hat{y}(k) = \sum_{i=0}^{n_u} \theta_i u(k-i) \tag{78}$$

For nonlinear identification, the input should also excite a sufficient range of amplitudes. The binary nature of $u(k)$ therefore represents a worst scenario and, as a consequence, parameters in some nonlinear models may not be identifiable. For example, consider the following channel model

$$\hat{y}(k) = \sum_{i=0}^{2} \theta_i u(k-i) + \sum_{i=0}^{2}\sum_{j=i}^{2} \theta_{i,j} u(k-i)u(k-j) + \sum_{i=0}^{2}\sum_{j=i}^{2}\sum_{l=j}^{2} \theta_{i,j,l} u(k-i)u(k-j)u(k-l) \tag{79}$$

The rank of the 19×19 autocorrelation matrix of the estimator input vector is only 8. It is therefore impossible to identify all the 19 parameters in (79).

The requirement of exciting a sufficient range of amplitudes is difficult to meet in practice. Normal operation of an industrial plant is often concerned with controlling the plant close to some operating points. Perturbing signals that the experimenter injects into the plant can only have a small amplitude in order not to cause large disturbances to the operation of the plant. Such a small perturbing data set is bad for nonlinear system identification. If normal operation of the plant includes several operating levels, several sets of small perturbing data records can be obtained without violating the amplitude constraints for normal operation. These data records together may cover a sufficient range of amplitudes. This is illustrated in figure 20. A "global-model" fitting procedure [34] can then be applied to obtain a nonlinear model that is valid in the whole operating region of interest.

Using a single nonlinear model to represent a nonlinear system obviously has many advantages. But this depends on whether the system under investigation can be represented by a single "global" model. Typically, many nonlinear systems require more than one model to capture different dynamic behaviour over different operating regions. Fitting several local models [82] to a system may be particularly useful for modelling such systems, and this can be included in the framework of the threshold NARMAX (TNARMAX) model [34]. A general TNARMAX model is given by

$$\hat{y}(k) = f^{(l)}(\mathbf{x}(k); \Theta_l), \quad \text{if } \mathbf{x}(k) \in R^{(l)}, \quad 1 \le l \le M_L \tag{80}$$

where $R^{(l)}$ are given regions of the model input space $\mathbf{X}$, $f^{(l)}(\cdot)$ are some nonlinear mappings on $R^{(l)}$, and Θ_l are the parameter vectors of local models $f^{(l)}(\cdot)$ respectively.

A key step in TNARMAX modelling is a proper partition of the model input space $\mathbf{X}$ into M_L regions. A simple and effective way is to use the enhanced κ-means clustering algorithm. The data set is divided into M_L clusters as illustrated in figure 21. Each data cluster is then modelled. Various models and identification schemes such as those discussed previously can be employed in this local model fitting. In order to provide a smooth transition from one local model to another, regularisation techniques should be adopted in local model identification, and the second-order regularisation or curvature-driven smoothing schemes [21] are particularly useful. But care needs to be exercised if local models are to provide a representative model of a nonlinear system [82].

8 Conclusions

An artificial neural network which has the ability to learn sophisticated nonlinear relationships provides an ideal means of modelling complex nonlinear systems. In this chapter we have summarised several important results relating to nonlinear dynamic system modelling using neural networks and highlighted the connections between the traditional system identification approach and the neural network modelling approach. Neural network research has matured considerably during the past decade and two areas, constructing parsimonious networks and improving generalisation properties, have attracted extensive interest. The results of this research are particularly relevant to dynamic system modelling, since what really counts is how well a model captures the underlying system dynamics not how good it fits to the (noisy) observed data.

It is the authors belief that model structure determination will remain a key future research topic. A crucial issue is finding relevant model inputs. Often inputs to a neural network model

$$\mathbf{x} = [x_1 \; x_2 \cdots x_{n_I}]^T \tag{81}$$

are over complicated in the sense that only a subset of these inputs

$$\mathbf{x}_S = [x_{l_1} \; x_{l_2} \cdots x_{l_{n_S}}]^T \tag{82}$$

are significant and should only be considered as inputs. Development of efficient and reliable methods to detect relevant model inputs is of fundamental importance because a small reduction in the network input dimension can descrease the network complexity considerably. A useful scheme for determining important inputs is presented in [83]. A second important issue is that many systems are naturally composed of subsystems and ideally neural network models should be able to reflect this structural information. For example, a nonlinear system of inputs x_1, x_2, x_3, x_4 may simply consist of two additive subsystems

$$f(x_1, x_2, x_3, x_4) = f_1(x_1, x_4) + f_2(x_2, x_3) \tag{83}$$

If this structure information can be captured, a significant reduction in the network complexity can be achieved. Some neural network models may be inherently better in dealing with this kind of system identification problem than others, and previous research [17],[55],[19] provides a useful starting point for further investigation.

In view of past research results, we can confidently say that both the theory and practice of nonlinear system identification has advanced considerably during the past decade. Future research will involve multi-disciplinary approaches, including traditional control engineering, nonlinear dynamical systems theory and neural networks.

References

[1] G.C. Goodwin and R.L. Payne, *Dynamic System Identification: Experiment Design and Data Analysis.* New York: Academic Press, 1977.

[2] L. Ljung and T. Söderström, *Theory and Practice of Recursive Identification.* Cambridge, MA.: MIT Press, 1983.

[3] I.J. Leontaritis and S.A. Billings, "Input-output parametric models for non-linear systems – Part I: deterministic non-linear systems; Part II: stochastic non-linear systems," *Int. J. Control*, Vol.41, No.2, pp.303–344, 1985.

[4] I.J. Leontaritis and S.A. Billings, "Model selection and validation methods for non-linear systems," *Int. J. Control*, Vol.45, No.1, pp.311–341, 1987.

[5] S. Chen and S.A. Billings, "Recursive prediction error estimator for non-linear models," *Int. J. Control*, Vol.49, No.2, pp.569–594, 1989.

[6] S. Chen and S.A. Billings, "Representation of non-linear systems: the NARMAX model," *Int. J. Control*, Vol.49, No.3, pp.1013–1032, 1989.

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

[8] K.S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, Vol.1, pp.4–27, 1990.

[9] S. Chen, S.A. Billings and P.M. Grant, "Non-linear systems identification using neural networks," *Int. J. Control*, Vol.51, No.6, pp.1191–1214, 1990.

[10] K.J. Hunt, D. Sbarbaro, R. Zbikowski and P.J. Gawthrop, "Neural networks for control systems – a survey," *Automatica*, Vol.28, No.6, pp.1083–1112, 1992.

[11] S. Chen and S.A. Billings, "Neural networks for non-linear dynamic system modelling and identification," *Int. J. Control*, Vol.56, No.2, pp.319–346, 1992.

[12] M. Brown and C.J. Harris, *Neurofuzzy Adaptive Modelling and Control.* Hemel-Hempstead, UK: Prentice Hall, 1994.

[13] Y.le Cun, J.S. Denker and S.A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, Vol.2, D. Touretzky, Ed., 1990, pp.598–605.

[14] H. Thodberg, "Improving generalization on neural networks through pruning," *Int. J. Neural Syst.*, Vol.1, pp.317–326, 1991.

[15] R. Reed, "Pruning algorithms – a survey," *IEEE Trans. Neural Networks*, Vol.4, No.5, pp.740–747, 1993.

[16] S. Chen, C.F.N. Cowan and P.M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. Neural Networks*, Vol.2, No.2, pp.302–309, 1991.

[17] T.D. Sanger, "A tree-structured adaptive network for function approximation in high-dimensional space," *IEEE Trans. Neural Networks*, Vol.2, No.2, pp.285–293, 1991.

[18] J.H. Friedman, "Multivariate adaptive regression splines," (with discussion), *Annals of Statistics*, Vol.19, pp.1–141, 1991.

[19] T. Kavli, "ASMOD – an algorithm for adaptive spline modelling of observation data," *Int. J. Control*, Vol.58, No.4, pp.947–967, 1993.

[20] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE*, Vol.78, No.9, pp.1481–1497, 1990.

[21] C. Bishop, "Improving the generalization properties of radial basis function neural networks," *Neural Computation*, Vol.3, No.4, pp.579–588, 1991.

[22] F. Girosi, M. Jones and T. Poggio, "Regularisation theory and neural networks architectures," *Neural Computation*, Vol.7, pp.219–269, 1995.

[23] D.J.C. MacKay, "Bayesian interpolation," *Neural Computation*, Vol.4, No.3, pp.415–447, 1992.

[24] S. Chen, "Regularised OLS algorithm with fast implementation for training multi-output radial basis function networks," in *Proc. IEE 4th Int. Conf. Artificial Neural Networks* (Cambridge), June 26–28, 1995, pp.290–294.

[25] S. Chen, E.S. Chng and K. Alkadhimi, "Regularised orthogonal least squares algorithm for constructing radial basis function networks," to appear, *Int. J. Control*, 1996.

[26] M.J.L. Orr, "Regularisation in the selection of radial basis function centres," *Neural Computation*, Vol.7, No.3, pp.606–623, 1995.

[27] S. Chen and S.A. Billings, "Modelling and analysis of non-linear time series," *Int. J. Control*, Vol.50, No.6, pp.2151–2171, 1989.

[28] G. Cybenko, "Approximations by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, Vol.2, pp.303–314, 1989.

[29] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, Vol.2, pp.183–192, 1989.

[30] J. Park and I.W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, Vol.3, pp.246-257, 1991.

[31] L.X. Wang and J.M. Mendel, "Fuzzy basis functions, universal approximation, and orthogonal least-squares learning," *IEEE Trans. Neural Networks*, Vol.3, No.5, pp.807–814, 1992.

[32] D.S. Broomhead and G.P. King, "Extracting qualitative dynamics from experimental data," *Phsica D*, Vol.20, pp.217–236, 1986.

[33] S.A. Billings and S. Chen, "Identification of non-linear rational systems using a prediction-error estimation algorithm," *Int. J. Systems Sci.*, Vol.20, No.3, pp.467–494, 1989.

[34] S.A. Billings and S. Chen, "Extended model set, global data and threshold model identification of severely non-linear systems," *Int. J. Control*, Vol.50, No.5, pp.1897–1923, 1989.

[35] Yoh-Han Pao, *Adaptive Pattern Recognition and Neual Networks.* Reading, MA.: Addison-Wesley, 1989.

[36] S. Chen and S.A. Billings, "Prediction-error estimation algorithm for non-linear output-affine systems," *Int. J. Control*, Vol.47, No.1, pp.309–332, 1988.

[37] N.R. Draper and H. Smith, *Applied Regression Analysis.* New York: Wiley, 1981.

[38] S. Chen, C.F.N. Cowan, S.A. Billings and P.M. Grant, "Parallel recursive prediction error algorithm for training layered neural networks," *Int. J. Control*, Vol.51, No.6, pp.1215–1228, 1990.

[39] D.E. Rumelhart and J.L. McClelland, Eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Cambridge, MA.: MIT Press, 1986.

[40] G.H. Golub and C.F. Van Loan, *Matrix Computations.* Baltimore, MD.: Johns Hopkins University Press, 1989.

[41] W. Luo and S.A. Billings, "Adaptive model selection and estimation for nonlinear systems using a sliding data window," *Signal Processing*, Vol.46, pp.179–202, 1995.

[42] W. Luo and S.A. Billings, "Structure selective updating for nonlinear models and radial basis function neural networks," submitted to *Int. J. Adaptive Control and Signal Processing*, 1996.

[43] D.S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, Vol.2, pp.321–355, 1988.

[44] S.A. Billings and W.S.F. Voon, "Correlation based model validity tests for non-linear models," *Int. J. Control*, Vol.44, pp.235–244, 1986.

[45] S.A. Billings and Q.M. Zhu, "Model validation tests for multivariable nonlinear models including neural networks," *Int. J. Control*, Vol.62, No.4, pp.749–766, 1995.

[46] T. Bohlin, "Maximum power validation of models without higher-order fitting," *Automatica*, Vol.14, pp.137–146, 1978.

[47] S.A. Billings, S. Chen and R.J. Backhouse, "The identification of linear and non-linear models of a turbocharged automotive diesel engine," *Mechanical Systems and Signal Processing*, Vol.3, No.2, pp.123–142, 1989.

[48] S. Chen, S.A. Billings, C.F.N. Cowan and P.M. Grant, "Non-linear systems identification using radial basis functions," *Int. J. Systems Sci.*, Vol.21, No.12, pp.2513–2539, 1990.

[49] M.J.D. Powell, "Radial basis functions for multivariable interpolation: a review," in *Algorithms for Approximation*, J.C. Mason and M.G. Cox, eds., Oxford, 1987, pp.143–167.

[50] M.J.D. Powell, "Radial basis function approximations to polynomials," in *Proc. 12th Biennial Numerical Analysis Conf.*, Dundee, 1987, pp.223–241.

[51] E.J. Hartman, J.D. Keeler and J.M. Kowalski, "Layered neural networks with Gaussian hidden units as universal approximations," *Neural Computation*, Vol.2, No.2, pp.210–215, 1990.

[52] J. Moody and C.J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, Vol.1, pp.281–294, 1989.

[53] M.T. Musavi, W. Ahmed, K.H. Chan, K.B. Faris and D.M. Hummels, "On the training of radial basis function classifiers," *Neural Networks*, Vol.5, No.4, pp.595–603, 1992.

[54] S. Lee and R.M. Kil, "A Gaussian potential function network with hierarchically self-organizing learning," *Neural Networks*, Vol.4, pp.207–224, 1991.

[55] E.J. Hartman and J.D. Keeler, "Predicting the future: advantages of semilocal units," *Neural Computation*, Vol.3, No.4, pp.566–578, 1991.

[56] L.X. Wang, *Adaptive Fuzzy Systems and Control: Design and Stability Analysis.* Prentice-Hall, 1994.

[57] B. Kosko, *Neural Networks and Fuzzy Systems.* Prentice-Hall, 1992.

[58] B. Kosko, *Fuzzy Engineering.* Englewood Cliffs, N.J.: Prentice-Hall, 1996.

[59] Special Issue on Dynamic Recurrent Neural Networks, *IEEE Trans. Neural Networks*, Vol.5, No.2, 1994.

[60] S.A. Billings and C. Fung, "Recurrent radial basis function networks for adaptive noise cancellation," *Neural Networks*, Vol.2, pp.273–290, 1995.

[61] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA.: Addison-Wesley, 1989.

[62] S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by simulated annealing," *Science*, Vol.220, pp.671–680, 1983.

[63] S.A. Billings and G.L. Zheng, "Radial basis function network configuration using mutual information and the orthogonal least squares algorithm," *Neural Networks*, to appear, 1996.

[64] S.A. Billings and G.L. Zheng, "Radial basis function network configuration using genetic algorithms," *Neural Networks*, Vol.6, pp.877–890, 1995.

[65] S. Chen, S.A. Billings and P.M. Grant, "Recursive hybrid algorithm for non-linear system identification using radial basis function networks," *Int. J. Control*, Vol.55, No.5, pp.1051–1070, 1992.

[66] J. Hertz, A. Krough and R. Palmer, *Introduction to the Theory of Neural Computation.* Redwood City, CA: Addison Wesley, 1991.

[67] A.E. Hoerl and R.W. Kennard, "Ridge regression: biased estimation for nonorthogonal problems," *Technometrics*, Vol.12, No.1, pp.55–67, 1970.

[68] G.H. Golub, M. Heath and G. Wahba, "Generalized cross-validation as a method for choosing a good ridge parameter," *Technometrics*, Vol.21, No.2, pp.215–223, 1979.

[69] S. Chen, "Radial basis functions for signal prediction and system modelling," *J. Applied Science and Computations*, Special Issue on the Theory and Applications of Radial Basis Functions, Vol.1, No.1, 1994

[70] A.S. Weigend, B.A. Huberman and D.E. Rumelhart, "Predicting the future: a connectionist approach," *Int. J. Neural Systems*, Vol.1, No.3, pp.193–209, 1990.

[71] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis.* New York: John Wiley, 1973.

[72] C. Chinrungrueng and C.H. Séquin, "Optimal adaptive κ-means algorithm with dynamic adjustment of learning rate," *IEEE Trans. Neural Networks*, Vol.6, No.1, pp.157–169, 1995.

[73] S. Chen, "Nonlinear time series modelling and prediction using Gaussian RBF networks with enhanced clustering and RLS learning," *Electronics Letters*, Vol.31, No.2, pp.117–118, 1995.

[74] I. Cha and S.A. Kassam, "Interference cancellation using radial basis function networks," *Signal Processing*, Vol.47, No.3, 1995.

[75] P.E. An, M. Brown, C.J. Harris and S. Chen, "Comparative aspects of neural network algorithms for on-line modelling of dynamic processes," *Proc. I. MECH. E., Pt.I, J. Systems and Control Eng.*, Vol.207, pp.223–241, 1993.

[76] E.L. Sutanto and K. Warwick, "Multivariable cluster analysis for high speed industrial machinery," *IEE Proc. Meas. Technol.*, Vol.142, No.5, pp.417–423, 1995.

[77] G.L. Zheng and S.A. Billings, "Radial basis function network training using a fuzzy clustering scheme," submitted to *Mechanical Systems and Signal Processing*, 1996.

[78] W. Luo, S.A. Billings and K.M. Tsang, "On-line structure detection and parameter estimation with exponential windowing for nonlinear systems," submitted to *European J. Control*, 1996.

[79] C. Fung, S.A. Billings and W. Luo "On-line supervised adaptive tracking using radial basis function networks," *Neural Networks*, to appear, 1996.

[80] I.J. Leontaritis and S.A. Billings, "Experiment design and identifiability for nonlinear systems," *Int. J. Systems Sci.*, Vol.18, No.1, pp.189–202, 1987.

[81] S. Chen, "Modelling and identification of nonlinear dynamic systems," in *Proc. SPIE Advanced Signal Processing: Algorithms, Architectures, and Implementations V* (San Diego, USA), July 24–29, 1994, pp.270–278.

[82] S.A. Billings and W.S.F. Voon, "Piecewise linear identification of nonlinear systems," *Int. J. Control*, Vol.46, pp.215–235, 1987.

[83] H. Pi and C. Peterson, "Finding the embedding dimension and variable dependencies in time series," *Neural Computation*, Vol.6, pp.509–520, 1994.

Numerator p_N		Denominator p_D	
	constant		constant
	$y(k-1)$		$y(k-1)$
	$u(k-1)$		$u(k-1)$
	$\epsilon(k-1)$		$\epsilon(k-1)$
	$y^2(k-1)$		$y^2(k-1)$
	$y(k-1)u(k-1)$		$y(k-1)u(k-1)$
	$y(k-1)\epsilon(k-1)$		$y(k-1)\epsilon(k-1)$
	$u^2(k-1)$		$u^2(k-1)$
	$u(k-1)\epsilon(k-1)$		$u(k-1)\epsilon(k-1)$
	$\epsilon^2(k-1)$		$\epsilon^2(k-1)$

Table 1: Full Rational Model Set for Example 1.

Elimination step	Eliminated parameter	C-criterion value
	full model	-2678.4
1	$u(k-1)\epsilon(k-1)$	-2682.4
2	$y^2(k-1)$	-2686.3
3	$u^2(k-1)$	-2690.0
4	constant	-2693.5
5	$u(k-1)^*$	-2697.3
6	$\epsilon^2(k-1)$	-2700.6
7	$\epsilon(k-1)^*$	-2703.3
8	$y(k-1)^*$	-2705.7
9	$y(k-1)\epsilon(k-1)^*$	-2708.2
10	$y(k-1)u(k-1)^*$	-2710.3
11	$u^2(k-1)^*$	-2712.0
12	$\epsilon(k-1)$	-2714.2
$\underline{13}$	$\epsilon^2(k-1)^*$	$\underline{-2715.4}$
14	$y(k-1)\epsilon(k-1)$	-2695.5

Table 2: Model Reduction Using Stepwise Backward Elimination for Example 1. The underline indicates where the procedure stops and $*$ denotes a denominator parameter.

	Model terms	Parameter estimates
Numerator	$y(k-1)$	0.61074
	$u(k-1)$	0.61245
	$y(k-1)u(k-1)$	0.60968
	$y(k-1)\epsilon(k-1)$	0.58427
Denominator	constant	0.60820
	$y^2(k-1)$	0.61379
	$u(k-1)\epsilon(k-1)$	0.55981

Table 3: Final Subset Rational Model for Example 1.

	Model terms selected	Parameter estimates	$[err]_l$
Initial iteration	$u(k-2)$	0.10110E+1	0.67104E+0
$\eta = 0.032$	$y(k-1)$	0.63448E+0	0.28703E+0
	$u^2(k-1)$	0.86768E-1	0.85520E-2
	$y(k-3)u(k-2)$	-0.20542E-1	0.63062E-3
	$u^3(k-3)$	-0.64772E-1	0.55407E-3
	$y(k-2)$	-0.68662E-1	0.15841E-2
	σ_ϵ^2	0.50169E-1	
	$\sigma_\epsilon^2/\sigma_y^2$	0.30603E-1	
1st iteration	$u(k-2)$	0.10073E+1	0.67104E+0
$\eta = 0.03$	$y(k-1)$	0.50464E+0	0.28703E+0
	$u^2(k-1)$	0.92469E-1	0.85520E-2
	$\epsilon(k-1)$	0.40841E+0	0.49355E-2
	σ_ϵ^2	0.44924E-1	
	$\sigma_\epsilon^2/\sigma_y^2$	0.27404E-1	
2nd iteration	$u(k-2)$	0.10052E+1	0.67103E+0
$\eta = 0.027$	$y(k-1)$	0.50226E+0	0.28703E+0
	$u^2(k-1)$	0.90645E-1	0.85520E-2
	$\epsilon(k-1)$	0.49454E+0	0.61062E-2
	$u(k-1)\epsilon(k-2)$	0.22055E+0	0.14793E-2
	σ_ϵ^2	0.40719E-1	
	$\sigma_\epsilon^2/\sigma_y^2$	0.24839E-1	
3rd iteration	$u(k-2)$	0.10045E+1	0.67104E+0
$\eta = 0.025$	$y(k-1)$	0.50171E+0	0.28703E+0
	$u^2(k-1)$	0.90395E-1	0.85520E-2
	$\epsilon(k-1)$	0.54986E+0	0.68936E-2
	$u(k-1)\epsilon(k-2)$	0.25074E+0	0.17351E-2
	σ_ϵ^2	0.40230E-1	
	$\sigma_\epsilon^2/\sigma_y^2$	0.24540E-1	
4th iteration	$u(k-2)$	0.10033E+1	0.67047E+0
$\eta = 0.025$	$y(k-1)$	0.50289E+0	0.28735E+0
	$u^2(k-1)$	0.90966E-1	0.855640E-2
	$\epsilon(k-1)$	0.54796E+0	0.69244E-2
	$u(k-1)\epsilon(k-2)$	0.23848E+0	0.16174E-2
	σ_ϵ^2	0.41020E-1	
	$\sigma_\epsilon^2/\sigma_y^2$	0.25073E-1	
5th iteration	$u(k-2)$	0.10032E+1	0.67047E+0
$\eta = 0.0251$	$y(k-1)$	0.50281E+0	0.28735E+0
	$u^2(k-1)$	0.91097E-1	0.85640E-2
	$\epsilon(k-1)$	0.54843E+0	0.69317E-2
	$u(k-1)\epsilon(k-2)$	0.23785E+0	0.16098E-2
	σ_ϵ^2	0.41019E-1	
	$\sigma_\epsilon^2/\sigma_y^2$	0.25072E-1	

Table 4: Iterative Procedure of Subset Model Selection for Example 2.

node	link	weight	saliency	node	link	weight	saliency
output	1	-11.66116	63595.75425	hidden 3	0	5.20112	6.10191
	2	16.42509	657.95127		1	3.69828	3.66240
	3	0.28607	33.88771		2	-0.30648	0.02250
	4	-28.59449	1152.50990		3	-0.39791	0.03398
	5	26.22748	58284.29225		4	9.21591	3.19361
					5	-8.45146	2.39434
					6	4.75899	0.70737
					7	-3.67219	0.40132
					8	1.65329	0.07686
hidden 1	0	7.81765	881.21335	hidden 4	0	-5.39978	10370.91985
	1	-4.33839	872.64808		1	1.96907	3646.88325
	2	-6.75760	2511.20255		2	2.99032	9345.46625
	3	12.57424	9719.85827		3	-6.47447	45683.86413
	4	5.54735	100.48206		4	-0.64769	22.94355
	5	-1.23030	4.72108		5	-1.65110	158.83720
	6	8.51095	246.69947		6	-3.42483	901.78776
	7	-5.88247	150.03042		7	2.20655	471.78389
	8	1.35375	6.89820		8	-0.17182	2.75034
hidden 2	0	-5.08538	5593.24296	hidden 5	0	-0.26397	1342.97186
	1	1.69406	1548.08633		1	0.10828	88.46429
	2	2.72633	4352.99617		2	0.04182	13.22511
	3	-6.10927	22012.26445		3	-0.00407	0.12590
	4	0.13381	0.54297		4	0.04258	3.49782
	5	-2.33698	187.10634		5	-0.00018	0.00006
	6	-2.94343	399.16156		6	-0.01648	0.52305
	7	2.00734	224.53101		7	0.01038	0.20746
	8	-0.17158	1.60094		8	-0.00493	0.04687

Table 5: Fully Connected 8-5-1 Feedforward Network Model Obtained Using Batch PPEA for Example 4. The underlined weights are to be eliminated.

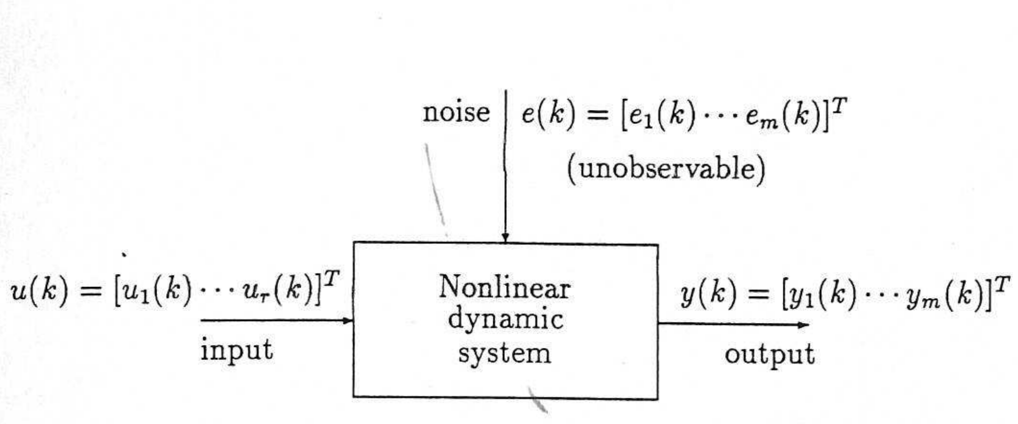

noise
$e(k) = [e_1(k) \cdots e_m(k)]^T$
(unobservable)
$u(k) = [u_1(k) \cdots u_r(k)]^T$
input
Nonlinear dynamic system
$y(k) = [y_1(k) \cdots y_m(k)]^T$
output

Figure 1: Block Diagram of Multivariable Nonlinear Dynamic System.

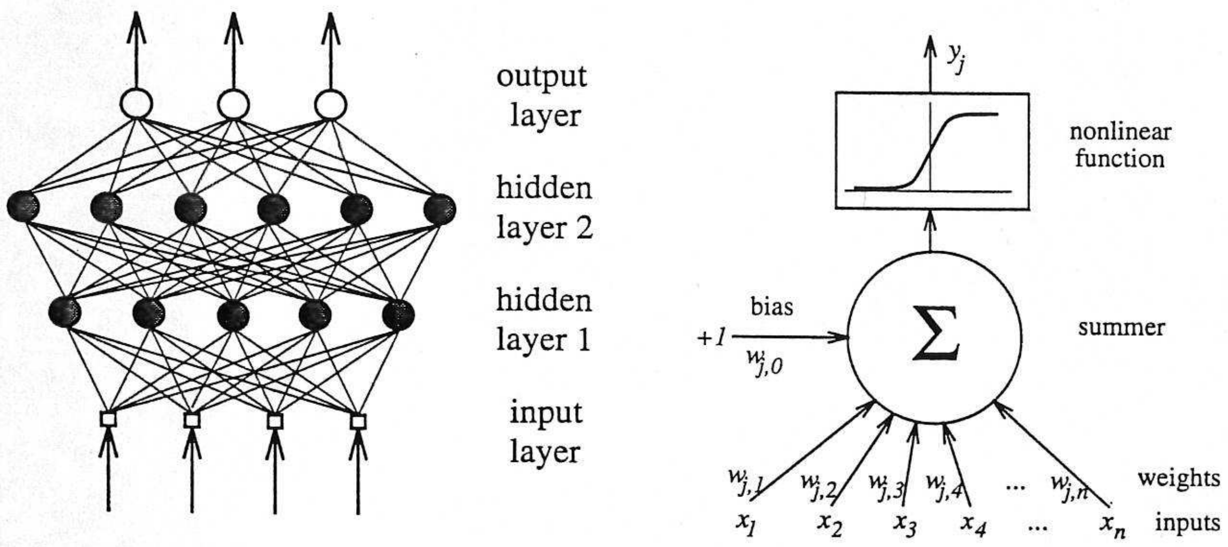


Figure 2: Multilayer Perceptron (a) and Model of A Neuron (b).

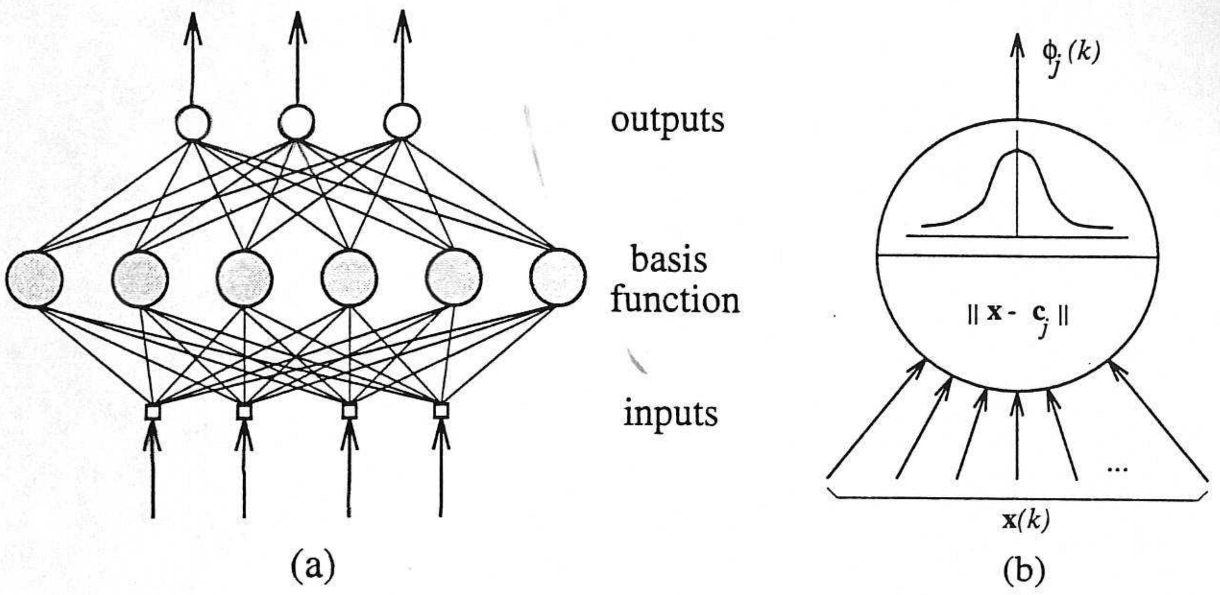


Figure 3: Radial Basis Function Network (a) and Model of A Basis Function (b).

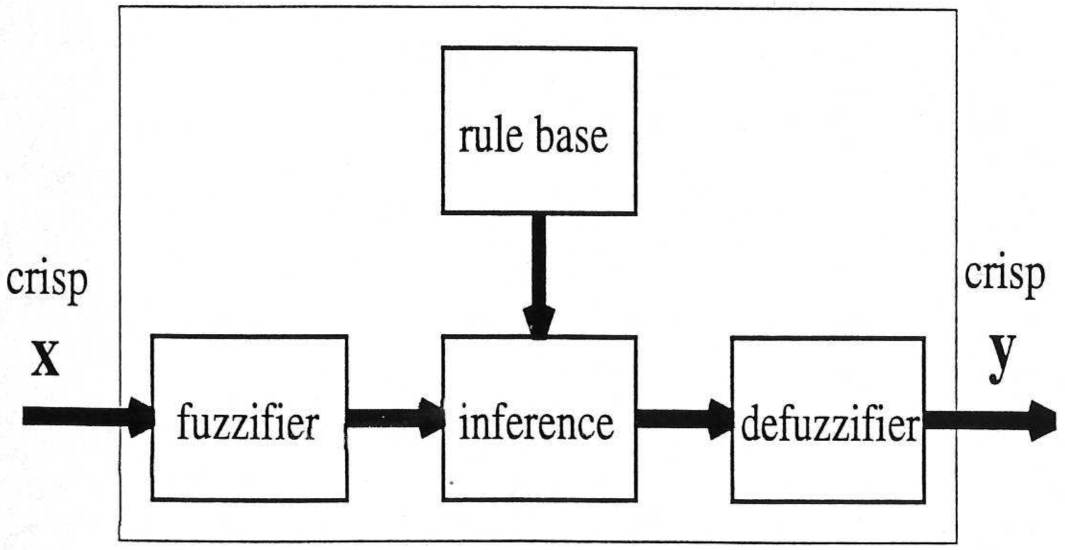


Figure 4: MIMO Fuzzy System Architecture.

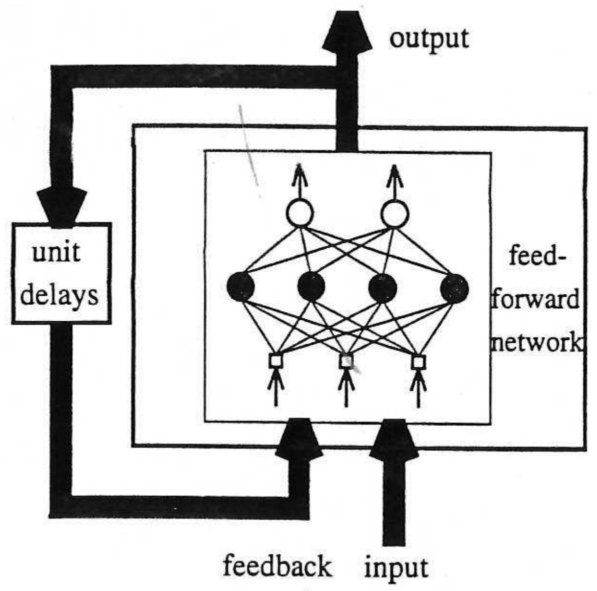


Figure 5: Recurrent Network Obtained by Feeding Back the Network Output to the Input.

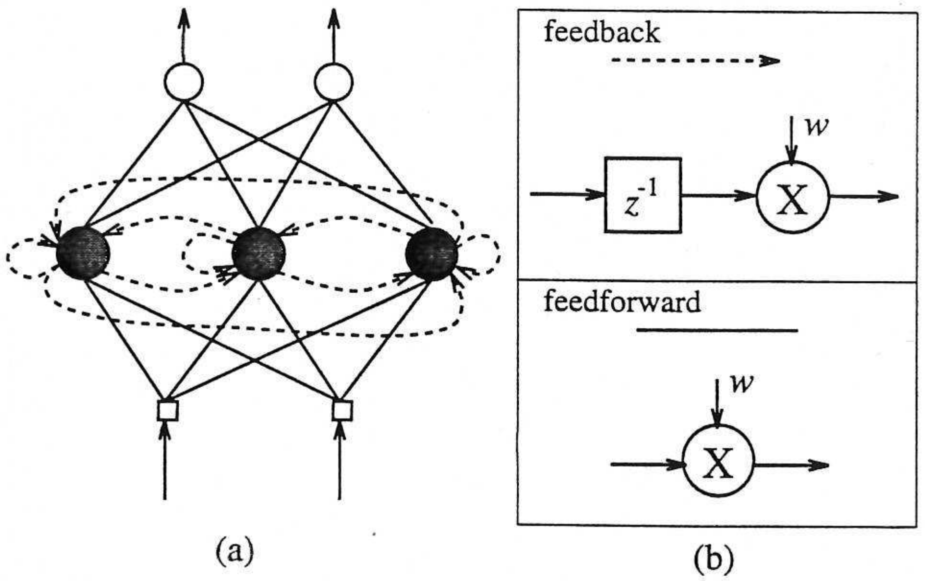


Figure 6: Recurrent Network with Internal Feedback (a) and Two Kinds of Connections (b).

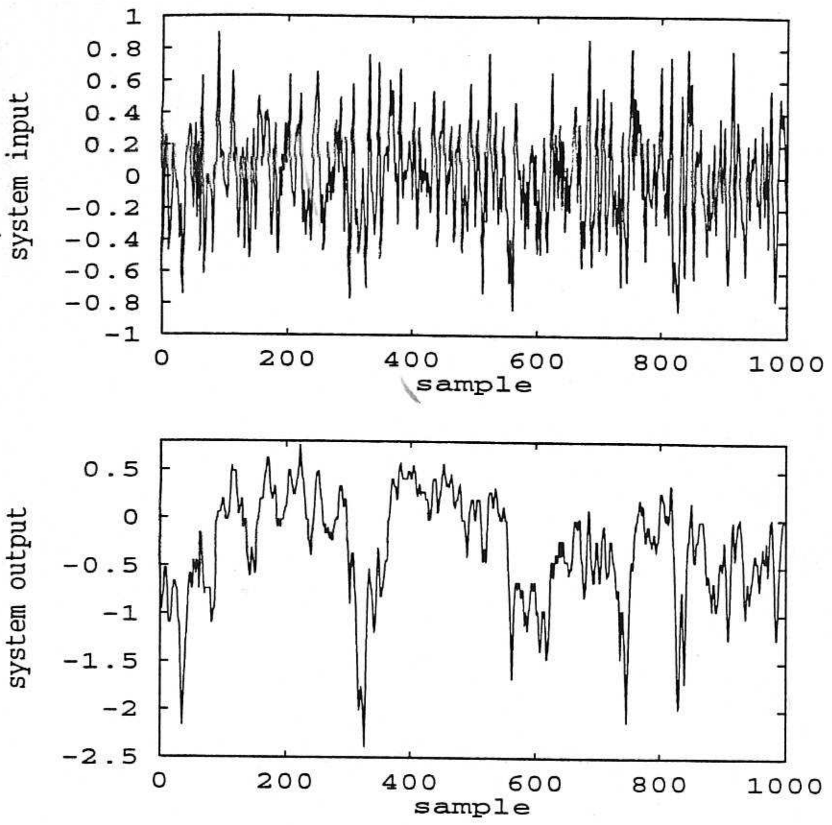


Figure 7: Inputs and Outputs of A Liquid Level System.

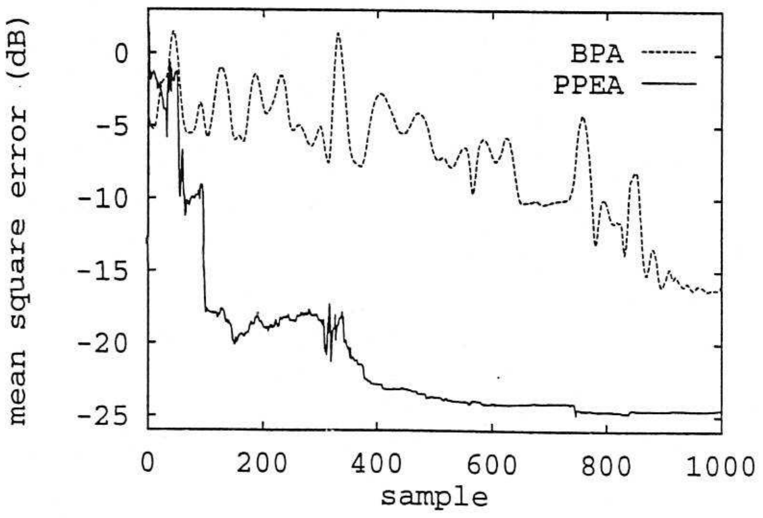
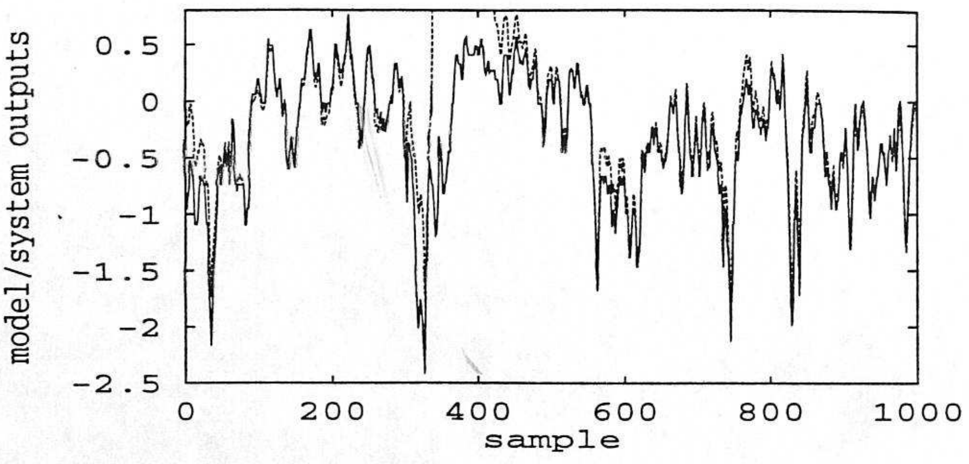
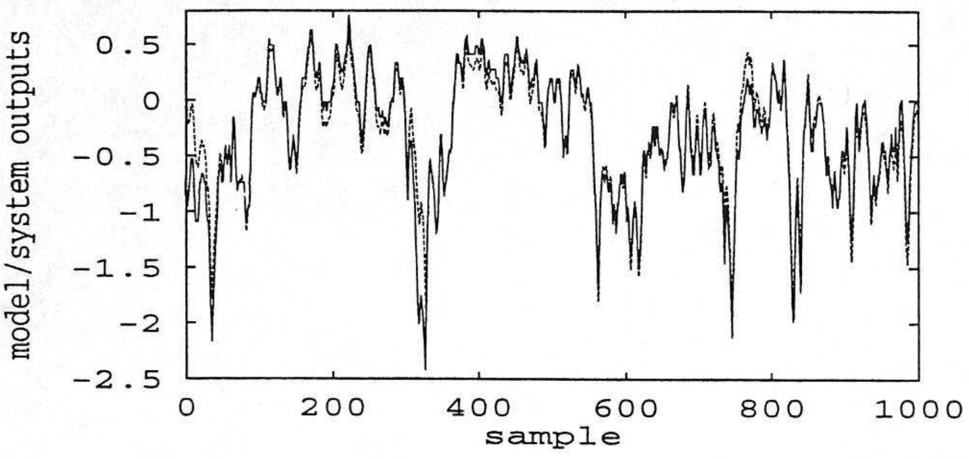


Figure 8: Evolution of Mean Square Error (Liquid Level System).



(a)



(b)

Figure 9: Iterative Model Ouput (Dashed) Superimposed on Liquid Level System Output (Solid). (a) Full network model; (b) Pruned network model.

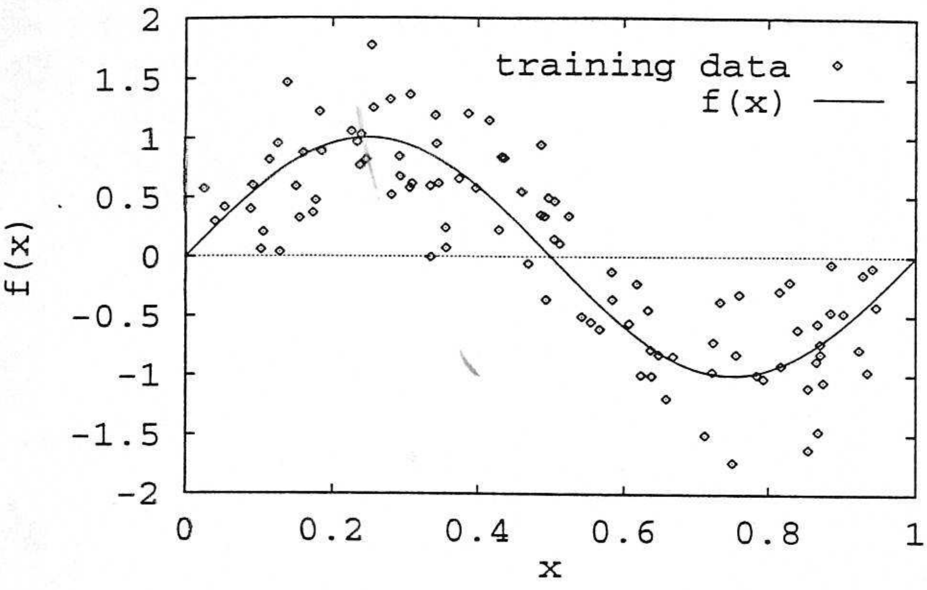


Figure 10: Noisy Training Data (Points) and Underlying Function (Curve) for Example 5.

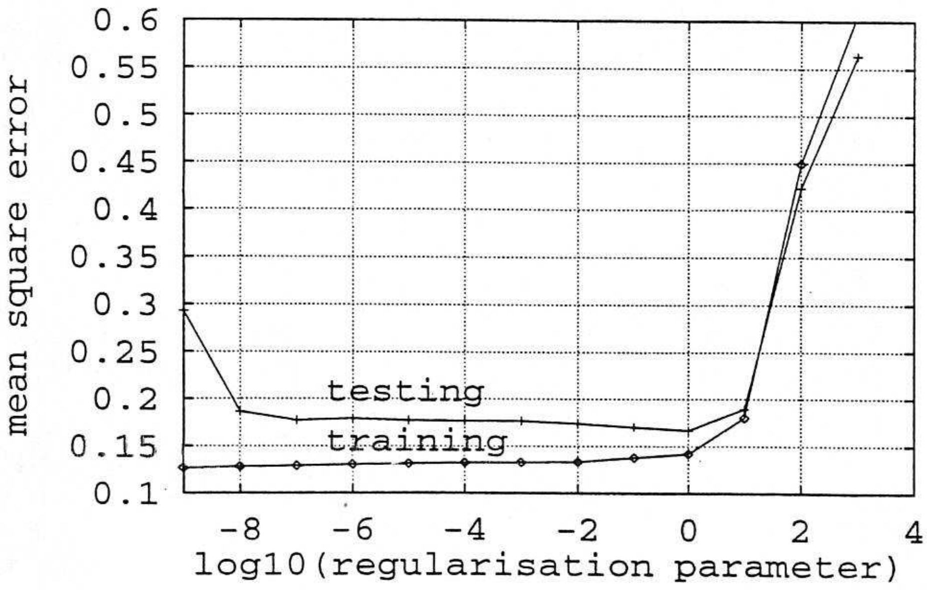


Figure 11: Mean Square Error as A Function of the Regularisation Parameter for Example 5.

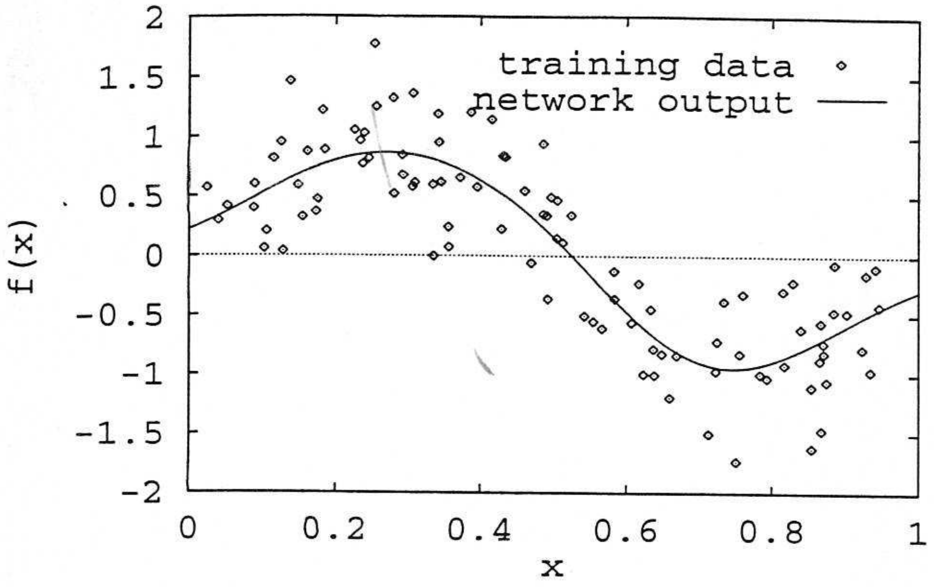


Figure 12: Network Mapping Constructed by the ROLS Algorithm for Example 5.

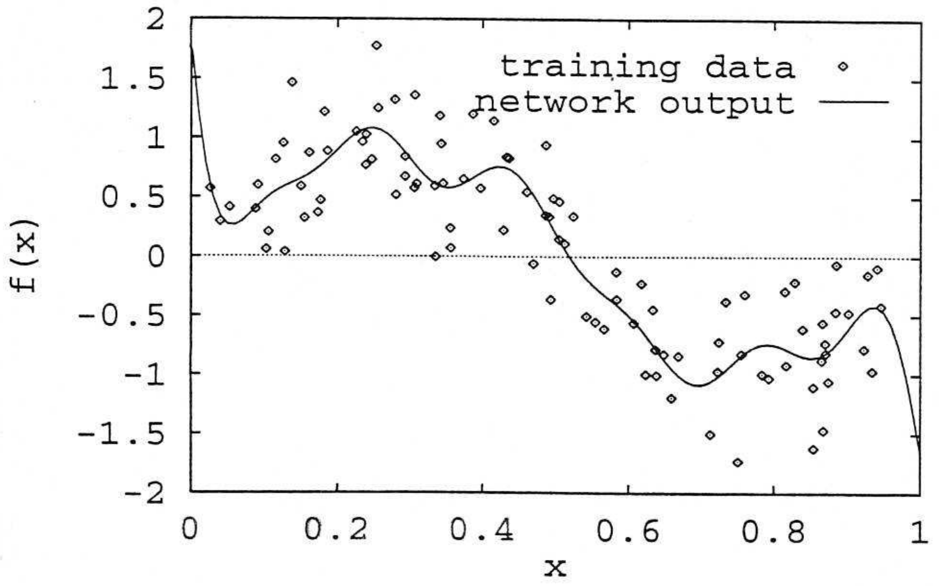


Figure 13: Network Mapping Constructed by the OLS Algorithm for Example 5.

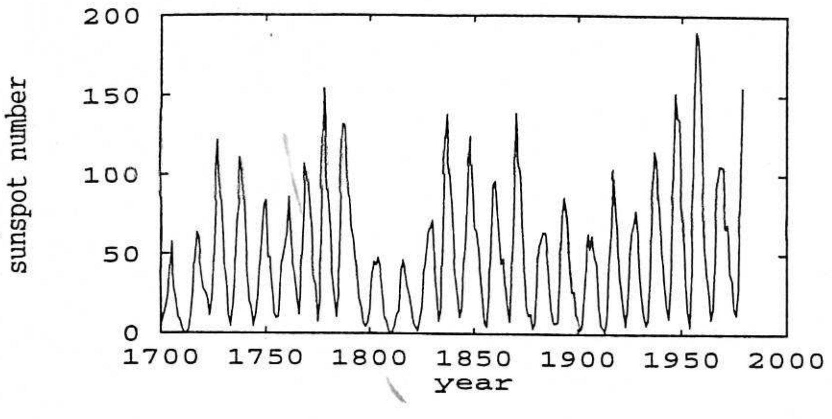
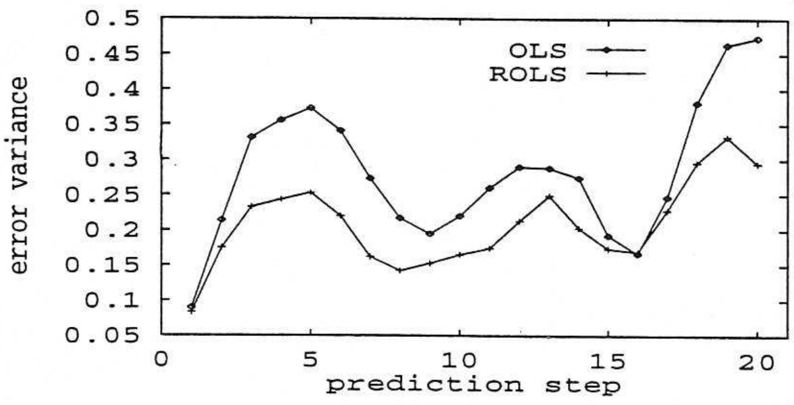
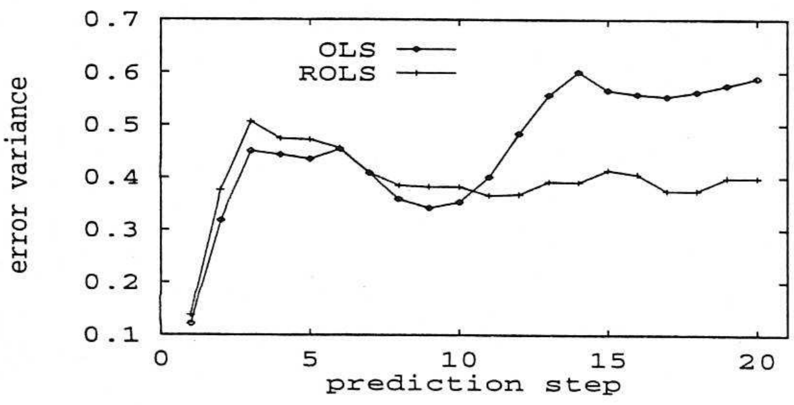


Figure 14: Time Series of Annual Sunspot Numbers.



(a) over years 1921–1955.



(b) over years 1921–1979

Figure 15: Normalised Variances of Multi-Step Prediction Errors for Sunspot Time Series.

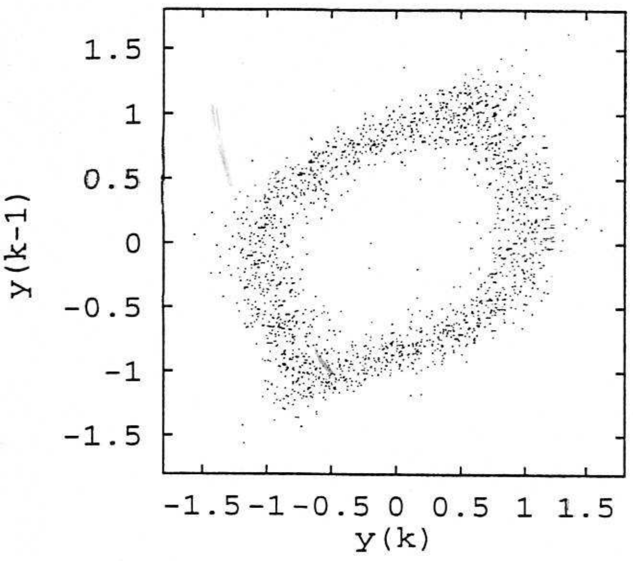


Figure 16: Noisy Observations of the Two-Dimensional Time Series Example.

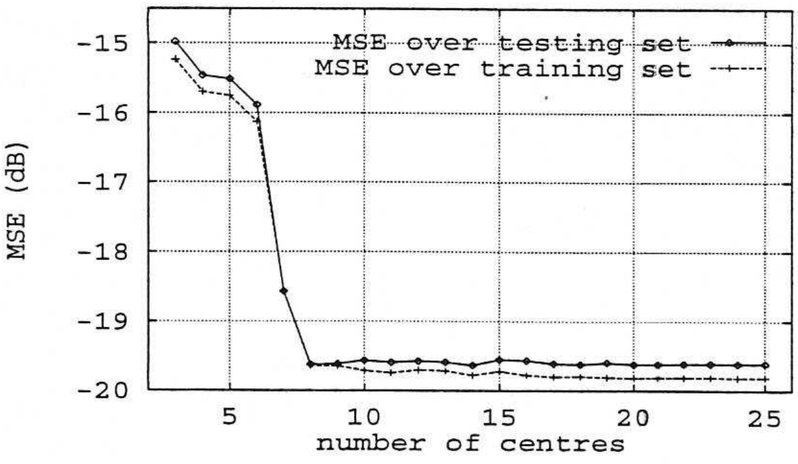


Figure 17: Mean Square Error as A Function of Centre Number for the Two-Dimensional Time Series Example.

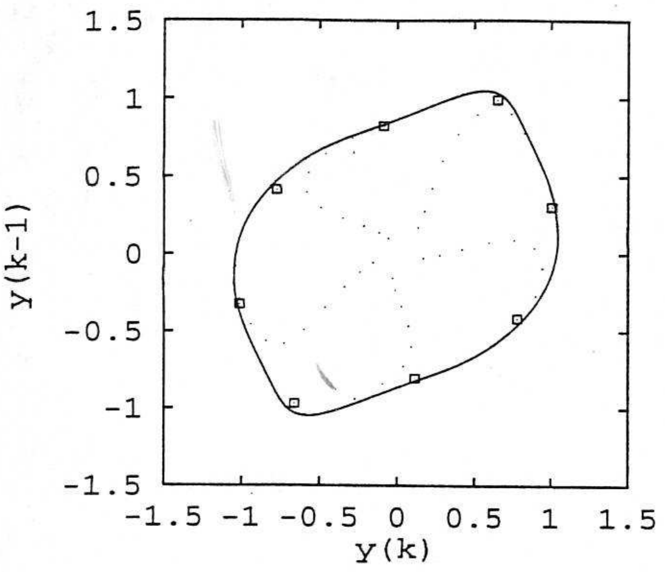


Figure 18: Noise-Free Two-Dimensional System (Limit Cycle) and RBF Centre Locations (□).

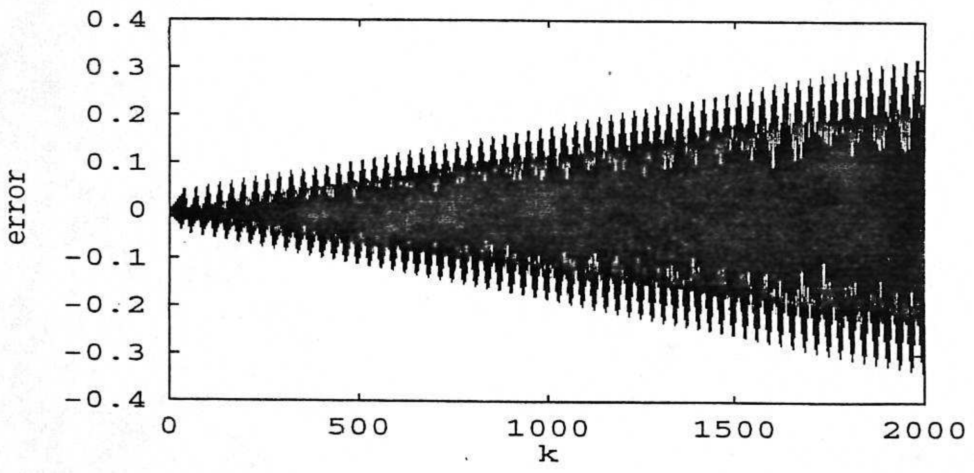


Figure 19: Iterative Network Model Errors for the Two-Dimensional Time Series Example.

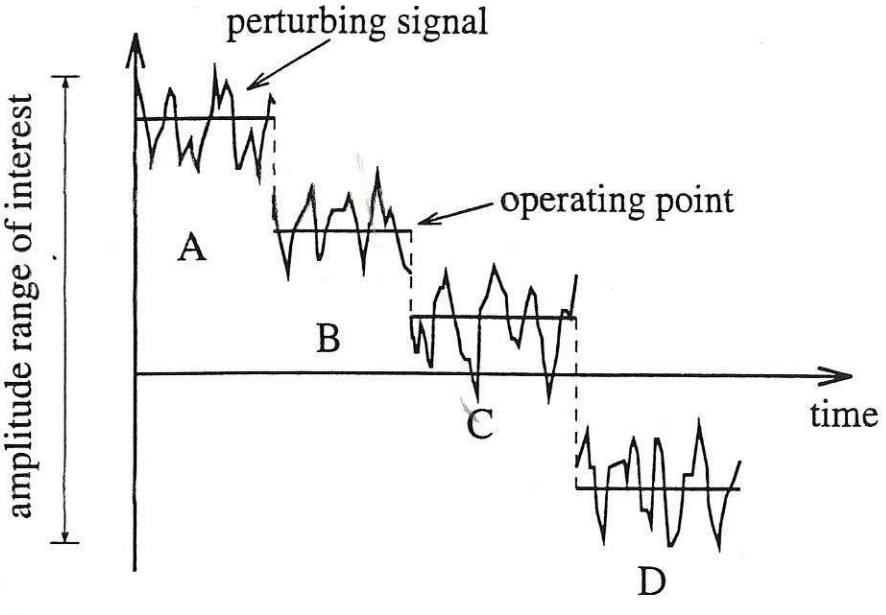


Figure 20: Input Design for Nonlinear System Identification. Perturbing data sets A to D collected together excite the amplitude range of interest.

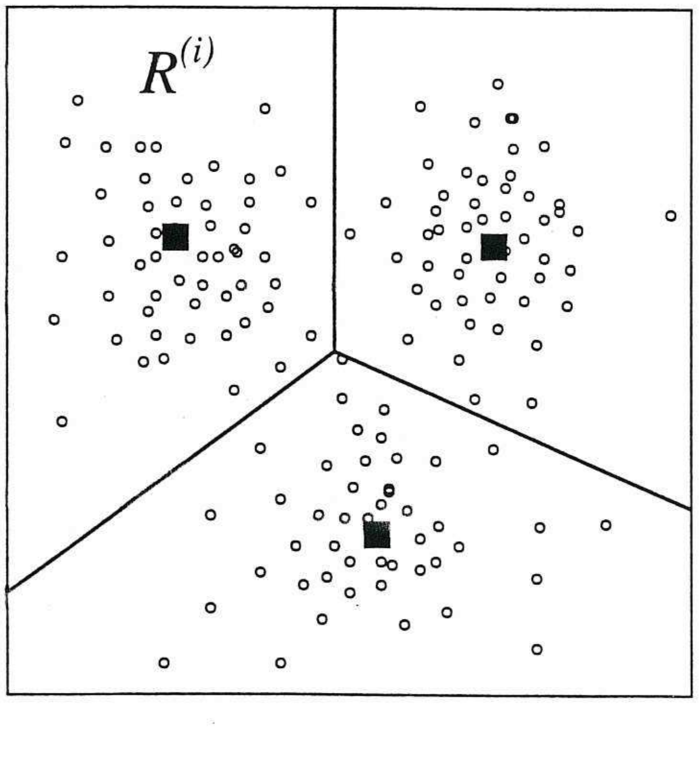


Figure 21: Model Input Space Partition Using Clustering.