Comparative aspects of neural network algorithms for on-line modelling of dynamic processes

P E An, BSc, MSc, PhD, **M Brown**, BSc, PhD and **C J Harris**, BSc, MA, PhD
Department of Aeronautics and Astronautics, University of Southampton

S Chen, BSc, PhD
Department of Electrical Engineering, University of Edinburgh

This paper reviews the model structures and learning rules of four commonly used artificial neural networks: the cerebellar model articulation controller (CMAC), B-splines, radial basis functions (RBF) and multi-layered perceptron (MLP) networks. Their dynamic modelling abilities are compared using a two-dimensional non-linear noisy time series. The network performances are evaluated based on their network surface plots, phase/time history plots, learning curves, prediction error autocorrelation functions and finally their short-range prediction error variances. The modelling results suggest that all four networks were able to capture the underlying dynamics of the time series. Also, specific prior knowledge about the time series was incorporated into the B-splines model, and is used to highlight an important trade-off between the model flexibility and high-dimensional modelling ability in the B-splines and CMAC networks. In general, when the network model is well conditioned and linear with respect to its adaptable parameters, simpler on-line learning rules often provide adequate convergence properties. Alternatively, when the model is highly non-linear, complicated learning rules which utilize high-order gradient information are generally required at the expense of increased computational complexity.

1 INTRODUCTION

In traditional modelling and control applications, a linear adaptive model* has often been used to model an unknown process or to form an inverse mapping of the controlled process. Being characterized by its transient convergence behaviour and steady-state mismatch, the modelling performance relies greatly on the characteristic of the process, and is *optimal* when the process is linear and time invariant. However, when the process is highly non-linear, its behaviour in different operating regions can vary significantly. This means that a well-learned linear model in some region might have a catastrophic effect in other untrained regions unless adequate relearning has taken place. Because the model is linear, the relearning procedure often destroys the previously stored knowledge about the process. In order to ensure stable transition, the drift of the process's operating region must be slow compared with the model adaptation, which imposes a stringent condition on the process dynamics.

Unlike the linear model, artificial neural networks† can generally approximate any continuous multi-dimensional non-linear function (**1**, **2**), and have been widely explored in the area of modelling and control applications (**3**–**8**). While the learning capabilities of these networks can vary significantly depending on the non-linearity incorporated in their model structures, these networks share an important characteristic: they internally transform‡ every training input into a higher dimensional space so that the desired output can be made approximately *linear* to the transformed input.

This paper was presented at an Ordinary Meeting held in London on 16 March 1993. The MS was received on 23 March 1993 and was accepted for publication on 25 August 1993.

* With respect to the process input.

† This usage should be understood as algorithms which approximate mathematical functions, rather than as biological models which account for electrochemical activities among neurons.

‡ This transformation can either be fixed or adaptive.

There are many ways to describe the non-linearities defined in these networks. One useful way of describing them is based on the type of generalization by which the network transforms the input. In general, neural networks can be classified as either globally generalizing or locally generalizing. The generalization is considered local if only few adaptable parameters can potentially affect the network output for each input. Examples are lattice associative networks (LANs), such as the cerebellar model articulation controller (CMAC) and B-splines (**9**, **10**). On the other hand, the generalization is considered global if each adaptable parameter can potentially affect the network output at every point in the input space. Examples are multi-layered perceptron (MLP) and radial basis functions (RBF) networks (**11**–**13**).

Another useful way of describing the non-linearity is based on the relationship of the network output to its adaptable parameters. This relationship is linear in the lattice associative network, and linear optimization techniques can readily be applied. However, the same relationship in the MLP network is highly non-linear and the cost function is highly irregular, containing plateaus and local sub-optimal solutions.

With this specific description of non-linearities, the RBF network can be considered as an intermediate model which lies between the MLP and lattice associative networks. While the RBF network has global support (similar to the MLP), the energy of the Gaussian basis function is mostly local, similar to the LAN. Also, the inner structure of the model can be either fixed, similar to the LAN, or can be made adaptive, as with the MLP network. Based on these descriptions, a wide variety of existing neural networks can be represented by the MLP, RBF and associative networks.

In this paper, the model structures and learning rules of the CMAC, B-splines, RBF and the MLP are described. The modelling abilities of these networks are evaluated using a two-dimensional non-linear time

series. This time series has sufficiently rich dynamic properties such that useful insights can be gained about the low-dimensional modelling abilities of these networks. The evaluation measures for the time series modelling are based on their network surfaces' reconstructions, phase/time history plots, error autocorrelation functions and variance characteristics, and also their learning curves. These measures can generally provide an accurate indication of the dynamic modelling performance. The rest of this paper is organized as follows. In Section 2, the model structures and the learning rules of the LANs are reviewed. Sections 3 and 4 present the model structures and learning rules for the RBF and MLP networks. Section 5 describes the time series experiment and the four network model conditions. Section 6 discusses the time series modelling results and, finally, Section 7 presents comments on the general modelling abilities of these networks.

2 LATTICE ASSOCIATIVE NETWORK (LAN)

The output of a LAN is formed of a linear combination of overlapping basis functions which are evenly distributed in an n-dimensional sub-space of R^n. Each of the basis functions is defined on a hyper-rectangular region which is a compact region in R^n, and this is known as its *receptive field*. Therefore the output of each basis function is non-zero only when the input lies in its receptive field. This feature endows the LANs with an ability to generalize locally: similar inputs are mapped onto nearby hyper-rectangular receptive fields, which produce similar outputs, while dissimilar inputs are mapped onto distant hyper-rectangles, and this produces independent outputs. A general associative memory network in which the input is non-linearly transformed is shown in Fig. 1. Notice that the function approximation is only valid for a bounded input space because of the finite number of localized basis functions, although the function may be assumed to remain constant outside the bounded input space.

2.1 Model structure

The network output at time t, $\hat{y}\{\boldsymbol{x}(t)\}$, is computed by first transforming the input vector, $\boldsymbol{x}(t)$ ($\in R^n$), into a higher dimensional space (R^p), which is generated by the outputs of the p basis functions. The inner product of the transformed input vector, $\phi\{\boldsymbol{x}(t)\}$ (containing the outputs of the individual basis functions), with a p-dimensional adjustable weight vector, $\boldsymbol{w}(t-1)$, is then calculated, so the network output is given by

$$\hat{y}\{\boldsymbol{x}(t)\} = \phi^T\{\boldsymbol{x}(t)\} \cdot \boldsymbol{w}(t-1) \tag{1}$$

Because of the compact receptive fields, $\phi\{\boldsymbol{x}(t)\}$ has only a few non-zero components, and if a simple algorithm is available for determining which basis functions are non-zero the computational cost of forming the output can be significantly reduced. The implicit dependency of ϕ and $\boldsymbol{w}$ on time will be subsequently dropped in order to simplify the notation. Each $\phi_k(\boldsymbol{x})$ (the kth element of ϕ) is uniquely determined by the type of basis function used in the LAN, in particular the width and the centre of the receptive field relative to $\boldsymbol{x}$. In general, the n-dimensional multi-variate basis function $\phi_k(\boldsymbol{x})$ is defined by its n univariate basis functions and the operator which combines univariate basis functions to produce a single value. Two common composition operators that have also been used in fuzzy systems to represent conjunctions (**14**) are the minimum operator and the product operator. The minimum operator sets $\phi_k(\boldsymbol{x})$ to be the minimum univariate function value, that is $\phi_k(\boldsymbol{x}) = \min\{\phi_{k,1}(x_1), \phi_{k,2}(x_2), \ldots, \phi_{k,n}(x_n)\}$, where x_i is the ith element of $\boldsymbol{x}$ and $\phi_{k,i}(x_i)$ is the ith univariate basis function corresponding to the kth multi-variate basis function, and both are defined on the ith input axis. The product operator computes $\phi_k(\boldsymbol{x})$ by simply multiplying together n univariate function values, that is

$$\phi_k(\boldsymbol{x}) = \phi_{k,1}(x_1) * \phi_{k,2}(x_2) * \cdots * \phi_{k,n}(x_n) = \prod_{i=1}^{n} \phi_{k,i}(x_i)$$

The individual univariate basis functions generate the n-dimensional multi-variate basis functions, therefore only the former need to be described. The ith input axis is partitioned into S_i neighbouring, non-overlapping intervals by a set of $(S_i + 1)$ knots, λ_i. The first and last knots are termed *exterior* knots as they are positioned at the minimum and maximum values of x_i respectively. The remaining $S_i - 1$ values are called *interior* knots, and these represent the position of the end of one interval and the beginning of the next. Each univariate basis function is non-zero only over a small number of adjacent intervals, therefore the position of the knots defines the size of the univariate receptive fields. With respect to the quantized input space, each univariate basis function has a receptive field which is ρ_i intervals wide.

The quantization of each input axis generates an n-dimensional lattice on which the multi-variate basis functions are defined. Each $(n-1)$ dimensional hyperplane which divides the input space passes through one of the univariate knots and is parallel to the remaining $(n-1)$ axes, therefore the lattice is generated by the knot matrix $\Lambda\{ = (\lambda_1\ \lambda_2\ \cdots\ \lambda_n)\}$ which contains the knot vectors for each input axis. The multi-variate basis functions are also defined on receptive fields of size $\boldsymbol{\rho}\{ = (\rho_1\ \rho_2\ \cdots\ \rho_n)\}$ relative to the lattice. This is illustrated in Fig. 2 for a two-dimensional input space with $\rho_1 = 4$, $\rho_2 = 3$. It is important to notice that the knots do not always specify the *centres* of the basis functions as generally, when ρ_i is odd, they occur midway between two knots.

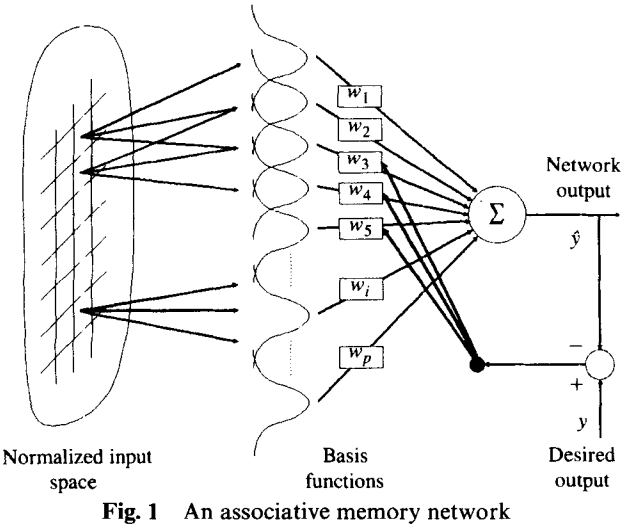


Fig. 1 An associative memory network

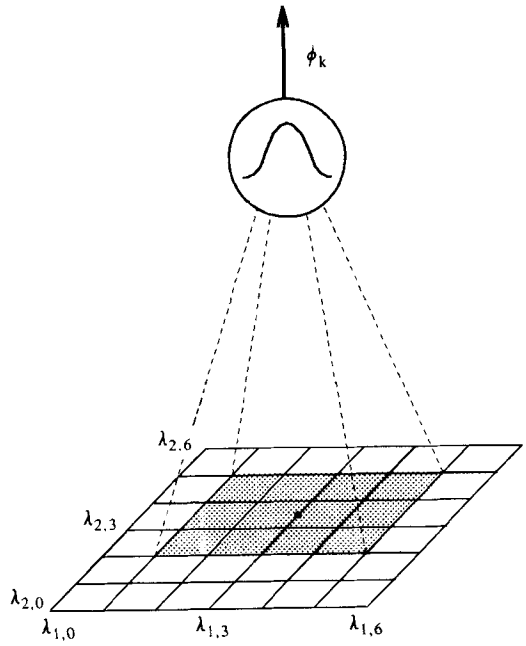


Fig. 2 A two-dimensional input lattice with univariate knots. The grid points are formed at the intersection of the hyperplanes, and the dot represents the centre of the shaded receptive field

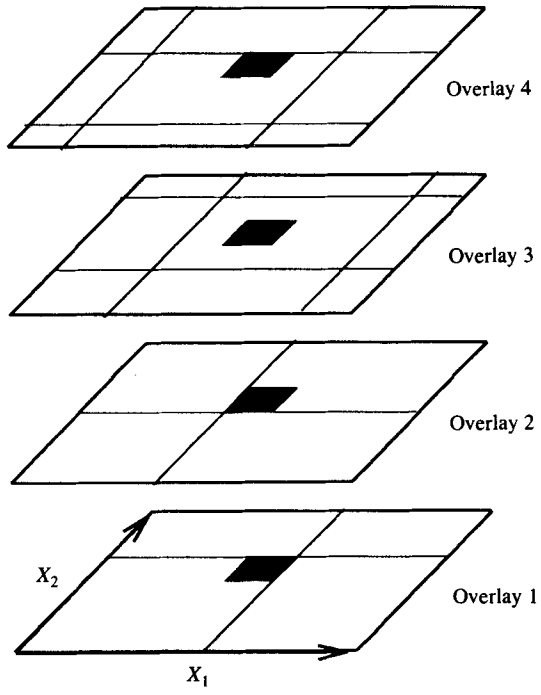


Fig. 3 CMAC overlay structure: $\rho = \kappa = 4$

In general, the knot density is chosen to be uniform in order to preserve a uniform resolution of the network response in R^n. If any prior knowledge is available, the knot density can be arranged so that more knots are placed in some part of the input space where the function varies significantly, and fewer knots in other area where the function is approximately constant. ρ is an important parameter vector (fixed prior to learning) which significantly affects the approximation capability and the rate of convergence of the network. When ρ is chosen too large, the network is *slow* to learn a function containing high spatial Fourier components. On the other hand, when ρ is chosen too small, the network is *unable* to generalize between neighbouring training samples.

The evenly distributed set of the basis functions can be geometrically decomposed into κ sets of overlays. An overlay is defined as a union of basis functions with non-overlapping hyper-rectangular receptive fields which exactly covers the input lattice. These overlays have different partitioning of the receptive fields so that the same input maps to different basins of the basis functions in different overlays. An example of the overlay structure for a two-dimensional input is given in Fig. 3. This overlay arrangement thus forces the input to lie in one and *only* one active receptive field and, in turn, κ non-zero basis functions in the entire network. Notice that the number of available basis functions in each overlay varies, depending on a displacement matrix, **D**, which defines the partitioning configuration. The displacement matrix contains the offset values $d_{i,j}$ and is defined as

$$\begin{bmatrix} 0 & 0 & \dots & 0 \\ d_{1,2} & d_{2,2} & \dots & d_{n,2} \\ \vdots & \vdots & \vdots & \vdots \\ d_{1,k} & d_{2,k} & \dots & d_{n,k} \end{bmatrix} \tag{2}$$

When $d_{i,j}$ is k, the partitioning in the jth overlay starts on the kth interior knot along the ith axis. Thus, $\mathbf{\Lambda}$ determines not only the actual size of the receptive fields, but also the physical offset distance among the overlays. Notice that the displacement matrix for the first overlay has zero offset values, indicating that the partitioning starts on the individual axes. Based on the lattice defined by $\mathbf{\Lambda}$ and the basis function distribution defined by **D**, the number of basis functions p can be calculated from

$$p = \sum_{j=1}^{\kappa} \prod_{i=1}^{n} \left\lceil \frac{S_i + d_{i,j}}{\rho_i} \right\rceil \qquad 0 \leqslant d_{i,j} \leqslant \rho - 1 \tag{3}$$

2.1.1 CMAC

The CMAC network was originally proposed (**15**) as a model of the neurophysiological functioning of the mammalian cerebellum. This network has a unique method for defining receptive fields as they are n-dimensional hypercubes of width ρ, and this is the *same* as the number of overlays in the network κ. Therefore the number of active receptive fields does not explicitly depend on n, and is equal to the receptive field width. While this relationship forces the basis functions to be sparsely distributed on the lattice, the numerical computation for any training input is *linearly* dependent on the input dimension, which is an important characteristic of this network.

The displacement matrix **D** associated with the CMAC has the size $\rho \times n$, and is defined as in equation (4). Based on this matrix, the centres of the basis functions are defined on the diagonal and sub-diagonals on the lattice. An example of the overlay structure is shown in Fig. 3, where n is 2 and ρ is 4. Given that the knot density is uniform in each axis, the diagonal placement provides a uniform projection of receptive fields onto each axis: the displaced input by one interval parallel to any input axis always shares $\rho - 1$ receptive fields with

the undisplaced one.

$$\begin{bmatrix} 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \rho-1 & \rho-1 & \dots & \rho-1 \end{bmatrix} \tag{4}$$

The total number of receptive fields which lies in the input lattice is given in equation (5), which can be further simplified if the receptive field width is much smaller than the number of intervals on each axis, that is $0 \ll \rho \ll S$:

$$p = \sum_{k=0}^{k=\rho-1} \left\lceil \frac{S+k}{\rho} \right\rceil^n \approx \rho \cdot \left(\frac{S}{\rho}\right)^n \approx \frac{S^n}{\rho^{n-1}} \tag{5}$$

Thus for a larger ρ, there will be fewer receptive fields available in the network. This also forces the placement to be less uniform within the hypercube of side ρ. A less uniform placement is likely to deteriorate the approximation capability if the basis functions are input dependent. While a theoretical analysis for an optimal placement scheme with an arbitrary ρ is not available, alternative schemes of improving the uniformity of the receptive field placement have been proposed (**16**, **17**).

A traditional univariate basis function is a binary function which gives rise to piecewise constant approximation of the desired output. The multi-variate basis function can be formed using either the minimum operator or the product operator (Fig. 4a). Higher order piecewise polynomial univariate basis functions which generate a smoother network output have also been investigated (**16**, **18**, **19**). These univariate basis functions have a maximum value at the centre of the receptive field which diminishes as the input moves towards the edge of the receptive field. For these multi-variate basis functions the operator can be either the minimum operator or the product operator. Based on the binary basis function, the modelling capability of the multi-dimensional CMAC network for a certain class of training functions has been investigated (**20**).

2.1.2 B-splines

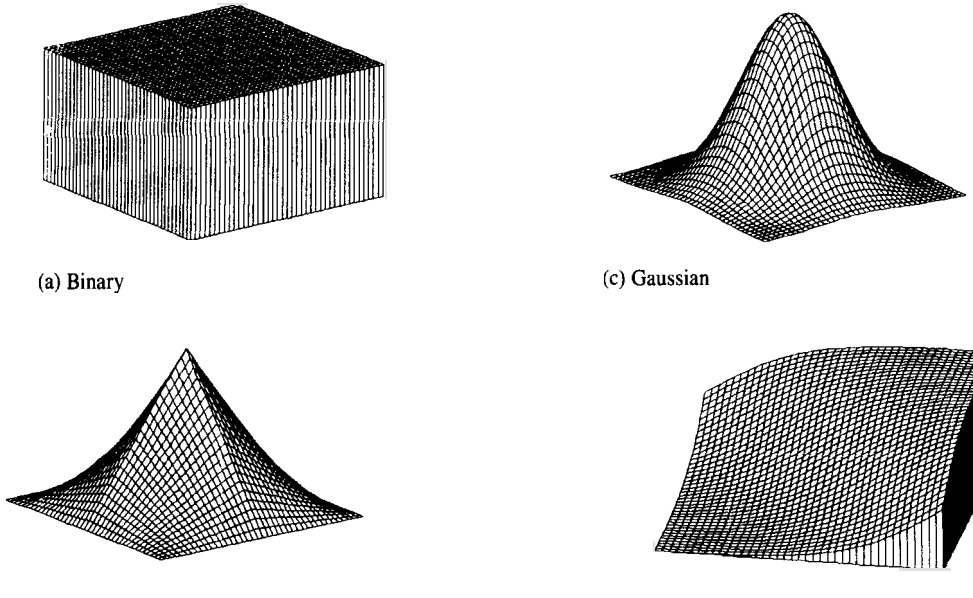
B-splines were originally proposed for use in geometrical modelling and in graphical applications. Unlike the CMAC network, the orders (or smoothness) of the B-splines (or basis functions) and the knots' locations can be chosen arbitrarily along each input axis, which allows the network to incorporate useful prior knowledge about the training function. In addition, there are as many overlays as there are grid points on the lattice. This means that the number of active receptive fields is *exponentially* dependent on n (or κ is ρ^n). The displacement matrix of size ($\rho^n \times n$) is given as in equation (6), and p can be determined as in equation (3). A set of overlays for $\kappa = 4$ and $n = 2$ is shown in Fig. 5.

$$\begin{bmatrix} 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & \rho-1 \\ 0 & \cdots & 1 & 0 \\ 0 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \rho-1 & \cdots & \rho-1 & \rho-1 \end{bmatrix} \tag{6}$$

Also, unlike the CMAC network, the order of the B-splines plays a unique role in determining the widths of the basis functions. A simple and stable recurrence relationship is commonly used to define the univariate B-spline functions $\{\phi_{i,\rho,j}(x_i)\}$ (**21**), and is given as

$$\begin{aligned} \phi_{i,\rho,j}(x_i) &= \left(\frac{x_i - \lambda_{i,j-\rho}}{\lambda_{i,j-1} - \lambda_{i,j-\rho}}\right)\phi_{i,\rho-1,j-1}(x_i) \\ &\quad + \left(\frac{\lambda_{i,j} - x_i}{\lambda_{i,j} - \lambda_{i,j-\rho+1}}\right)\phi_{i,\rho-1,j}(x_i) \\ \phi_{i,1,j}(x_i) &= \begin{cases} 1 \text{ if } & x_i \in I_{j-1} \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{7}$$

where I_j is the jth interval $(\lambda_{i,j}, \lambda_{i,j+1})$ with the last interval being closed at both ends. The multi-variate



(a) Binary

(c) Gaussian

(b) Second-order B-splines

(d) Sigmoid

Fig. 4 Various basis function types

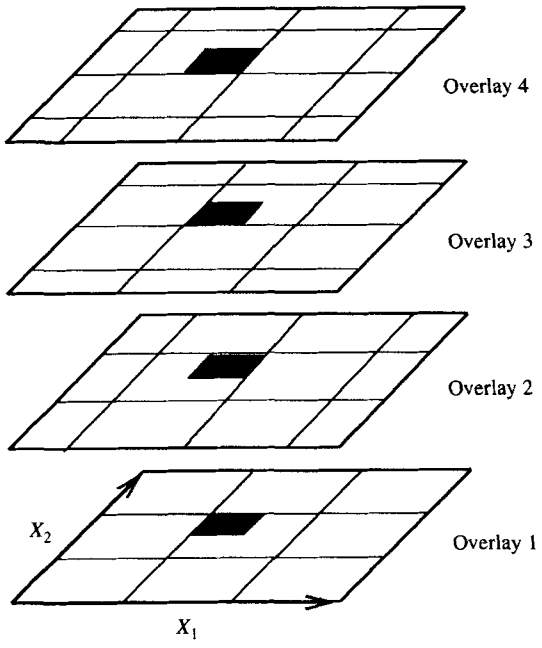


Fig. 5 B-splines overlay structure: $\rho = 2, \kappa = 4$

B-splines are then formed using the product operator. This basis function structure has two important properties. First, the basis functions are normalized so that the sum of ρ^n multi-variate basis function values is independent of the location of the training input. This ensures that the network is not biased toward any region in R^n. Second, as the order (or width) of the B-splines increases, the basis functions become smoother and neighbouring basis functions have a larger area of overlap on their receptive fields (see Fig. 6). For example, a first-order B-spline is a binary function, while a second-order B-spline is a piecewise linear function (Fig. 4b).

A more flexible set of B-splines (dilated B-splines) which allows the width and the order of the B-splines to be decoupled has also been investigated (**19**). The dilated B-splines can be interpreted as a union of coarsely resolved (large physical interval) low-order B-splines networks, each defined in the input space with an appropriate relative offset. When the training function is strongly correlated in some sub-dimensional input space, several variants of the B-splines networks with adaptable model structures have been proposed (**22, 23**). These networks are constructed from a union of sub-models, each defined in a sub-dimensional input space. Iterative refinement procedures are then used to develop more complex sub-models when the inputs are found to be correlated in a higher dimensional space. While these networks maintain adequate modelling capabilities using a minimum number of adaptable parameters, the refinement procedure is often restricted to be carried out *off-line*.

2.2 Instantaneous gradient descent learning

The weight adjustment procedure for the LAN utilizes *on-line* optimization techniques, such as the least-mean-square (LMS) and the normalized LMS (NLMS) method. Both methods utilize an instantaneous gradient estimate to adjust the weight vector (**24, 25**). In general when these methods are used, the weight vector will converge to an optimal or near an optimal solution ($\mathbf{w}^*$)† if the cost performance surface in the weight space ($\mathbf{w}^p$) has only one global minimum.

From equation (1), these networks share an identical structure when the model output is linear with respect to its weight vector. The cost performance surface of these networks is therefore quadratic, which partially justifies the use of these methods. It is also required that the set of transformed input vectors be minimally correlated, which in turn depends on the training inputs and

† This optimal solution is often referred to as Wiener–Hopf solution.

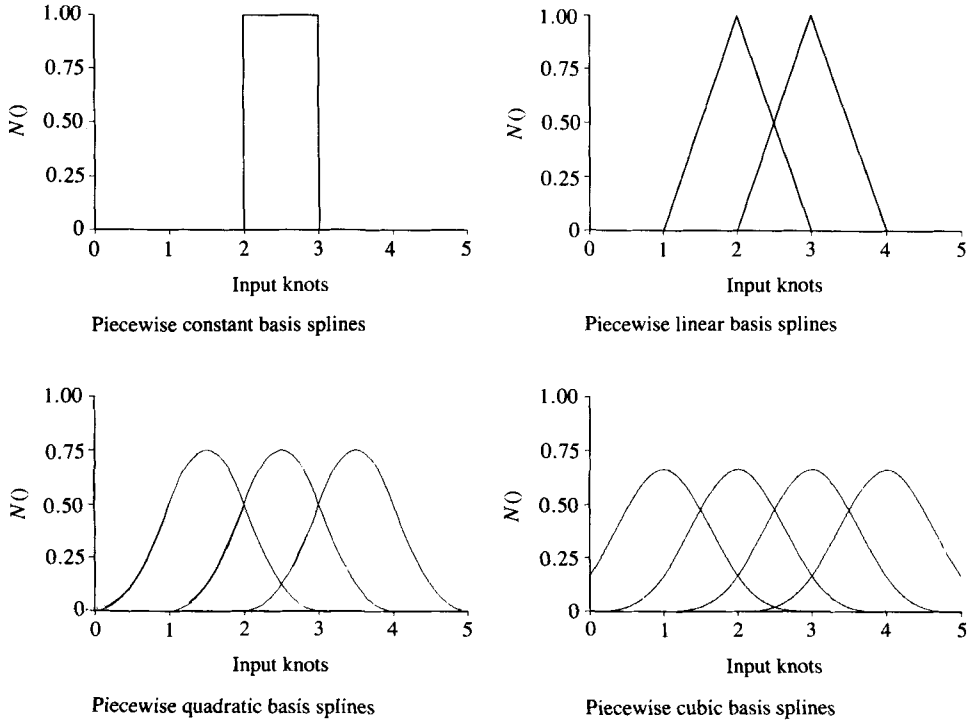


Fig. 6 B-splines univariate field shapes

the basis function shapes. The cost performance index is typically based on an instantaneous mean square output error $[\varepsilon^2\{\boldsymbol{x}(t)\}]$, and is given as follows:

$$J\{\boldsymbol{x}(t)\} = \tfrac{1}{2}\varepsilon^2\{\boldsymbol{x}(t)\} \tag{8}$$

where $\varepsilon\{\boldsymbol{x}(t)\} = y\{\boldsymbol{x}(t)\} - \hat{y}\{\boldsymbol{x}(t)\}$, $y\{\boldsymbol{x}(t)\}$ is the desired output and $\hat{y}\{\boldsymbol{x}(t)\}$ is the network output for an input $\boldsymbol{x}(t)$. The instantaneous gradient estimate is given by

$$\frac{\partial J\{\boldsymbol{x}(t)\}}{\partial \boldsymbol{w}(t-1)} = -\varepsilon\{\boldsymbol{x}(t)\}\boldsymbol{\phi}\{\boldsymbol{x}(t)\} \tag{9}$$

The LMS method then adjusts the weight vector in the direction of the transformed input vector

$$\Delta \boldsymbol{w}(t-1) = \boldsymbol{w}(t) - \boldsymbol{w}(t-1) = \beta\varepsilon\{\boldsymbol{x}(t)\}\boldsymbol{\phi}\{\boldsymbol{x}(t)\} \tag{10}$$

where β is the learning rate. Because the transformed input vector is sparse, the computational complexity in each adjustment cycle is $O(\kappa)$. Assuming that the desired model is linear {or y is a linear combination of $\phi_k(\boldsymbol{x})$} and the sequence of the transformed input vectors is statistically independent, the LMS method ensures that the weight vector converges to $\boldsymbol{w}^*$ in the mean if the following condition is satisfied:

$$0 < \beta < \frac{2}{v_{\max}} \tag{11}$$

where $v_{\max}$ is defined as the largest eigenvalue of the ensemble-averaged transformed input correlation matrix $\mathbf{R}$.

$$\mathbf{R} = E\{\boldsymbol{\phi}(\boldsymbol{x})\boldsymbol{\phi}^T(\boldsymbol{x})\} \tag{12}$$

In practice, the entire set of training samples is not available. Thus, $\mathbf{R}$ and in turn its eigenvalues cannot be determined prior to learning. On the other hand, if the training inputs are stationary and sufficiently rich, good performance can usually be achieved with a small learning rate.

It can be seen that the LMS method adjusts $\boldsymbol{w}$ according to the magnitude of the transformed input vector equation (10), which is often undesirable. The NLMS method provides an improved weight adjustment technique which eliminates this dependency condition when setting β. The weight vector is then updated as follows:

$$\Delta \boldsymbol{w}(t-1) = \beta\varepsilon\{\boldsymbol{x}(t)\}\frac{\boldsymbol{\phi}\{\boldsymbol{x}(t)\}}{\| \boldsymbol{\phi}\{\boldsymbol{x}(t)\} \|^2} \tag{13}$$

where $\| \cdot \|$ is the common Euclidean norm. It should be noticed that the normalization factor in equation (13) for the binary CMAC is constant, which further simplifies the numerical computation. Both the NLMS and LMS learning rules are very similar in that three observations can be made. First, if the instantaneous mean square error is normalized by $\| \boldsymbol{\phi}(\boldsymbol{x}) \|^2$ in equation (8), the LMS method then becomes equivalent to the NLMS method. Second, if a new learning rate, defined as $\beta/\| \boldsymbol{\phi} \|^2$, is made to vary according to the magnitude of the transformed input vector, the NLMS method is then reduced to the standard LMS method. Finally, the expression in equation (13) is also equivalent to the LMS method if $\hat{y}$ and $\boldsymbol{\phi}(\boldsymbol{x})$ are each normalized by $\| \boldsymbol{\phi}(\boldsymbol{x}) \|$ in equations (8) and (9) (**26**).

It is well known that the sum of the p eigenvalues of any matrix is identical to the trace of the matrix. It can easily be seen that the sum for the normalized input correlation matrix $\bar{\mathbf{R}}$ is always equal to 1. Thus, the largest eigenvalue of $\bar{\mathbf{R}}$ must be less than or equal to 1. Based on the condition, equation (11), the modified convergence condition is now given as in equation (14) for the NLMS method. In addition to the established convergence properties using these learning rules, the rate of convergence is another important element which determines the effectiveness of the learning process. The convergence rate is primarily limited to the smallest positive eigenvalue of $\mathbf{R}$, and is generally slow if the inputs are highly correlated (or $\mathbf{R}$ is ill-conditioned) (**14, 24, 25**).

$$0 < \beta < 2 \tag{14}$$

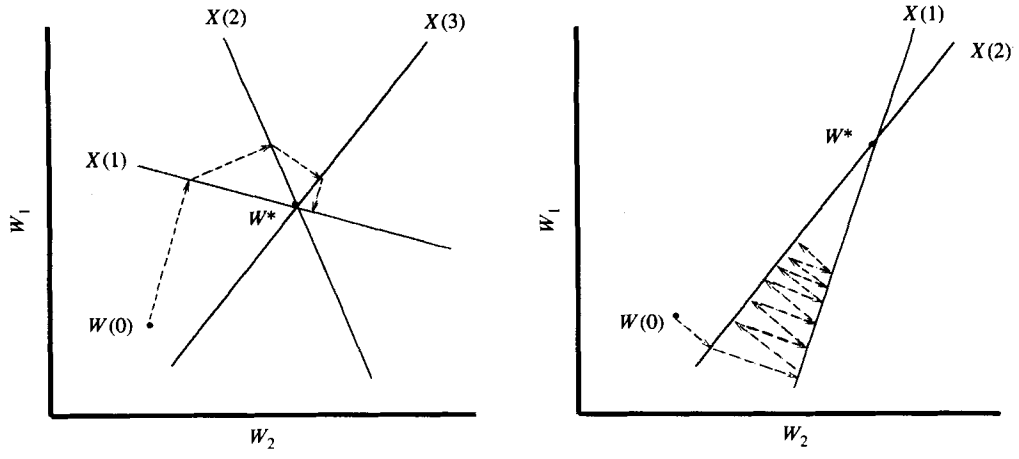
Both the LMS and the NLMS methods inherit an important convergence characteristic: their weight updates follow a 'minimum disturbance principle' (**26**). Geometrically each training sample forms a R^{p-1} hyperplane in the weight space, and the normal of the hyperplane is parallel to the associated transformed input vector. Given a sequence of training samples, the weights move along the normal of one hyperplane defined by one transformed input vector towards another one defined by a different transformed input vector, resulting in minimum changes in the weight magnitude approaching the hyperplanes (Fig. 7a). These methods are often referred to as orthogonal projection algorithms because the weight change is parallel to the normal of the hyperplane. Using the NLMS method and setting β to 1, the weight vector will drop onto the hyperplane in one step. If $0 \leqslant \beta \leqslant 1$, the weight vector will approach the hyperplane incrementally; if $1 \leqslant \beta \leqslant 2$, the weight vector will move past the hyperplane. A geometrical interpretation of slow convergence due to highly correlated inputs is depicted in Fig. 7b. If the desired model is linear and the set of training samples are noiseless and sufficiently exciting, the weight vector will eventually converge toward $\boldsymbol{w}^*$. However, when the desired model is noisy or nonlinear, the weight vector will converge to a capture zone around $\boldsymbol{w}^*$, of size depending on the modelling error and measurement noise (**27, 28**).

Given that the weight vector converges to a zone near $\boldsymbol{w}^*$, the minimum mean square output error might still be large when the modelling error and measurement noise are present. The misadjustment can often be reduced using a stochastic approximation technique in which the learning rate for w_i is made to vary as in equation (15) based on the frequency of its update. This allows the network to be plastic during the transient stage, but robust enough to filter out the measurement noise and the modelling error in the steady-state condition. However, this technique requires that the training data be stationary unless the learning rate is continually reset to a large value. The convergence properties of the binary CMAC network for various training rules have been established (**29**).

$$\sum_{i=0}^{\infty} \beta_i = \infty$$

$$\sum_{i=0}^{\infty} \beta_i^2 < \infty \tag{15}$$

During the transient learning, if the training inputs are separately distributed and most w_i's are unin-



(a) Well-conditioned case (b) Ill-conditioned case

Fig. 7 Weight convergence trajectories based on NLMS learning rule with $\beta = 1$

itialized, the network response can sometimes be weak even for a large κ. A heuristic scheme which increases the strength of the network response has been proposed for the CMAC network (**30**). In this scheme, a vector of size p is required to register the update status for each w_i. Using this information, the normalization factor is then redefined as the number of active $w_i(\hat{\kappa})$ which has been updated. The network response can thus be increased depending on $\hat{\kappa}$, which is generally less than κ. In a similar manner, this scheme can be generalized to the B-splines network.

3 RADIAL BASIS FUNCTIONS

The radial basis function network was originally employed as a numerical interpolation technique in a multi-dimensional space (**31**), and was later adopted as a one-hidden-layer feedforward network (**32**). An excellent review on this topic is given in (**12**). While the RBF network exhibits an associative memory characteristic similar to the LAN, they are different in their centre placements, the shape of their receptive fields and basis functions.

3.1 Model structure

The topology of the RBF network can be summarized as $n_0 - n_1 - n_2$, where n_0 is the network input dimension, n_1 is the number of basis functions (or hidden nodes) and n_2 is the network output dimension. Each node in the hidden layer of an RBF network has a radially symmetric response around a node parameter vector called its centre. Similar to the LAN, the output node of the RBF network is formed from a linear combination of these basis functions.

Given a network input vector $\boldsymbol{x} = [x_1 \cdots x_{n_0}]^T$, the outputs of hidden nodes are specified by

$$\phi_j = f\left(\frac{\| \boldsymbol{x} - \boldsymbol{c}_j \|}{\rho_j}\right) \qquad 1 \leqslant j \leqslant n_1 \tag{16}$$

where $\| \cdot \|$ denotes the Euclidean norm, ρ_j are positive scalars known as the widths of the basis functions (notice that these are defined differently from those in the LAN), $\boldsymbol{c}_j = [c_{j,1} \cdots c_{j,n_0}]^T$ are the RBF centres and $f(\cdot)$ is a non-linear function from R^+ to R, which is referred to as the non-linearity of hidden nodes. The output nodes are defined by

$$\hat{y}_i = f_{r,i}(\boldsymbol{x}) = \sum_{j=1}^{n_1} w_{i,j}\phi_j, \; 1 \leqslant i \leqslant n_2 \tag{17}$$

where $w_{i,j}$ are the weights connecting the ith output node to the jth hidden node. The overall response of the RBF network realizes a mapping f_r: $R^{n_0} \rightarrow R^{n_2}$.

There are a variety of choices for the node non-linearity $f(\cdot)$. Typical examples are the Gaussian function (Fig. 4c)

$$f(z) = \exp\left(\frac{-z^2}{\rho}\right) \tag{18}$$

and the thin-plate-spline function

$$f(z) = z^2 \log(z) \tag{19}$$

An alternative 'Gaussian bar' basis function which responds to a more localized input region than the sigmoid function, but responds to a less localized region than the Gaussian function, has recently been proposed (**33**). The multi-dimensional basis function is formed using a weighted summation operator, and can generally provide better approximation than the Gaussian function when the training set is small and the inputs are highly *redundant*.

The widths of the basis functions can be treated as free parameters, and each width can be assigned to a different value. Alternatively, all the widths can be fixed to a same value ρ, although some choices of $f(\cdot)$ such as equation (19) do not require such a parameter.

In general, there are many ways to distribute the hidden node centres in R^{n_0}. Traditionally, one centre is placed at every training sample provided that the data set contains a small number of noiseless training examples. Interpolation can thus be carried out using standard least-square techniques.* When the data set is large and the training data are noisy, the number of basis functions is often chosen to be less than the size of the data set and the centres are placed at selected training inputs in order to minimize the overfitting due to noise. Other methods, such as distributing the centres

* Or singular valued decomposition techniques when the training data are inconsistent.

on a lattice (**34**) or at random locations in R^{n_0}, have also been adopted. When the initial centre placement is not optimal, the centre placement can be adapted using on-line gradient descent or κ-means clustering algorithms (**12, 35**).

The RBF network is a general function approximator, and its performance does not depend critically on the choice of $f(\cdot)$. Moreover, even when all the widths are fixed to a same value, the RBF network with a sufficient number of hidden nodes is still capable of uniformly approximating any continuous function. Theoretical investigation on the approximation capabilities of the RBF network can be found in (**2, 36**).

3.2 Recursive learning

By adopting ideas from non-linear system identification (**37–39**), the following recursive prediction error (RPE) algorithm is a general non-linear learning method which can readily be applied to the RBF network (**40, 41**). For the general single-output case with p adaptable parameters, the RPE algorithm takes the form

$$\varepsilon(t) = y(t) - \hat{y}(t)$$

$$\boldsymbol{\psi}(t, \boldsymbol{\theta}) = \boldsymbol{\psi}(t) = \frac{\partial \hat{y}(t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

$$\mathbf{P}(t) = \frac{1}{\lambda}\left\{\mathbf{P}(t-1) - \frac{\mathbf{P}(t-1)\boldsymbol{\psi}(t)\boldsymbol{\psi}^T(t)\mathbf{P}(t-1)}{\lambda + \boldsymbol{\psi}^T(t)\mathbf{P}(t-1)\boldsymbol{\psi}(t)}\right\}$$

$$\boldsymbol{\theta}(t) = \boldsymbol{\theta}(t-1) + \mathbf{P}(t)\boldsymbol{\psi}(t)\varepsilon(t) \tag{20}$$

where $\boldsymbol{\theta}(t)$ is a p-dimensional parameter vector and $\boldsymbol{\psi}(t)$ is the gradient vector of $\hat{y}(t)$ with respect to $\boldsymbol{\theta}(t)$. $\mathbf{P}(t)$ can be interpreted as the time-average inverse-input correlation matrix, and λ is the forgetting factor. The RPE algorithm has a similar form to the recursive least-squares (RLS) algorithm and degenerates into the latter when the network model is linear. The computational complexity of the RPE algorithm is $O(p^2)$. The multi-output version of the RPE algorithm can be found in (**42**). The total number of adaptable parameters in the RBF network is

$$p = (n_0 + 1)n_1 + n_1 n_2 \tag{21}$$

If the size of θ is large, on-line numerical computation can still be achieved by using a local learning version of the RPE algorithm which is known as the parallel RPE (PRPE) algorithm (**40, 41**). In this distributed learning procedure, each node performs its own RPE algorithm simultaneously. For the generic ith node in the kth layer, local learning is achieved using

$$\boldsymbol{\psi}_{k,i}(t) = \frac{\partial \hat{y}(t)}{\partial \boldsymbol{\theta}_{k,i}}$$

$$\mathbf{P}_{k,i}(t) = \frac{1}{\lambda}\left\{\mathbf{P}_{k,i}(t-1) - \frac{\mathbf{P}_{k,i}(t-1)\boldsymbol{\psi}_{k,i}(t)\boldsymbol{\psi}_{k,i}^T(t)\mathbf{P}_{k,i}(t-1)}{\lambda + \boldsymbol{\psi}_{k,i}^T(t)\mathbf{P}_{k,i}(t-1)\boldsymbol{\psi}_{k,i}(t)}\right\}$$

$$\boldsymbol{\theta}_{k,i}(t) = \boldsymbol{\theta}_{k,i}(t-1) + \mathbf{P}_{k,i}(t)\boldsymbol{\psi}_{k,i}(t)\varepsilon(t) \tag{22}$$

where $\boldsymbol{\theta}_{k,i}$ is the parameter vector of the ith node in the kth layer. The computational complexity of the PRPE algorithm for the RBF network is $O(p')$, where p' ($p' \ll p^2$) is

$$p' = (n_0 + 1)^2 n_1 + n_1^2 n_2 \tag{23}$$

The PRPE algorithm for training the RBF network has the same form as equation (22) with $k = 1, 2$ and $1 \leqslant i \leqslant n_k$. Specifically, for the single-output node

$$\boldsymbol{\theta}_{2,1} = \boldsymbol{w} = [w_1 \cdots w_{n_1}]^T \tag{24}$$

$$\boldsymbol{\psi}_{2,1} = \frac{\partial \hat{y}}{\partial \boldsymbol{\theta}_{2,1}} = \boldsymbol{\phi} = [\phi_1 \cdots \phi_{n_1}]^T \tag{25}$$

Notice that the learning rule is identical to the RLS algorithm when the network model is linear. For the ith hidden node, $1 \leqslant i \leqslant n_1$

$$\boldsymbol{\theta}_{1,i} = [\rho_i c_{i,1} \cdots c_{i,n_0}]^T \tag{26}$$

$$\boldsymbol{\psi}_{1,i} = \frac{\partial \hat{y}}{\partial \boldsymbol{\theta}_{1,i}} = [\psi_{1,i,0}\, \psi_{1,i,1} \cdots \psi_{1,i,i_0}]^T \tag{27}$$

and, if the non-linearity is chosen to be equation (18),

$$\psi_{1,i,0} = w_i \exp\left(-\frac{\| \boldsymbol{x} - \boldsymbol{c}_i \|^2}{\rho_i}\right)\left(\frac{\| \boldsymbol{x} - \boldsymbol{c}_i \|^2}{\rho_i^2}\right) \tag{28}$$

$$\psi_{1,i,j} = 2w_i \exp\left(-\frac{\| \boldsymbol{x} - \boldsymbol{c}_i \|^2}{\rho_i}\right)\left(\frac{\boldsymbol{x}_j - \boldsymbol{c}_{i,j}}{\rho_i}\right) \qquad 1 \leqslant j \leqslant n_0 \tag{29}$$

In order to avoid ρ_i becoming too small or too large, the widths are generally constrained to be $\rho_{\min} \leqslant \rho_i \leqslant \rho_{\max}$ during learning.

An alternative learning scheme can also be employed by which the RBF centres are adjusted using a recursive clustering algorithm and the weight vector $\boldsymbol{w}$ is adjusted using the RLS algorithm (**35, 42**). Instead of adjusting the centres using the output error, this clustering algorithm first finds a centre that is nearest to the network input vector $\boldsymbol{x}(t)$ and then moves the centre closer to $\boldsymbol{x}(t)$. The algorithm can be described as follows. Let

$$d_i(t) = \| \boldsymbol{x}(t) - \boldsymbol{c}_i(t-1) \| \qquad 1 \leqslant i \leqslant n_1 \tag{30}$$

and

$$d_k(t) = \min\{d_1(t), \ldots, d_{n_1}(t)\} \tag{31}$$

then

$$\boldsymbol{c}_k(t) = \boldsymbol{c}_k(t-1) + \alpha\{\boldsymbol{x}(t) - \boldsymbol{c}_k(t-1)\},$$
$$\boldsymbol{c}_i(t) = \boldsymbol{c}_i(t-1) \qquad 1 \leqslant i \leqslant n_1 \text{ and } i \neq k \tag{32}$$

where the learning rate α lies in the interval (0, 1) and is slowly decreasing. This learning rule for adjusting centres has its root in the κ-means clustering method (**43**), and is similar to the Kohonen's projection scheme (**44**). Since the distances $d_i(t)$ are needed in computing the network response in equation (17), this clustering algorithm thus requires a minimal computational overhead. The recursive clustering algorithm and the RLS algorithm are linear learning rules, which can often lead to faster convergence. On the other hand, parameter adaptation based only on the output error requires non-linear optimization, and often leads to sub-optimal modelling performances. Nevertheless, it is important to notice that the clustering technique is effective only if the density of the training data is correlated with the

steepness of the training function: more samples are gathered where the function varies significantly, and fewer samples are gathered where the function is smooth.

Besides the adaptable centres, the basis function widths can also be adjusted based on gradient methods or nearest-neighbour heuristics (**35**). In addition to the network output error, a higher order network derivative can also be incorporated in the cost function to adjust the parameters so that the approximation is constrained to be smooth (**12, 45**).

4 MULTI-LAYERED PERCEPTRON

In a MLP, all the nodes in one layer are fully connected to the nodes in its adjacent layers, but there is no connection between the nodes within the same layer and no bridging layer connections. By partitioning the input space using a set of hyperplanes, the MLP is particularly well suited to high-dimensional classification tasks (**46**).

4.1 Model structure

The topological structure of the MLP is depicted in Fig. 8. Inputs to the network are passed to each node in the first layer. The outputs of the first-layer nodes then become inputs to the second layer and so on. The last layer acts as the network output layer and all the preceding layers are called *hidden* layers. The architecture of an MLP can be summarized as $n_0 - n_1 - \cdot - n_l$, where n_0 is the network input dimension, l is the number of layers and $n_i (1 \leqslant i \leqslant l)$ is the numbers of nodes in the ith layer.

The input–output relationship of the ith node in the kth layer is defined by

$$x_{k,i} = f\left(\sum_{j=1}^{n_{k-1}} w_{k,i,j} x_{k-1,j} + \mu_{k,i}\right) \tag{33}$$

where $w_{k,i,j}$ are the node connection weights, $\mu_{k,i}$ is the node threshold, $f(\cdot)$ is the node activation function and $x_{0,j}$ denote the network inputs. Based on this relationship, each hidden node partitions the input space with its hyperplane along which its activation function has a constant output value. Two typical activation functions are the sigmoid function (10) (Fig. 4d) and the hyperbolic tangent function (34).

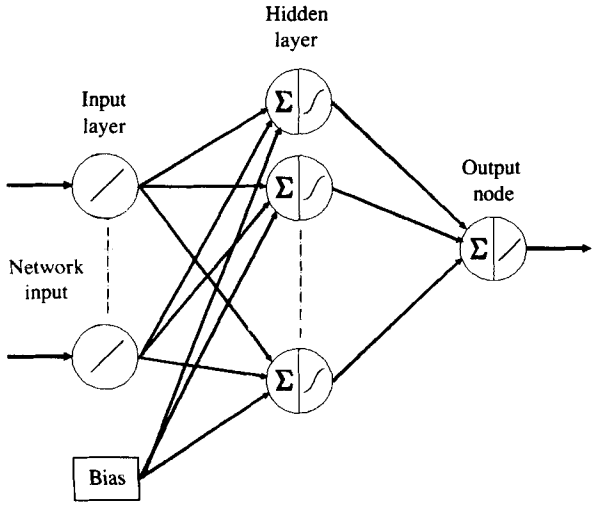


Fig. 8 Multi-layered perceptron network with one hidden layer

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{34}$$

$$f(x) = \tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)} \tag{35}$$

For the purpose of system modelling, the output nodes usually do not contain a threshold parameter and the associated activation functions are linear, that is

$$x_{l,i} = \sum_{j=1}^{n_{l-1}} w_{l,i,j} x_{l-1,j} \qquad 1 \leqslant i \leqslant n_l \tag{36}$$

The overall response of the network realizes a mapping $f_m: R^{n_0} \rightarrow R^{n_l}$.

The MLP is a general function approximator, and a one-hidden-layer network is sufficient to represent any arbitrary continuous function provided that there are a sufficient number of hidden nodes in the network. For many practical problems, networks with two or more hidden layers may be more efficient in terms of the total hidden nodes required. The theoretical modelling capabilities of the MLP have extensively been investigated, for example (**1, 47, 48**).

4.2 Recursive learning

Recursive learning algorithms for the MLP are generally based on gradient-type techniques (**13, 40, 41**). For notational simplicity, the single-output case ($n_l = 1$) is again described. However, the method is readily applicable to the general multi-output case. Introducing the network input vector at sample t as

$$\boldsymbol{x}(t) = [x_{0,1}(t) \ldots, x_{0,n_0}(t)]^{\mathrm{T}} \tag{37}$$

and collecting all the weights and thresholds of the MLP into a p-dimensional vector $\boldsymbol{\theta}$, where

$$p = \sum_{i=0}^{l-2} (n_i + 1) n_{i+1} + n_{l-1} n_l \tag{38}$$

Then the overall network output can concisely be written as

$$\hat{y}(t, \boldsymbol{\theta}) = x_{l,1}(t) = f_m\{\boldsymbol{x}(t), \boldsymbol{\theta}\} \tag{39}$$

The well-known backpropagation (BP) algorithm (**13**) can be considered as a special case of equation (20) by replacing $\mathbf{P}(t)$ with the identity matrix and by using a smoothed version of $\boldsymbol{\psi}(t)$. The BP algorithm thus takes the form

$$\begin{aligned} \varepsilon(t) &= y(t) - \hat{y}(t) \\ \bar{\boldsymbol{\psi}}(t) &= \beta \bar{\boldsymbol{\psi}}(t-1) + \alpha \boldsymbol{\psi}(t) \varepsilon(t) \\ \boldsymbol{\theta}(t) &= \boldsymbol{\theta}(t-1) + \bar{\boldsymbol{\psi}}(t) \end{aligned} \tag{40}$$

where β and α are the momentum and adaptive gain respectively. It is straightforward to rewrite equation (40) into the usual form of the BP algorithm given in (**13**). The computational complexity of the BP algorithm is $O(p)$, where p is defined in equation (38). Alternatively, the PRPE algorithm can be used. Let $\boldsymbol{\psi}_{k,i}$ be the gradient of $\hat{y}(t)$ with respect to $\boldsymbol{\theta}_{k,i}$, where $\boldsymbol{\theta}_{k,i}$ is defined as

$$\boldsymbol{\theta}_{k,i} = [\mu_{k,i} w_{k,i,1} \cdots w_{k,i,n_{k-1}}]^{\mathrm{T}} \tag{41}$$

The PRPE algorithm for this node has the same form as in equation (22). The computational complexity of the PRPE algorithm for the MLP is $O(p')$, where p' is

$$p' = \sum_{i=0}^{l-2} (n_i + 1)^2 n_{i+1} + n_{l-1}^2 n_l \tag{42}$$

Although the PRPE algorithm is more complex than the BP algorithm ($p < p'$), the former method generally provides much better convergence properties. The gradients $\psi_{k,i}$ for an $n_0 - n_1 - n_2 - (n_3 = 1)$ MLP are given in the Appendix.

5 TWO-DIMENSIONAL NON-LINEAR TIME SERIES MODELLING

A simulated non-linear time series was employed to test the comparative modelling capabilities of the previously described networks. The time series is described by the following second-order non-linear difference equation:

$$\begin{aligned} y(t) = {} & [0.8 - 0.5 \exp\{-y^2(t-1)\}]y(t-1) \\ & -[0.3 + 0.9 \exp\{-y^2(t-1)\}]y(t-2) \\ & + 0.1 \sin\{\pi y(t-1)\} + e(t) \end{aligned} \tag{43}$$

where $e(t)$ is an additive Gaussian white noise sequence with zero mean and variance 0.01. If the function which defines the system non-linearity of the time series equation (43) is denoted as $f_s(\cdot)$, the difference equation can be expressed concisely as

$$y(t) = f_s\{y(t-1), y(t-2)\} + e(t) \tag{44}$$

A three-dimensional plot of $f_s(\cdot)$ is given in Fig. 9. Note that $f_s(\cdot)$ is smooth and is linear with respect to the $y(t-2)$.

In order to train the four network models, 1000 iterated noisy time series samples were generated from an initial condition $y(-1) = y(0) = 0$, and were used as an identification set. A two-dimensional phase plot of the identification set is shown in Fig. 10a, in which $y(t)$ is plotted against $y(t-1)$. In order to test the fitness of these network models, another 1000 iterated noiseless time series samples $\{y_d(t)\}$ were generated from an initial condition $y_d(-1) = y_d(0) = 0.1$, and were used as a validation set. Figure 10b shows a two-dimensional phase plot of the validation set. Note that the time

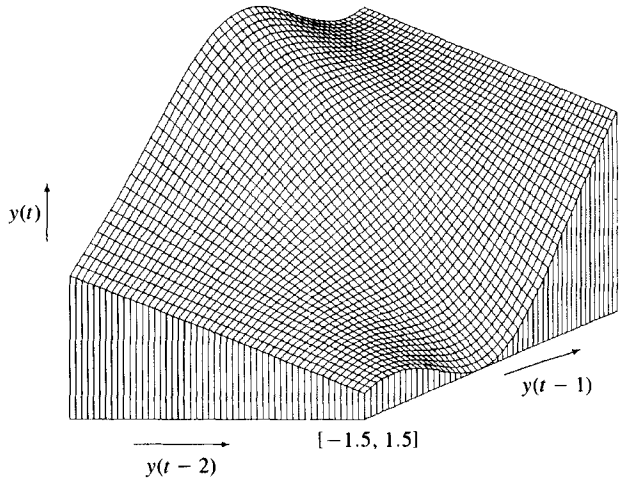


Fig. 9 Deterministic time series surface. Input region: $[-1.5, 1.5] \times [-1.5, 1.5]$. Surface height range: $[-1.872, 1.872]$

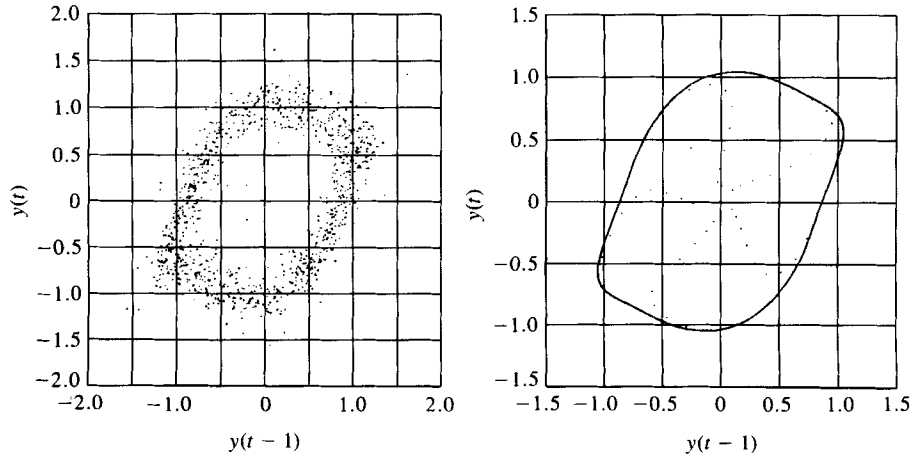
series sequence was bounded within the input region $[-1.5, 1.5] \times [-1.5, 1.5]$. This forms a necessary input domain on which the LAN can be employed to model the time series dynamics.

It can be seen from the phase plot that the underlying dynamics of the simulated system, equation (43), has a stable limit cycle and an unstable origin. Any slight perturbation near the origin will result in the iterated series diverging toward the limit cycle in a form of spiral arms. Based on this characteristic, the gathered training samples are thus very sparse near the origin, which can be used to test the abilities of these networks to generalize.

Figure 11 shows the first 60 time history samples in the validation set. Similar to an amplitude modulation process, the time history has a fast cycle which repeats every five samples, and a slow cycle which repeats roughly every 30 samples. The fast cycle can also be observed in the spiral arms shown in the phase plot of the validation set.

5.1 Modelling evaluation

The four networks are trained on the identification set consisting of iterated noisy time series samples. The



(a) Noisy identification set $x(0) = [0, 0]$ (b) Noiseless validation set $x(0) = [0.1, 0.1]$

Fig. 10 Non-linear time series phase plots

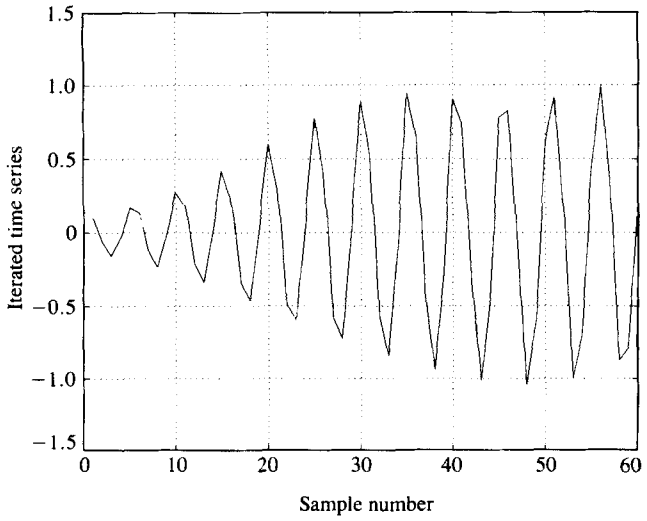


Fig. 11 A noiseless time history plot: $x(0) = [0.1, 0.1]$

network input vector at sample t is given by $x(t) = [y(t-1) \quad y(t-2)]^T$, and the corresponding desired output is $y(t)$. The fitted network models are then subject to the following performance measures which can be used to evaluate the transient convergence properties as well as the steady-state modelling abilities.

Surface plot

If the network model is adequate, the network must be able to reconstruct $f_s(\cdot)$ accurately within the input region of interest. Surface plots are thus generated over the region $[-1.5, 1.5] \times [-1.5, 1.5]$.

Normalized prediction error autocorrelation

A normalized autocorrelation function of prediction errors $\varepsilon(t+1\,|\,t)$ can be used to evaluate the approximation ability (or one-step ahead prediction ability) over the identification set, and is computed as follows:

$$C(k) = \frac{\sum_{t=1+k}^{1000} \{\varepsilon(t)\varepsilon(t-k)\}}{\sum_{t=1}^{1000} \{\varepsilon(t)\varepsilon(t)\}} \tag{45}$$

where k is the time lag and $\varepsilon(t)$ is the one-step-ahead prediction error at sample t over the identification set. Note that the normalization procedure allows the autocorrelation analysis to be independent of the magnitude of the prediction errors. In general, correlations between time-shifted prediction errors are considered insignificant if they lie within the confidence limits of $\pm 1.96/\sqrt{(N)}$, where N is the number of training samples in the identification set (**49**). The limits in this case are ± 6.2 per cent. It should be emphasized that the one-step prediction errors being uncorrelated is only a necessary condition for an adequate network model. Sufficient conditions to guarantee the adequateness of the model require more complex tests (**49**).

Phase plot

While the error autocorrelation function can generally determine the one-step-ahead modelling abilities of these networks, the dynamics modelling of the process requires additional measures. For example, it is important that the network is able to reconstruct the dynamics of the origin and the limit cycle. Let $\hat{f}_s(\cdot)$ be the fitted network model. $\hat{f}_s(\cdot)$ is then used to generate the network outputs iteratively as in (46). Notice that no time series observation is involved when the network outputs are being generated. A phase plot of $\hat{y}(t)$ against $\hat{y}(t-1)$ is used to evaluate graphically the dynamics of the fitted network models. The initial condition is $\hat{y}(-1) = \hat{y}(0) = 0.1$.

$$\hat{y}(t) = \hat{f}_s\{\hat{y}(t-1), \hat{y}(t-2)\} \tag{46}$$

Time history plot

In addition to the phase plots, the time history plots are also used to evaluate graphically the dynamic modelling abilities of these networks. Both the network and the deterministic time series are given the same initial condition of $\hat{y}(-1) = 1.0$, $\hat{y}(-2) = 0.5$, and are iterated 60 times independently of each other. In general, the prediction error increases with the number of iterations when the network model is not exact. The time history plot can thus be used to determine the prediction horizon under which the network can accurately model the time series dynamics.

k-step-ahead prediction error variance

This provides a numerical evaluation of the multi-step-ahead modelling performance. The k-step-ahead prediction is computed as follows:

$$\hat{y}(t+k\,|\,t) = \hat{f}_s\{\hat{y}(t+k-1\,|\,t), \hat{y}(t+k-2\,|\,t)\} \tag{47}$$

where $k \geqslant 1$ and, if $k - j \leqslant 0$, $\hat{y}(t+k-j\,|\,t) = y(t+k-j)$ are time series observations. The k-step-ahead prediction error is defined as

$$\varepsilon(t+k\,|\,t) = y(t+k) - \hat{y}(t+k\,|\,t) \tag{48}$$

The prediction horizon is chosen to be $1 \leqslant k \leqslant 20$. The prediction accuracy is determined by the variance of $\varepsilon(t+k\,|\,t)$ over the validation set. Apart from a few initial transient points, the samples in the validation set all lie on the limit cycle. The variance over the limit cycle is thus evaluated based on the prediction errors using the last 900 samples in the validation set.

$$V(k) = \frac{\sum_{t=100+k}^{t=1000} \{\varepsilon(t+k\,|\,t)^2\}}{900 - k + 1} \tag{49}$$

Instantaneous learning curve

While the above measures focus on the steady-state modelling abilities, this measure evaluates the transient convergence properties of the learning procedures. All four networks require some forms of gradient estimate procedures in adjusting the network parameters. The convergence rate thus depends on the gradient estimate and the structure of the network model. A root-mean-square (RMS) of the one-step-ahead prediction errors over the validation set is thus computed after each training sample has been presented to the network. The RMS error generally decreases at a rate depending on the gradient noise magnitude and the sequence of the training samples. A biased training sequence is likely to deteriorate the gradient formation. The ordering of the

samples in the identification set is thus randomized so that the instantaneous learning curve is less sensitive to the training sequence.

5.2 Modelling based on CMAC

A CMAC network with piecewise linear univariate basis functions and the product operator was used to model the time series. The bounded input space of interest was chosen to be $[-1.5, 1.5] \times [-1.5, 1.5]$. Any sample lying outside this region was considered an outlier, and was excluded from training. Based on this consideration, there were a total of six outliers in the training set. S and ρ were both chosen to be 17. By equation (5), there were a total of 65 adaptable weights in the network. A use of such large field widths was based on the assumed prior knowledge that the function is smooth. The large ρ results in a small set of adaptable parameters, which improves the noise-filtering performance. Because the centre placement is modular (that is, the centre placement in any overlay is unchanged when the overlay is displaced ρ units parallel to any axis), the chosen displacement matrix can be described as

$$\begin{bmatrix} 0 & 0 \\ 1 & 7 \\ 2 & 14 \\ i & (7)\%\rho \\ \vdots & \vdots \\ 16 & 10 \end{bmatrix} \tag{50}$$

where % represents the modulus operator, as the overlay displacements are calculated using modulo ρ arithmetic.

In every training instance, the network output was normalized by a sum of ρ active basis function outputs. A stochastic NLMS (SANLMS) learning rule was used to adjust the weights, where the learning rate was defined as in equation (51). The initial learning rate (β_0) was chosen to be 1, the decaying rate constant (β_d) was chosen to be 30 and i is defined as the number of times that the parameter has been updated throughout training. The training was carried out incrementally in that only one sample was used at a time in adjusting the network weights. One thousand noisy samples were used in each cycle, and altogether 20 training cycles were carried out.

$$\beta = \frac{\beta_0}{1 + i/\beta_d} \tag{51}$$

After training, 60 weights were actually used in the identification process. The surface plot $\hat{f}_s(\cdot)$ is shown in Fig. 12. The error autocorrelation and the phase plot are shown in Fig. 13a and Fig. 14a respectively. The time history plot is given in Fig. 15a. The k-step-ahead prediction error variance is shown in Fig. 16, and finally the instantaneous learning curve is shown in Fig. 17a.

5.3 Modelling based on B-splines

Similar to the CMAC, only those samples within $[-1.5, 1.5] \times [-1.5, 1.5]$ were used in the identification process. $S_{y(t-2)}$ was chosen to be 1 and $S_{y(t-1)}$ was chosen to be 6. The univariate basis functions were

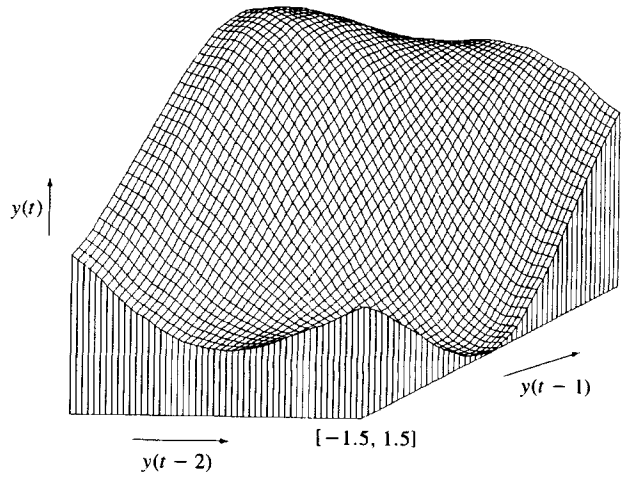


Fig. 12 CMAC surface reconstruction. Input region: $[-1.5, 1.5] \times [-1.5, 1.5]$. Surface height range: $[-1.272, 1.271]$

chosen to be piecewise linear along $y(t-2)$ and piecewise quadratic along $y(t-1)$. This choice was based on the assumed knowledge about the smoothness of $f_s(\cdot)$ and the linear relationship of $f_s(\cdot)$ with the $y(t-2)$ axis. It must be stressed clearly that the modelling performance will, of course, be degraded in the absence of such knowledge, as shown in (**50**) using the same time series. This particular model merely serves as an example to demonstrate the flexibility of the B-splines over the CMAC to incorporate prior knowledge. By equation (3), there were a total of 16 weights in the network. The displacement matrix was defined as in equation (6). The learning rule was again based on equation (51), in which β_0 was chosen to be 1 and β_d was chosen to be 50. Again, 20 training cycles were carried out.

After training, all 16 weights in the network were used in the identification process. The surface plot $\hat{f}_s(\cdot)$ is shown in Fig, 18. The error autocorrelation function and the phase plot are shown in Fig. 13b and Fig. 14b respectively. The time history plot is given in Fig. 15b. The k-step-ahead prediction error variance is shown in Fig. 16, and finally the instantaneous learning curve is shown in Fig. 17b.

5.4 Modelling based on RBF

A RBF network with ten hidden nodes and the Gaussian non-linearity, equation (18), was chosen to model the time series. By equation (21), there were a total of 40 adaptable parameters in the network. Initial weights were set to $w_i(0) = 0.0$, initial centres $c_i(0)$ were randomly chosen from the region $[-1.0, 1.0] \times [-1.0, 1.0]$ and initial widths were set to $\rho_i(0) = 2.0$. The PRPE algorithm was employed to fit this 2–10–1 RBF network. The forgetting factor was computed using the rule, equation (52), and initial matrices $\mathbf{P}_{k,i}(0) = 10.0\mathbf{I}_{k,i}$ where $\mathbf{I}_{k,i}$ are identity matrices of appropriate dimensions. Four training cycles were carried out. During learning, the widths were constrained to be $0.000001 \leqslant \rho_i(t) \leqslant 1000000.0$.

$$\lambda(t) = \lambda_0 \lambda(t-1) + 1 - \lambda_0$$
$$\lambda_0 = 0.99 \text{ and } \lambda(0) = 0.95 \tag{52}$$

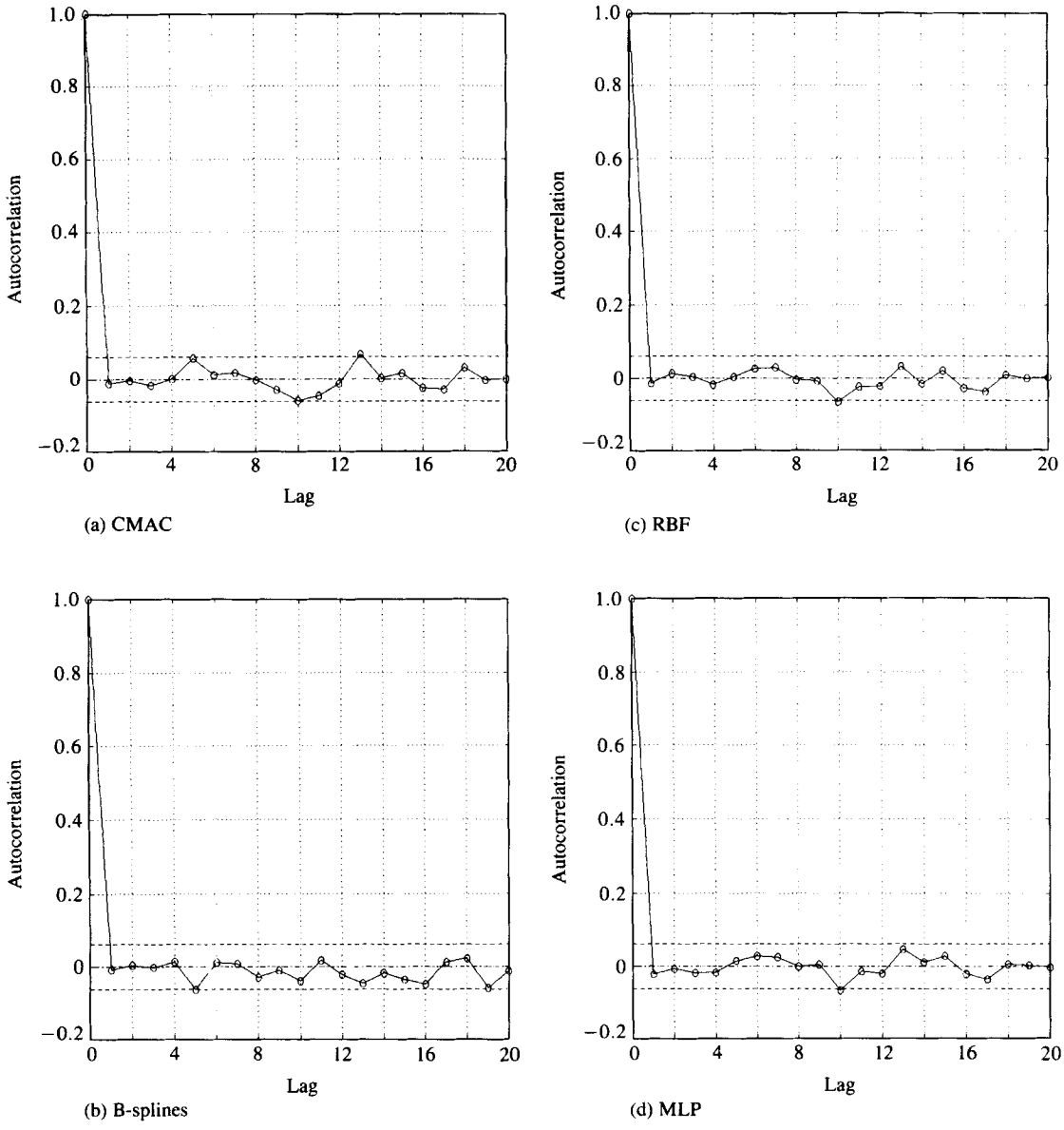


Fig. 13 Normalized error autocorrelations (lag from 0 to 20)

No further improvement in modelling was found after four training cycles. The surface plot $\hat{f}_s(\cdot)$ is shown in Fig. 19. The error autocorrelation function and the phase plot are shown in Fig. 13c and Fig. 14c respectively. The time history plot is given in Fig. 15c and the k-step-ahead prediction error variance is shown in Fig. 16. The instantaneous learning curve is shown in Fig. 17c. The centre placement is shown in Fig. 20, and finally the distribution of weights/widths is depicted in Fig. 21.

5.5 Modelling based on MLP

A two-layer perceptron was employed to model the time series. The structure of the network was defined by 2–16–1 and the activation function of hidden nodes was chosen to be equation (34). By equation (38), there were a total of 64 adaptable parameters in the network. Initial weights and thresholds were randomly set to values between -0.1 to 0.1. The PRPE algorithm was employed as the learning algorithm. The forgetting factor was computed according to equation (52), and the matrices $\mathbf{P}_{k,i}$ were initially set to $10.0\mathbf{I}_{k,i}$. Again, four training cycles were carried out.

After training, the network reached steady-state modelling condition. The surface plot $\hat{f}_s(\cdot)$ is shown in Fig. 22. The error autocorrelation function and the phase plot are shown in Fig. 13d and Fig. 14d respectively. The time history plot is given in Fig. 15d and the k-step-ahead prediction error variance is shown in Fig. 16. The instantaneous learning curve is shown in Fig. 17d, and finally the hyperplane placement in the input space is shown in Fig. 23.

6 DISCUSSION

The modelling results in the previous section suggest that all four networks are able to capture the essential time series dynamics accurately. The network surfaces were found to be similar within the limit cycle region, but were different near the outline of the region $[-1.5, 1.5] \times [-1.5, 1.5]$. As the training data were sparse, the

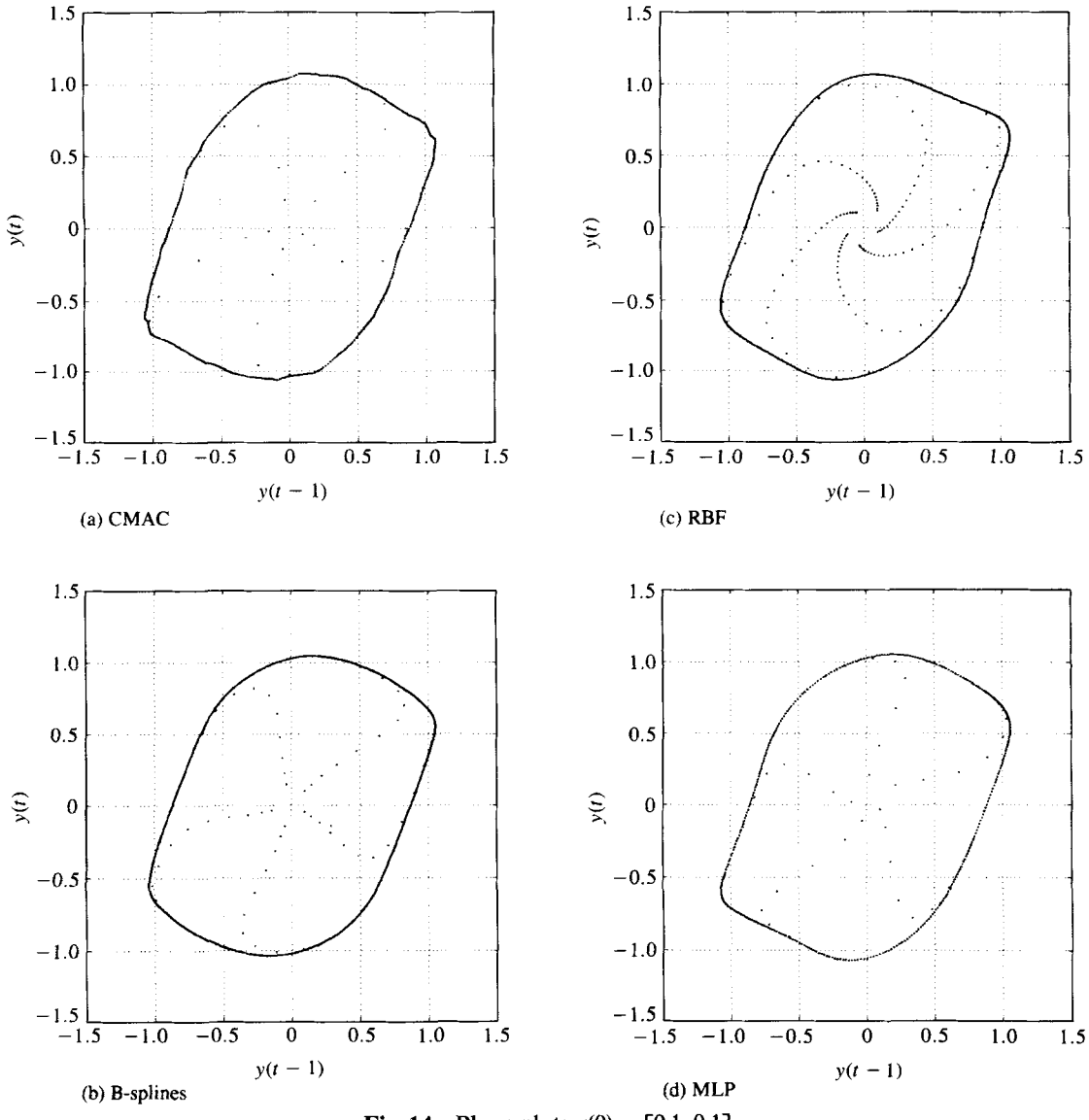


Fig. 14 Phase plots $x(0) = [0.1, 0.1]$

extrapolation characteristics near the outline are thus strongly determined by the shape and extent of their basis functions. This implies that the reconstructed limit cycle might go unstable for any network if the initial conditions are set near the edge of $[-1.5, 1.5] \times [-1.5, 1.5]$. Their error autocorrelation functions are also very similar, indicating that these networks have formed adequate one-step-ahead models for the time series. Their time history plots and k-step-ahead prediction variances indicate that these networks also have developed accurate iterated dynamics over the limit cycle region. Note that this was possible mainly because the initial conditions were set near the limit cycle. Of all the networks the RBF and MLP produced the smaller variances. That is, these networks have developed better short-range iterated dynamics over the limit cycle in the average sense.

When the initial conditions were set near the origin, the iterated dynamics was found to be quite different, as can be seen in the phase plots. The fitness of the spiral arm dynamics can be inferred graphically from its diverging rate (for example number of points along each arm) and its phase lag (or its orientation) with respect to the true spiral arm. Based on these characteristics, the B-splines network generated the best spiral arm. The spiral arms in the CMAC and MLP networks had slightly faster divergences, implying that their reconstructed origins were slightly less stable than the true one. On the other hand, the origin in the RBF network was found to be much stabler. This implies that the RBF surface near the origin was much flatter than the true one.

The B-splines network has incorporated the '$y(t-2)$ linearity and smoothness' prior knowledge about the time series into its model. This results in fewer adaptable weights and, in turn, a better noise-filtering condition. It must be stressed again that the use of such prior knowledge is merely used to demonstrate the flexibility of the model structure in the B-splines over the CMAC. Without such knowledge, the modelling performance of the B-splines will be similar to that of the CMAC.

The CMAC network has also incorporated the 'smoothness' knowledge into its model by using the globally extended receptive fields. However, the '$y(t-2)$ linearity' knowledge cannot be straightforwardly incorporated into this model because the receptive fields

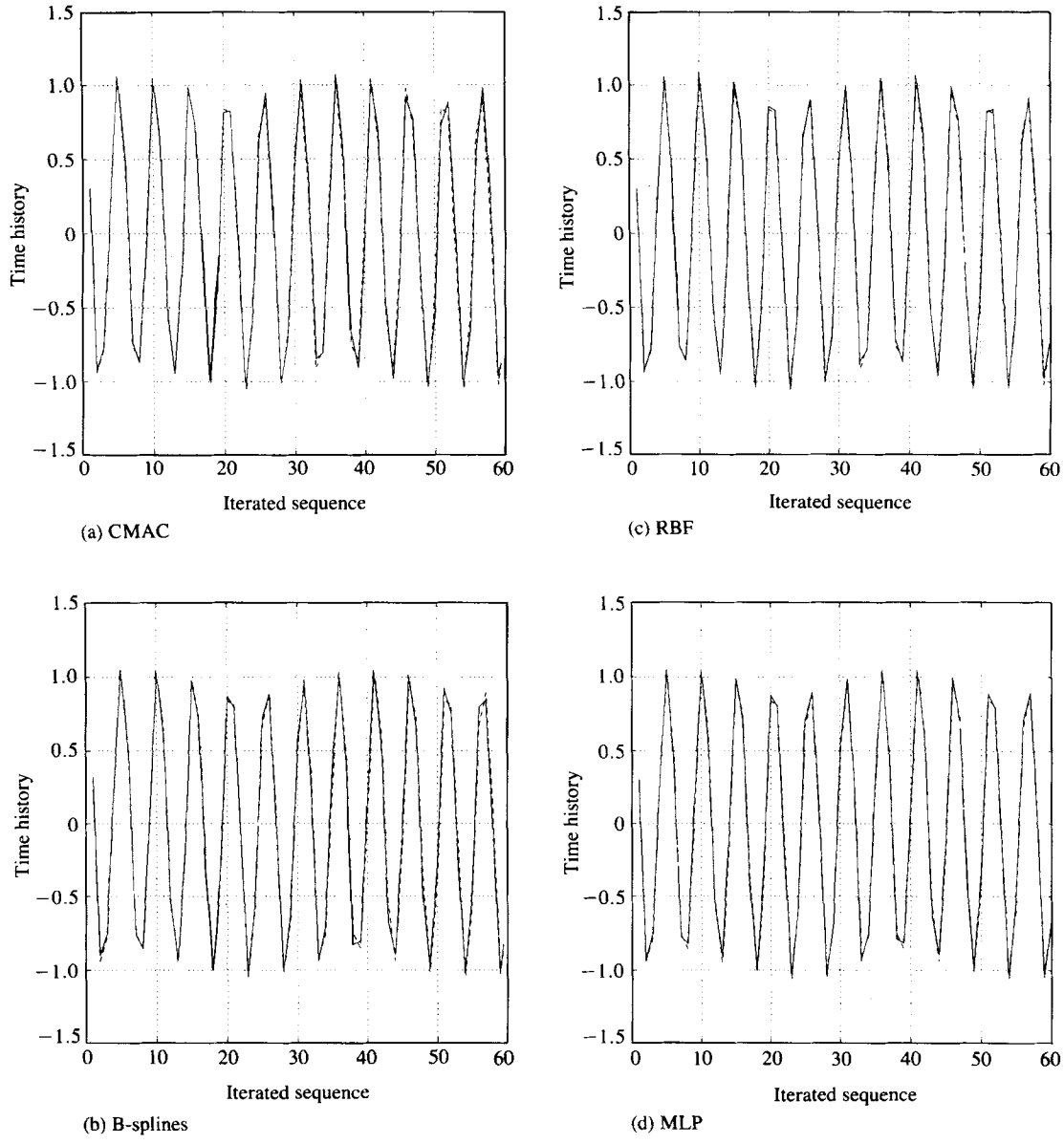


Fig. 15 Comparison of the iterated time history between the noiseless time series (solid) and each network output (dash) $\{x(0) = [1.0, 0.5]\}$

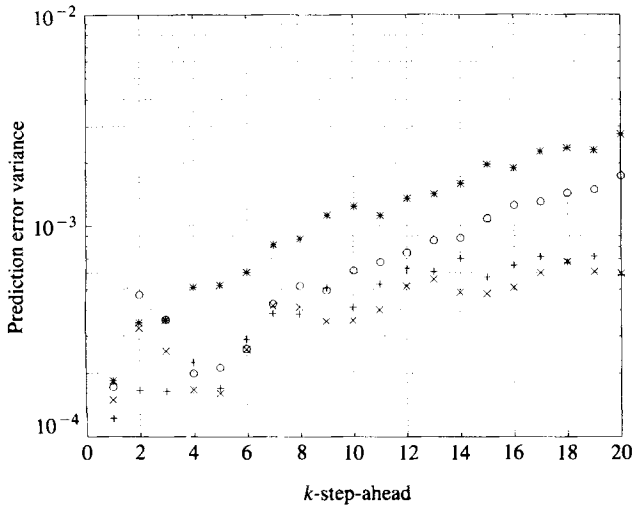


Fig. 16 k-step-ahead prediction error variances. $(0 \leqslant k \leqslant 20)$. (*) CMAC (○) B-splines (×) RBF (+) MLP

are only sparsely distributed. This thus brings up an important compromise between the two LAN model structures: while the CMAC can be employed for modelling in a much higher dimensional space, the existing placement scheme requires ρ to be the same in each axis, which then restricts the flexibility of the CMAC model.

Similar to the LAN, the RBF and MLP have also incorporated the 'smoothness' knowledge about $f_s(\cdot)$ into their models in order to constrain the networks from overfitting the noise. This was done by optimizing the number of nodes in these networks on a trial and error basis. However, because the RBF basis function responses are restricted to be radially symmetrical,* the '$y(t-2)$ linearity' knowledge cannot be incorporated. Nevertheless, it is possible that other forms of prior knowledge can be incorporated into the RBF model.

* Except the described Gaussian bar functions.

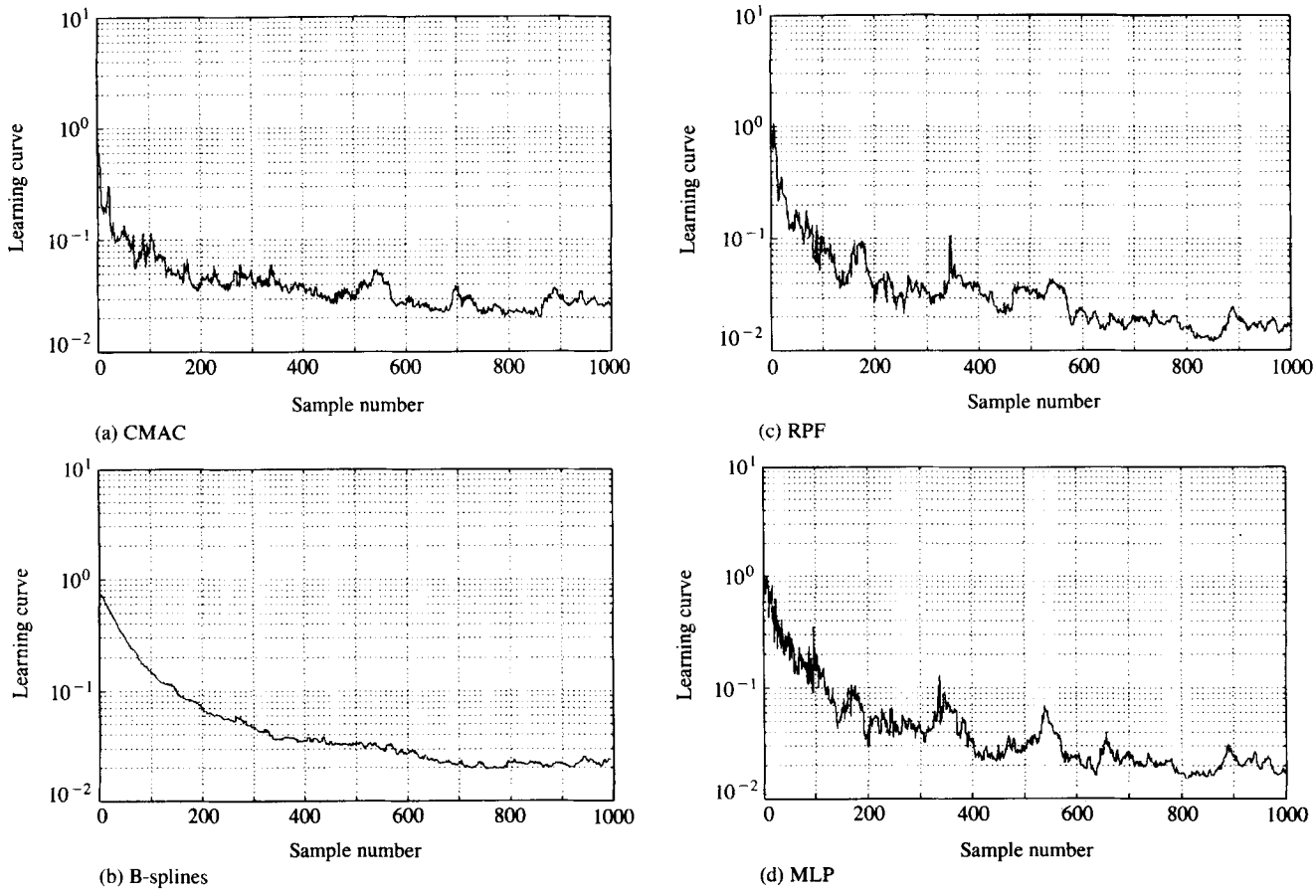


Fig. 17 Instantaneous learning curves in the first training cycle

One example is to place the RBF centres near the time series limit cycle.

With regard to the transient convergence characteristics, their instantaneous learning curves indicate that the one-step-ahead prediction errors were similar based on their decaying rates and their RMS errors at the end of the first training cycle. Among these curves, the one generated by the B-splines had the least jittering effect. This might be explained by the use of a relatively small p for modelling. A summary of the network parameters used in the time series modelling is given in Table 1.

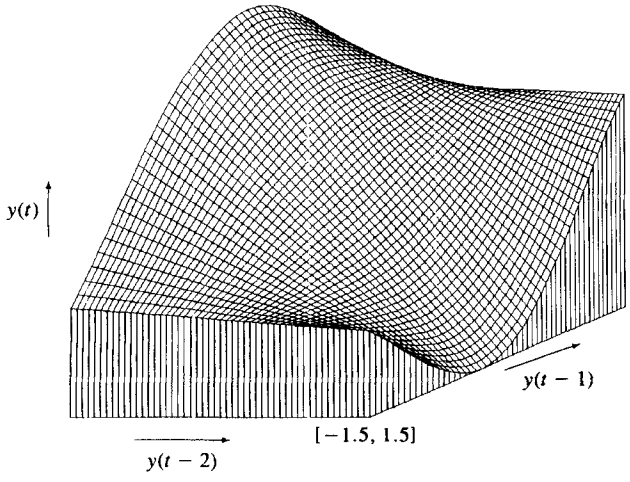


Fig. 18 B-splines surface reconstruction. Input region: $[-1.5, 1.5] \times [-1.5, 1.5]$. Surface height range: $[-1.864, 1.881]$

7 SUMMARY

The model structures and learning rules of the CMAC, B-splines, RBF and the MLP networks have been described. Their modelling abilities were compared using a two-dimensional non-linear noisy time series. The network performances were evaluated based on their surface reconstructions, normalized error autocorrelation characteristics, phase/time history plots, k-step-ahead prediction error variances and finally their instantaneous learning curves. The modelling results suggest that all four networks were able to capture the underlying dynamics of the iterated time series.

The LAN can generally achieve adequate convergence properties with simpler learning rules because their models are linear with respect to their adaptable weights. Also, their computational complexities in every training instance are on orders of $O(\kappa)$ where κ is a small number in most practice. Thus, they are well suited to applications with real-time constraints. The local generalization property also ensures that the learning interference across the network is restricted to be local. Between the two LAN models, the CMAC

Table 1

Network	Network size	Learning rule	Prior knowledge	Training cycles
CMAC	65	SANLMS	Smooth	20
B-splines	16	SANLMS	Linear/smooth	20
RBF	40	PRPE	Smooth	4
MLP	64	PRPE	Smooth	4

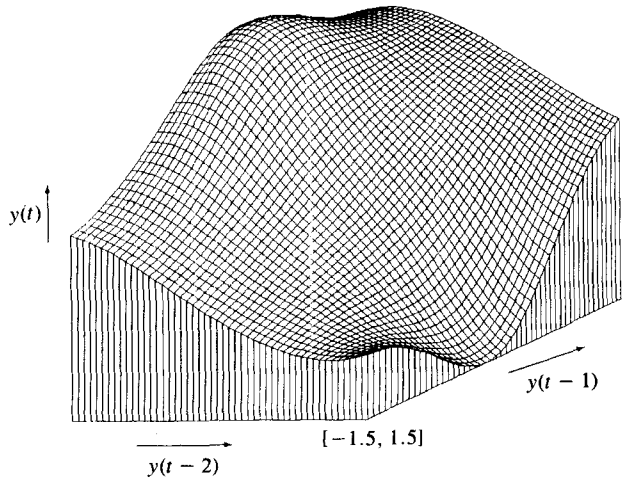


Fig. 19 RBF surface reconstruction. Input region: [−1.5, 1.5] × [−1.5, 1.5]. Surface height range: [−1.794, 1.428]

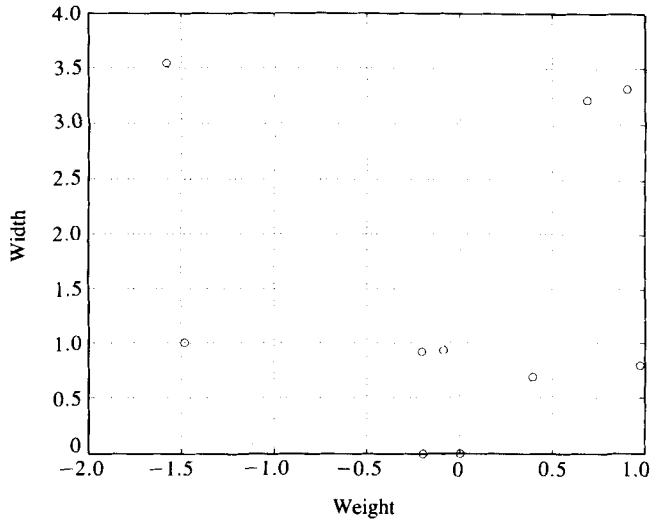


Fig. 21 RBF weight/width distribution after four training cycles

basis functions are sparsely distributed, resulting in a smaller p compared with the B-splines. In addition, the number of active weights in the CMAC is user defined, and is *not* dependent on n. Thus, the CMAC can be employed in applications of higher dimensionality relative to the B-splines. However, the existing placement scheme in the CMAC requires that ρ be the same in each axis, which thus restricts the flexibility of the CMAC compared with the B-splines, as was demonstrated in the time series example. Nevertheless, the sizes p of both networks are still exponentially dependent on n, which pose a computational constraint for high-dimensional modelling.

Unlike the LAN, the RBF and MLP models can be highly non-linear because their inner structures are generally adapted based on the commonly used output error. The convergence properties of these models are generally sensitive to the presentation order of the training data, and often result in undesired sub-optimal solutions, as can be seen in the placement of the RBF centres (Fig. 20). This means the adaptation based only on a single criterion might not be desirable when other effective criteria are available. For example, a better RBF centre placement for the same time series was

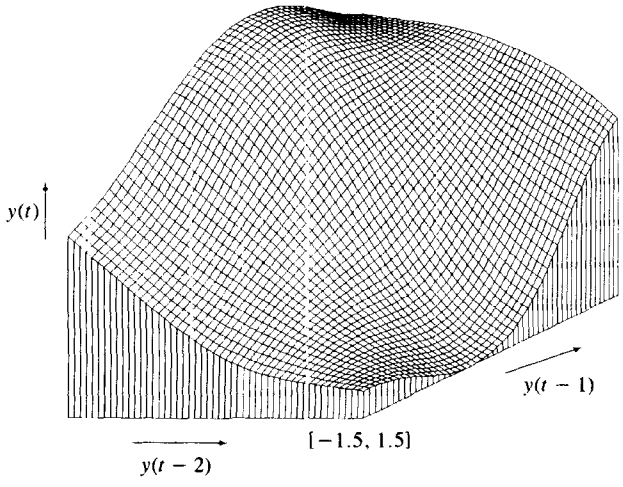


Fig. 22 MLP surface reconstruction. Input region: [−1.5, 1.5] × [−1.5, 1.5]. Surface height range: [−1.513, 1.482]

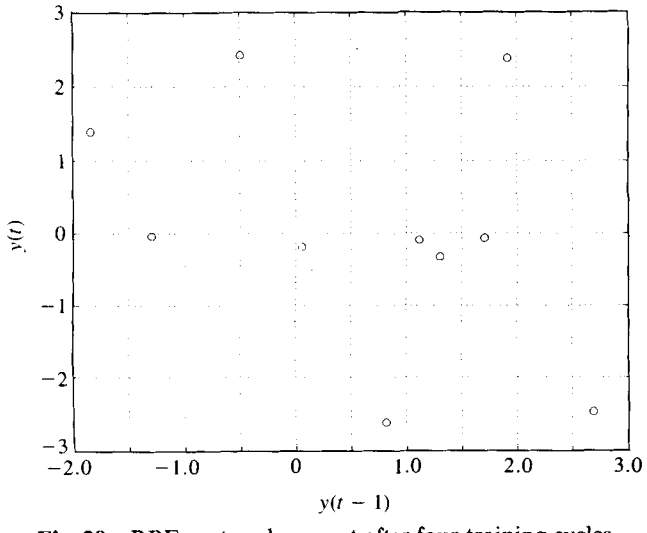


Fig. 20 RBF centre placement after four training cycles

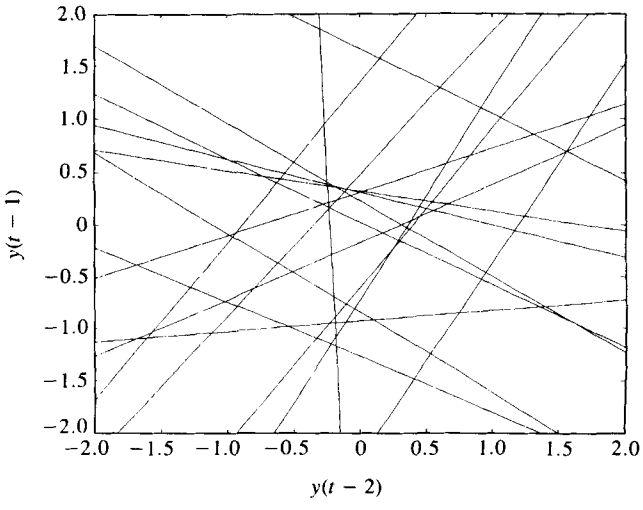


Fig. 23 MLP hyperplane formation after four training cycles

found when the thin-plate-spline non-linearity and the centre clustering algorithm were used (**41**). Note that, however, the sizes of these networks are generally not dependent on n because their basis functions are globally spanned. Thus, these networks are often better suited for fitting high-dimensional surfaces with sparse training data. The convergence rates for these models are generally slow because of the non-linear optimization procedures and maximum learning interferences.* Nevertheless, the convergence rates for these networks can be improved when a higher order gradient information is utilized for adaptation.

Knowing that the model selection can be influenced by the dimensionality of the problem domain, it is possible that the actual number of relevant inputs is unavailable. When the training inputs are strongly correlated or highly redundant, the modelling abilities are generally degraded as the network is employed in a much higher dimensional platform. In this situation, the non-linear models of the RBF† and MLP with adaptable inner structures can generally provide more efficient representations of the input–output relationship than the LAN of fixed inner structures‡.

Thus far, the modelling abilities are based on how accurately these networks interpolate between neighbouring training samples. In the untrained input region, the extrapolative abilities of these networks are solely determined by the shape and extent of their basis functions, and are generally unreliable for evaluating the modelling performance. Owing to the local definition of basis functions, the LAN embodies a 'do nothing' philosophy when an input is distant from the trained region, as opposed to a 'do something' philosophy in the MLP or a 'do little' philosophy in the Gaussian RBF. Notice that the 'do nothing' and 'do something' philosophies can be traced back to their initial network behaviours prior to learning. When these networks are applied to closed-loop control problems, the 'do nothing' philosophy is generally desirable based on the similar argument made for the linear adaptive model described in the introduction.

It has been found in various modelling and control literatures that certain types of network architectures are frequently chosen without having carefully justified their appropriateness for use in specific applications of interest. It is hoped that this paper can be used as an *unbiased* guideline of choosing a network model with appropriate *non-linearity*.

REFERENCES

1 **Cybenko, G.** Approximations by superpositions of a sigmoidal function. *Math. Control, Signals Syst.*, 1989, **2**(4), 303–314.

2 **Park, J.** and **Sandberg, I. W.** Universal approximation using radial-basis-function networks. *Neural Comp.*, 1991, **3**, 246–257.

3 **Handelman, D., Lane, S.,** and **Gelfand, J.** Integration of knowledge-based system and neural network techniques for robotic control. IFAC Workshop on *Artificial intelligence in real-time control*, September, 1988, Swansea, UK.

4 **Harris, C. J., Moore, C. G.** and **Brown, M.** *Intelligent control: some aspects of fuzzy logic and neural networks*, 1993 (World Scientific Press, London).

5 **Hunt, K., Sbarbaro, D., Zbikowski, R.** and **Gawthrop, P. J.** Neural networks for control systems—a survey. *Automatica*, 1992, **28**(6), 1083–1112.

6 **Miller, W. T., Glanz, F. H.** and **Kraft, L. G.** Application of a general learning algorithm to the control of robotic manipulators. *Int. J. Robotics Res.*, 1987, **6**(2), 84–98.

7 **Miller, W. T., Sutton, R. S.,** and **Werbos, P. J.** (Eds) *Neural networks for control* 1991 (MIT Press, Cambridge, MA).

8 **Tzirkel-Hancock, E.** and **Fallside, F.** Stable control of nonlinear systems using neural networks. Cambridge University Engineering Department, F-INFENG, Technical Report 81, 1991.

9 **An, P. E., Brown, M., Harris, C. J., Lawrence, A. J.** and **Moore, C. G.** Comparative aspects of associative memory networks for modelling. Accepted for *European control* conference, 1993, Groningen, The Netherlands.

10 **Miller, W. T., Glanz, F. H.** and **Kraft, L. G.** CMAC: An associative neural network alternative to backpropagation. *Proc. IEEE*, 1990, **78**, 1561–1567.

11 **Lippman, R. P.** An introduction to computing with neural nets. *IEEE ASSP Mag.*, April 1987.

12 **Poggio, T.,** and **Girosi, F.** Networks for approximation and learning. *Proc. IEEE*, 1990, **78**(9), 1481–1497.

13 **Rumelhart, D. E., Hinton, G. E.** and **Williams, R. J.** Learning internal representations by error propagation, In *Parallel distributed processing*: explorations in the microstructure of cognition (Eds. D. E. Rumelhart and J. L. McClelland), 1986, pp. 318–362 (MIT Press, Cambridge, MA).

14 **Brown, M.,** and **Harris, C. J.** *Neurofuzzy adaptive modelling and control*, 1993 (in press) (Prentice Hall, Englewood Cliffs, NJ).

15 **Albus, J.** A new approach to manipulator control: the cerebellar model articulation controller (CMAC). *Trans. ASME*, 1975, (**63**)9, 220–227.

16 **An, P. E., Miller, W. T.** and **Parks, P. C.** Design improvements in associative memories for cerebellar model articulation controllers (CMAC). Proceedings of the International Conference on *Artificial neural networks*, Vol. 2, 1991, pp. 1207–1210 (North-Holland, Helsinki).

17 **Parks, P. C.** and **Militzer, J.** Improved allocation of weights for associative memory storage in learning control systems. Symposium on *Design methods for control systems*, Sept. 4–6 1991, Zurich (Pergamon Press, Oxford)

18 **Lane, S., Handelman, D.** and **Gelfand, J.** Development of adaptive B-splines using CMAC neural networks. IEEE/INNS International Joint Conference on *Neural networks* (IJCNN), Vol. 1, July 1989, pp. 683–688, Washington, DC.

19 **Lane, S. H., Handelman, D. A.** and **Gelfand, J. J.** Theory and development of higher order CMAC neural networks. *IEEE Cont. Sys. Mag.*, April 1992, 23–30.

20 **Brown, M., Harris, C. J.** and **Parks, P. C.** The interpolation capabilities of the binary CMAC. *Neural Networks*, 1993, **6**(3), 429–440.

21 **Cox, M. G.** Algorithms for spline curves and surfaces. NPL Report DITC 166/90, 1990 (National Physical Laboratory, Teddington).

22 **Friedman, J. H.** Multi-variate adaptive regression splines. Technical Report 102, 1988, Stanford University Laboratory for Computational Statistics.

23 **Kavli, T.** ASMOD—An algorithm for adaptive spline modelling of observation data. Technical Report, 1992. (Centre for Industrial Research, Box 124 Blindern, 0314 Oslo 3, Norway).

24 **Haykin, S.** *Adaptive filter theory*, 2nd edn, 1991 (Prentice-Hall, Englewood Cliffs, NJ).

25 **Widrow, B.** and **Stearns, S. D.** *Adaptive signal processing*, 1985, (Prentice-Hall, Englewood Cliffs, NJ).

26 **Widrow, B.** and **Lehr, M. A.** 30 years of adaptive neural networks: perceptron, madaline and backpropagation. *Proc. IEEE*, 1990, **78**(9), 1415–1441.

27 **Kaczmarz, S.** Angenaherte Auflosung von Systemen Linearer Gleichugen *Bull. Int. Acad. Pol. Sci. Lett., Cl. Sci. Math. Nat.* Series A., 1937. 355–357 (a translated version appeared in *Int. J. Control*, 1993, **57** 1269–1271).

28 **Parks, P. C.** and **Militzer, J.** Convergence properties of associative memory storage for learning control systems. *Automat Rem. Control*, 1989, **50**(2), (part 2), 254–286.

29 **Parks, P. C.** and **Militzer, J.** A comparison of five algorithms for the training of CMAC memories for learning control systems. *Automatica*, 1992, **28**(5), 1027–1035.

* To a lesser extent for the Gaussian basis function.

† Especially the Gaussian bar function described earlier and the tree-structured basis function network (**51**), although the basis functions in the latter type are not necessarily restricted to be radially symmetrical.

‡ As mentioned earlier, variants of the LAN with adjustable inner structures have been proposed (**22, 23**) to tackle this problem.

30 **Tolle, H.** and **Ersü, E.** Neurocontrol: learning control systems inspired by neuronal architectures and human problem solving. In *Lecture Notes in Control and Information Sciences*, 1992, No. 172 (Springer, Berlin).

31 **Powell, M. J. D.** Radial basis functions for multivariate interpolation: a review. In *Algorithms for approximation* (Eds. J. C. Mason and M. G. Cox), 1987, pp. 143–167 (Oxford University Press).

32 **Broomhead, D. S.** and **Lowe, D.** Multivariable functional interpolation and adaptive networks. *Complex Syst.*, 1988, **2**, 321–355.

33 **Hartman, E.** and **Keeler, J. D.** Predicting the future: advantages of semilocal units. *Neural Comp.*, 1991, **3**, 566–578.

34 **Sanner, R. M.** and **Slotine, J. E.** Gaussian networks for direct adaptive control. *IEEE Trans. Neural Networks*, 1992, **3**(6), 837–867.

35 **Moody, J.** and **Darken, C. J.** Fast-learning in networks of locally-tuned processing units. *Neural Comp.*, 1989, **1**(2), 281–294.

36 **Powell, M. J. D.** Radial basis function approximation to polynomials. Proceedings of the 12th Biennial Conference on *Numerical analysis*, 1987, pp. 223–241 Dundee (Oxford University, Press).

37 **Chen, S.** and **Billings, S. A.** Recursive prediction error estimator for non-linear models. *Int. J. Control*, 1989, **49**(2), 569–594.

38 **Goodwin, G. C.** and **Payne, R. L.** *Dynamic system identification: experiment design and data analysis*, 1977 (Academic Press, New York).

39 **Ljung, L.** and **Soderstrom, T.** *Theory and practice of recursive identification*, 1983 (MIT Press, Cambridge, MA).

40 **Chen, S., Cowan, C. F. N., Billings, S. A.** and **Grant, P. M.** Parallel recursive prediction error algorithm for training layered neural networks. *Int. J. Control*, 1990, **51**(6), 1215–1228.

41 **Chen, S.** and **Billings, S. A.** Neural networks for linear dynamic system modelling and identification. *Int. J. Control*, Special Issue on Intelligent Control, 1992, **56**(2), 319–346.

42 **Chen, S., Billings, S. A.** and **Grant, P. M.** Recursive hybrid algorithm for non-linear system identification using radial basis function networks. *Int. J. Control*, 1992, **55**(5), 1051–1070.

43 **Duda, R. Q.** and **Hart, P. E.** *Pattern classification and scene analysis*, 1973 (John Wiley, New York).

44 **Kohonen, T.** *Self-organization and associative memory*, Springer Series in Information Sciences, 2nd edn 1987 (Springer, Berlin).

45 **Bishop, C.** Improving the generalization properties of radial basis function neural networks. *Neural Comp.* 1991, **3**, 579–588.

46 **Wright, W. A.** Image labelling with a neural network. Proceedings of the 5th *Alvey Vision Conference*, pp. 227–232 (Department of Trade and Industry, London).

47 **Funahashi, K.** On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 1989, **2**, 183–192.

48 **Hornik, K.** Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 1991 **4**, 251–257.

49 **Billings, S. A.** and **Voon, W. S.** Correlation based model validity tests for non-linear models. *Int. J. Control*, 1986, **44**(1), 235–244.

50 **Brown, M.** and **Harris, C. J.** The B-Spline neurocontroller. In *Parallel processing in a control systems environment* (Eds. E. Rogers and Y. Li) 1993, pp. 134–166 (Prentice-Hall, Englewood Cliffs, NJ).

51 **Sanger, T.** A tree-structured adaptive network for function approximation in high-dimensional spaces. *IEEE Trans. Neural Networks*, 1991, **2**(2), 285–293.

APPENDIX

The overall response of an $n_0 - n_1 - n_2 - (n_3 = 1)$ MLP is

$$\hat{y} = \sum_{i=1}^{n_2} w_{3,1,i} x_{2,i} = \sum_{i=1}^{n_2} w_{3,1,i} \phi\left(\sum_{j=1}^{n_1} w_{2,i,j} x_{1,j} + \mu_{2,i} \right)$$

$$= \sum_{i=1}^{n_2} w_{3,1,i} \phi\left\{ \sum_{j=1}^{n_1} w_{2,i,j} \phi\left(\sum_{k=1}^{n_0} w_{1,j,k} x_{0,k} + \mu_{1,j} \right) + \mu_{2,i} \right\}$$

The activation function $\phi(\cdot)$ is assumed to be equation (34) and its derivative is

$$\frac{\partial \phi(x)}{\partial x} = \phi(x)\{1 - \phi(x)\}$$

For the output node, the gradient vector is

$$\boldsymbol{\psi}_{3,1} = [\psi_{3,1,1} \cdots \psi_{3,1,n_2}]^{\mathrm{T}}$$

with

$$\psi_{3,1,j} = \frac{\partial \hat{y}}{\partial w_{3,1,j}} = x_{2,j}, \qquad 1 \leqslant j \leqslant n_2$$

For the ith node in the second layer, where $1 \leqslant i \leqslant n_2$, the gradient vector is

$$\boldsymbol{\psi}_{2,i} = [\psi_{2,i,0} \psi_{2,i,1} \cdots \psi_{2,i,n_1}]^{\mathrm{T}}$$

with

$$\psi_{2,i,0} = \frac{\partial \hat{y}}{\partial \mu_{2,i}} = w_{3,1,i} x_{2,i}(1 - x_{2,i})$$

and

$$\psi_{2,i,j} = \frac{\partial \hat{y}}{\partial w_{2,i,j}} = \psi_{2,i,0} x_{1,j} \qquad 1 \leqslant j \leqslant n_1$$

For the ith node in the first layer, where $1 \leqslant i \leqslant n_1$, the gradient vector is

$$\boldsymbol{\psi}_{1,i} = [\psi_{1,i,0} \psi_{1,i,1} \cdots \psi_{1,i,n_0}]^{\mathrm{T}}$$

with

$$\psi_{1,i,0} = \frac{\partial \hat{y}}{\partial \mu_{1,i}}$$

$$= \sum_{k=1}^{n_2} w_{3,1,k} x_{2,k}(1 - x_{2,k}) w_{2,k,i} x_{1,i}(1 - x_{1,i})$$

and

$$\psi_{1,i,j} = \frac{\partial \hat{y}}{\partial w_{1,i,j}} = \psi_{1,i,0} x_{0,j} \qquad 1 \leqslant j \leqslant n_0$$