Analysis of minimal radial basis function network algorithm for real-time identification of nonlinear dynamic systems

Y.Li, N.Sundararajan and P.Saratchandran

Abstract: A performance analysis is presented of the minimal resource allocating network (MRAN) algorithm for online identification of nonlinear dynamic systems. Using nonlinear time-invariant and time-varying identification benchmark problems, MRAN's performance is compared with the online structural adaptive hybrid learning (ONSAHL) algorithm. Results indicate that the MRAN algorithm realises networks using fewer hidden neurons than the ONSAHL algorithm, with better approximation accuracy. Methods for improving the run-time performance of MRAN for real-time identification of nonlinear systems are developed. An extension to MRAN is presented, which utilises a winner neuron strategy and is referred to as the extended minimum resource allocating network (EMRAN). This modification reduces the computation load for MRAN and leads to considerable reduction in the identification time, with only a small increase in the approximation error. Using the same benchmark problems, results show that EMRAN is well suited for fast online identification of nonlinear plants.

1 Introduction

Neural networks have been used as nonlinear dynamic system controllers to tackle problems for which conventional approaches have been proven to be ineffective [1]. However, because a large amount of computation time is required for the learning process, the practical use of neural networks for online control schemes is sparse (generally, an offline training process for neural controllers is required), especially in areas such as flight control [2, 3]. Hence, the problem of designing a fast online learning algorithm for practical implementation of neural control schemes remains an active research topic.

Since the late 1980s, there has been considerable interest in radial basis function (RBF) neural networks, due to their good global generalisation ability and a simple network structure that avoids lengthy calculations [4]. Gaussian functions are selected in a majority of cases as radial basis functions, even though other functions like thin plate functions can also be used [5].

These Gaussian functions have two parameters; the centre and width, which have to be fixed. Several algorithms have been proposed for training the RBF network [5–7]. The classical approach to RBF implementation is to fix the number of hidden neurons *a priori* along with its centres and widths, based on some properties of the input data, and then estimate the weights connecting the hidden and output neurons. Two methods have been proposed to find the proper number of hidden neurons for a given problem. The concept of building up of the hidden neurons from zero to the required number has been introduced [8], with the update of the RBF parameters done by a gradient descent algorithm. An alternate approach is to start with as many hidden units as the number of samples, and then reduce them using a clustering algorithm, which essentially puts patterns that are close in the input space into a cluster to remove the unnecessary hidden neurons [9]. However, in all these studies, the main learning scheme is of the batch type, which is not suitable for online learning.

Platt [10] proposed a sequential learning algorithm to overcome the above drawbacks. In Platt's resource allocating network (RAN) algorithm, hidden neurons are added based on the novelty of the input data, and the weights connecting the hidden neurons to the output neurons are estimated using the least mean square (LMS) algorithm. Platt showed the resulting network topology to be more parsimonious than the classical RBF networks. Modifications have been proposed to improve RAN by using an extended Kalman Filter (EKF) [11] instead of the LMS to estimate the network parameters. The resulting network, called RANEKF, is more compact and has better accuracy than RAN. A further improvement to RAN and RANEKF has been proposed [12], in which a pruning strategy was introduced to remove those neurons that consistently made little contribution to the network output. The resulting network, called minimal RAN(MRAN), was shown to be more compact than RAN and RANEKF for several applications in the areas of function approximation and pattern classification [13]. Preliminary results using MRAN for nonlinear system identification problems have been presented [14].

Another sequential learning algorithm has been proposed by Junge and Unbehauen [15, 16]. Their algorithm incorporates the idea of online structural adaptation

IEE Proceedings online no. 20000549

DOI: 10.1049/ip-cta:20000549

Paper first received 27th April 1999 and in revised form 13th April 2000

The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798
E-mail: ensundara@ntu.edu.sg

to add new hidden neurons, and the method uses an error-sensitive clustering algorithm to adapt the centre and width of the hidden neurons. The algorithm, known as online structural adaptive hybrid learning (ONSAHL), has been shown [16] to produce compact networks for nonlinear dynamic system identification problems.

We present a comparison of the performance of MRAN with the ONSAHL algorithm for the same nonlinear identification benchmark problems as Junge and Unbehauen [16]. This study is intended to compare the complexity of the resulting networks and the accuracy of approximation using MRAN and ONSAHL in the field of nonlinear system identification using RBF networks.

For any practical application of a newly developed identification algorithm, it is important to study the real-time implementation of the algorithm; this is undertaken for the MRAN algorithm in this paper. In the MRAN algorithm, the parameters of the network include all the hidden neuron's centres, widths and weights, and these have to be updated in every step. This causes the size of the matrices to be updated to become large as the hidden neurons increase and the RBF network structure becomes more computationally complex, resulting in a large computation load and limiting the use of MRAN for real-time implementation. An analysis of the breakdown of the computation time for one cycle of MRAN under the Visual C++ environment is presented in terms of the different steps in the algorithm. Based on this analysis of the number of operations, specifically on the number of multiplications in a cycle, the bottle-neck in the computation time is found.

Based on this analysis, an extension to MRAN, called extended MRAN (EMRAN), is proposed. The focus in EMRAN is to reduce the computation load of MRAN and to realise a scheme for fast online identification. For this purpose a 'winner neuron' strategy is incorporated into the conventional MRAN algorithm. The key idea in the EMRAN algorithm is that, in every step, only those parameters that are related to the selected winner neuron are updated by EKF. EMRAN attempts to reduce the online computation time considerably and avoids the overflow of the memory, retaining at the same time the good characteristics of MRAN, i.e. fewer hidden neurons and a lower approximation error. These benefits of EMRAN are illustrated using the same benchmark problems from the nonlinear system identification area, which consist of SISO nonlinear time-varying and MIMO nonlinear time-invariant plants. Simulation results show that EMRAN is well suited for real-time implementation of identification of nonlinear dynamic systems.

2 MRAN algorithm and nonlinear system identification

In this Section, the MRAN algorithm is briefly described and the problem of identifying a given nonlinear dynamic system is described. A brief summary of the ONSAHL algorithm is also presented.

2.1 Minimal resource allocating network algorithm

The MRAN algorithm, proposed by Lu *et al.* [12, 13], combines the growth criteria of RAN with a pruning strategy to realise a minimal network structure. This algorithm is an improvement to the RAN of Platt [10] and the RANEKF algorithm of Kadirkamanathan [11]. In this Section, we present the MRAN algorithm for training

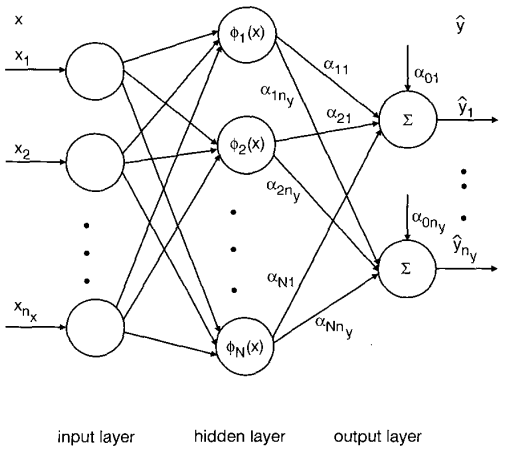

Fig. 1 *Radial basis function network model*

RBF networks. Before explaining the algorithm in detail, the RBF network is briefly described.

Fig. 1 shows a typical RBF network, with n_x inputs $\boldsymbol{x}$ $(x_1, \ldots, x_{n_x})$, and n_y outputs $\hat{\boldsymbol{y}}$ $(\hat{y}_1 \cdots \hat{y}_{n_y})$. The hidden layer consists of N computing units (ϕ_1 to ϕ_N), connected to the output by N weight vectors ($\boldsymbol{a}_1$ to $\boldsymbol{a}_N$). The outputs of the network which approximate the true output $\boldsymbol{y}$ are

$$\hat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{a}_0 + \sum_{n=1}^{N} \boldsymbol{a}_n \phi_n(\boldsymbol{x}) \tag{1}$$

where $\phi_n(\boldsymbol{x})$ is the response of the nth hidden neuron to the input $\boldsymbol{x}$, and $\boldsymbol{a}_n$ is the weight connecting the nth hidden unit to the output unit. $\boldsymbol{a}_0$ is the bias term, and $\phi_n(\boldsymbol{x})$ is a Gaussian function given by

$$\phi_n(\boldsymbol{x}) = exp\left(-\frac{\|\boldsymbol{x} - \underline{\mu}_n\|^2}{\sigma_n^2}\right) \tag{2}$$

$\underline{\mu}_n$ is the centre for the nth hidden neuron, and σ_n is the width of the Gaussian function. $\|\|$ denotes the Euclidean norm.

In the MRAN algorithm, the RBF network begins with no hidden units, i.e. $n = 0$. As the input-output training data $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ (i is time index) are received, the network is built up, based on certain growth criteria. The following steps describe the basic ideas of the MRAN algorithm.

Step 1: Calculate the three errors defined

The first step of the algorithm is to check whether the criteria for recuriting a new hidden unit are met

$$\|\boldsymbol{e}_i\| = \|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i(\boldsymbol{x}_i)\| > E_1 \tag{3}$$

$$e_{rmsi} = \sqrt{\sum_{j=i-(M-1)}^{i} \frac{\|\boldsymbol{e}_j\|^2}{M}} > E_2 \tag{4}$$

$$d_i = \|\boldsymbol{x}_i - \mu_{ir}\| > E_3 \tag{5}$$

where $\underline{\mu}_{ir}$ is the centre of the hidden unit closest to current input $\boldsymbol{x}_i$. E_1, E_2 and E_3 are thresholds to be selected appropriately. Eqn. 3 decides if the existing nodes are insufficient to obtain a network output that meets the error specification. Eqn. 4 checks whether the network met the required sum squared error specification for the past M outputs of the network. Eqn. 5 ensures that the new node to be added is sufficiently far from all the existing nodes.

Only when all these criteria are met has a new hidden node been recruited. Then go to Step 2 to add a new RBF hidden unit; otherwise go to Step 3 to update all the parameters of the network using EKF.

Step 2: Inclusion of new RBF hidden unit

When all the criteria in Step 1 are satisfied, a new hidden unit is recruited. Each new hidden unit added to the network will have the following associated parameters:

$$\boldsymbol{a}_{N+1} = \boldsymbol{e}_i, \quad \underline{\mu}_{N+1} = \boldsymbol{x}_i, \quad \sigma_{N+1} = \kappa\|\boldsymbol{x}_i - \underline{\mu}_{ir}\| \tag{6}$$

Those parameters are set to remove the error caused. The overlap of the responses of the hidden units in the input space is determined by κ, the overlap factor. After adding the new hidden neuron, go to Step 5 to perform a pruning strategy.

Step 3: Calculating gradient matrix $\boldsymbol{B}_i$

If the three criteria for adding a new hidden unit cannot be satisfied, then an adaptation of the network parameters should be performed. $\boldsymbol{B}_i = \nabla_w \boldsymbol{f}(\boldsymbol{x}_i)$ is the gradient matrix of the function $\boldsymbol{f}(\boldsymbol{x}_i)$ with respect to the parameter vector $\boldsymbol{w}$, evaluated at $\boldsymbol{w}_{i-1}$, which is used in the next step.

$$\begin{aligned}\boldsymbol{B}_i = [&\boldsymbol{I}, \phi_1(\boldsymbol{x}_i)\boldsymbol{I}, \phi_1(\boldsymbol{x}_i)(2\boldsymbol{a}_1/\sigma_1^2)(\boldsymbol{x}_i - \underline{\mu}_1)^T,\\ &\phi_1(\boldsymbol{x}_i)(2\boldsymbol{a}_1/\sigma_1^3)\|\boldsymbol{x}_i - \underline{\mu}_1\|^2, \ldots,\\ &\phi_N(\boldsymbol{x}_i)\boldsymbol{I}, \phi_N(\boldsymbol{x}_i)(2\boldsymbol{a}_N/\sigma_N^2)(\boldsymbol{x}_i - \underline{\mu}_N)^T,\\ &\phi_N(\boldsymbol{x}_i)(2\boldsymbol{a}_N/\sigma_N^3)\|\boldsymbol{x}_i - \underline{\mu}_N\|^2]^T\end{aligned} \tag{7}$$

After this preparation, the vector $\boldsymbol{w}$ can be updated; therefore go to Step 4.

Step 4: Updating the parameters using EKF

In this Step, the network parameters $\boldsymbol{w} = [\boldsymbol{a}_0^T, \boldsymbol{a}_1^T, \underline{\mu}_1^T, \sigma_1, \ldots, \boldsymbol{a}_N^T, \underline{\mu}_N^T, \sigma_N]^T$ are adapted using the EKF as follows:

$$\boldsymbol{w}_i = \boldsymbol{w}_{i-1} + \boldsymbol{K}_i\boldsymbol{e}_i \tag{8}$$

where $\boldsymbol{K}_i$ is the Kalman gain matrix given by

$$\boldsymbol{K}_i = \boldsymbol{P}_{i-1}\boldsymbol{B}_i[\boldsymbol{R}_i + \boldsymbol{B}_i^T\boldsymbol{P}_{i-1}\boldsymbol{B}_i]^{-1} \tag{9}$$

$\boldsymbol{R}_i$ is the variance of the measurement noise. $\boldsymbol{P}_i$ is the error covariance matrix, which is updated by

$$\boldsymbol{P}_i = [\boldsymbol{I}_{z\times z} - \boldsymbol{K}_i\boldsymbol{B}_i^T]\boldsymbol{P}_{i-1} + q\boldsymbol{I}_{z\times z} \tag{10}$$

q is a scalar that determines the allowed random step in the direction of the gradient vector. If the number of parameters to be adjusted is z, then $\boldsymbol{P}_i$ is a $z \times z$ positive definite symmetric matrix. When a new hidden neuron is allocated, the dimensionality of $\boldsymbol{P}_i$ increases to

$$\boldsymbol{P}_i = \begin{pmatrix} \boldsymbol{P}_{i-1} & 0 \\ 0 & p_0\boldsymbol{I}_{z_1\times z_1} \end{pmatrix} \tag{11}$$

where the new rows and columns are initialised by p_0. p_0 is an estimate of the uncertainty when the initial values are assigned to the parameters. The dimension z_1 of the identity matrix $\boldsymbol{I}$ is equal to the number of new parameters introduced by the new hidden neuron. Then go to Step 5 for a pruning strategy.

Step 5: Pruning strategy

The last Step of the algorithm is to prune those hidden neurons that contribute little to the network's output for N_w consecutive observations. Let matrix $\boldsymbol{O} = (\boldsymbol{o}_1, \ldots, \boldsymbol{o}_N)$ denote the outputs of the hidden layer and $\boldsymbol{A}$ denote the weight matrix $\boldsymbol{A} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N)$. Considering the jth output of the nth hidden neuron, $o_{nj} (j = 1 \cdots n_y)$

$$o_{nj} = a_{nj}\, exp\left(-\frac{1}{\sigma_n^2}\|\boldsymbol{x} - \underline{\mu}_n\|^2\right), \quad n = 1 \cdots N; \; j = 1 \cdots n_y \tag{12}$$

If a_{nj} or σ_n in eqn. 12 is small, o_{nj} might become small. In addition, if $\|\boldsymbol{x} - \underline{\mu}_n\|$ is large, the output will be small. This would mean that the input is sufficiently far away from the centre of this hidden neuron. To reduce inconsistency caused by using the absolute value of the output, this value is normalised to that of the highest output:

$$r_{nj} = \frac{o_{nj}}{max\{o_{1j}, o_{2j}, \cdots, o_{Nj}\}}, \quad n = 1 \cdots N; \; j = 1 \cdots n_y \tag{13}$$

The normalised output of each neuron r_{nj} is then observed for N_w consecutive inputs. A neuron is pruned if its output $r_{nj} (j = 1 \cdots n_y)$ falls below a threshold value (δ) for the N_w consecutive inputs. The dimensionality of all the related matrixes is then adjusted to suit the reduced network.

The sequential learning algorithm for MRAN can be summarised as follows.

(i) Obtain an input and calculate the network output (eqns. 1 and 2) and the corresponding errors (eqns. 3–5).
(ii) Create a new RBF centre (eqn. 6) if all three inequality (eqns. 3–5) hold.
(iii) If condition (ii) is not met, adjust the weights and widths of the existing RBF network using EKF (eqns. 7–11).

In addition, a pruning strategy is adopted,

(*a*) If a centre's normalised contribution to the output for a certain number of consecutive inputs is found to be below a threshold value, that centre is pruned.
(*b*) The dimensions of the corresponding matrix are adjusted and the next input is evaluated.

Successful applications of MRAN have been reported in different areas such as pattern classification, function approximation and time series prediction [12, 13].

2.2 Nonlinear system identification using RBF network

Generally, a nonlinear multi-input multi-output (MIMO) dynamic system in discrete form is represented by the following input-output description:

$$\boldsymbol{y}(i) = \boldsymbol{g}[\boldsymbol{y}(i-1), \cdots, \boldsymbol{y}[i-k_y), \boldsymbol{u}(i-1), \cdots, \boldsymbol{u}(i-k_u)] \tag{14}$$

where $\boldsymbol{y}$ is a vector containing m system outputs; $\boldsymbol{u}$ is a vector for r system inputs; $\boldsymbol{g}[\cdots]$ is a nonlinear vector function, representing m hyper-surfaces of the system: $\boldsymbol{i}$ represents the time index; and k_y and k_u are the maximum lags of the output and input vectors, respectively.

The problem of identification is as follows: given the output $\boldsymbol{y}$ and input $\boldsymbol{u}$ over a certain interval of time, find the nonlinear function $\boldsymbol{g}(.)$ which fits the data closely. This problem can be converted to a nonlinear approximation problem for RBF networks by defining the variables inside the brackets of eqn. 14 as the RBF network's inputs, i.e. $\boldsymbol{x}$, and $\hat{\boldsymbol{y}}_{n_y} (n_y = m)$ as the network outputs. Based on the past

system outputs and inputs, construct the inputs to the neural network. $\boldsymbol{x}_{n_x \times 1}$, as

$$\boldsymbol{x} = [\hat{\boldsymbol{y}}_{i-1}^T, \cdots, \hat{\boldsymbol{y}}_{i-k_y}^T, \boldsymbol{u}_{i-1}^T, \cdots, \boldsymbol{u}_{i-k_u}^T]^T \tag{15}$$

The output of the network will be an approximation to $\boldsymbol{y}(i)$ and is denoted by $\hat{\boldsymbol{y}}(i)$. Essentially, the problem of the identification of a nonlinear dynamic system is converted to a nonlinear time series problem with one step ahead prediction. The RBF network is used to give a best approximation of $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ for the plant nonlinear function $\boldsymbol{g}$.

Many learning algorithms exist for the RBF network. Generally, the good characteristics of a learning algorithm include a more compact network structure, fewer hidden neurons, lower approximate error etc. However, for real-time implementation, model identification for one sample of data must be finished before the next sample arrives. Suppose that the learning time for one sample of data (the cycle time) is t_c, then $t_c < T$, where $\boldsymbol{T}$ is the sampling time. For this reason, learning time defined by t_c plays a crucial role in selecting a RBF learning algorithm for nonlinear system identification problems.

2.3 ONSAHL algorithm

The ONSAHL algorithm has been proposed especially to identify online time-varying nonlinear dynamic systems [16]. The ONSAHL algorithm uses an extended RBF network, referred to as a direct linear feedthrough RBF (DLF-RBF). DLF-RBF is composed of a nonlinear RBF sub-network and a linear DLF sub-network, which are connected in parallel, performing a mapping from the input layer directly to the output layer. The ONSAHL algorithm uses the same growth criteria as in Platt's RAN, but differs in the way that the centres and widths are adjusted. Unlike RAN, where the centres of all hidden neurons are updated to fit the training data (in LMS sense), in the ONSAHL only the centre and width of the neuron nearest to the input are updated in the first step. Then all the weights connected to the output layer are updated, as in the case of RAN, using the RLS method.

The algorithm has been tested on two single-input single-output (SISO) nonlinear time-invariant and time-varying dynamic system identification problems. (For a detailed description of the ONSAHL algorithm see elsewhere [16]).

3 Benchmark problems on nonlinear dynamic systems identification

In this Section, we compare the performance of MRAN with the ONSAHL algorithm for two benchmark problems on nonlinear dynamic systems identification [16]. For convenience, these two benchmark problems are referred to as BM-1 and BM-2.

3.1 BM-1: nonlinear SISO time-invariant system

The nonlinear SISO time-invariant system (BM-1) to be identified is described by the following first-order difference equation:

$$y(i) = \frac{29}{40} sin\left(\frac{16u(i-1) + 8y(i-1)}{(3 + 4u(i-1)^2 + 4y(i-1)^2}\right) + \frac{2}{10}u(i-1) + \frac{2}{10}y(i-1) \tag{16}$$

A random signal uniformly distributed in the interval $[-1, 1]$ is used for $u(i)$ in the system. For comparison purposes, previous error criteria $I_d(i)$ are used [15]:

$$I_d(i) = \frac{1}{n_y \times N_w} \sum_{p=0}^{N_w-1} \sum_{j=1}^{n_y} |y_j(i-p) - \hat{y}_j(i-p)| \tag{17}$$

In this case, the MRAN's three criteria gates are selected as $E_1 = 0.01$, $E_2 = 0.1$, $E_3 = max\{\varepsilon_{max} \times \gamma^i, \varepsilon_{min}\}$, $\varepsilon_{max} = 1.2$, $\varepsilon_{min} = 0.6$, $\gamma = 0.997$. We use $\delta = 0.0001$ as the pruning threshold, and the size of the two sliding windows (M, N_w) as 48.

The identification results using MRAN and ONSAHL are shown in Figs. 2–4. Figs. 2 and 3 present the hidden neuron evolution history, along with the time histories of the three error functions. From Figs. 2 and 3 we can see clearly how the hidden neurons are added and pruned according to the three criteria. In this case, MRAN takes eight hidden units to identify the system; whereas the ONSAHL takes 23 hidden neurons for the same problem [16].

Fig. 4 shows the time history of the error index I_d. It can be seen from Fig. 4 that MRAN's error is decreasing

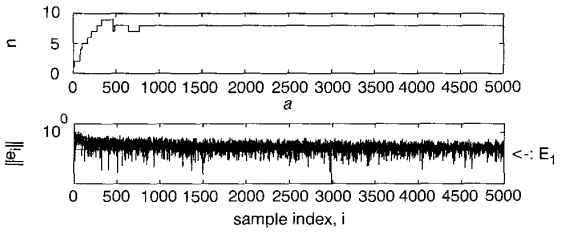

Fig. 2 *BM-1: performance of MRAN algorithm*

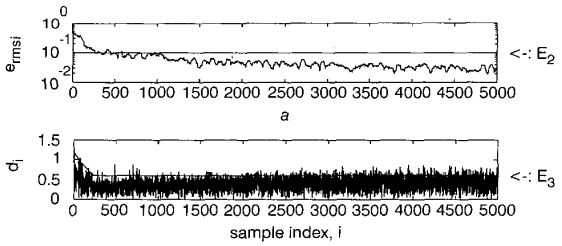

Fig. 3 *BM-1: performance of MRAN algorithm cont.*

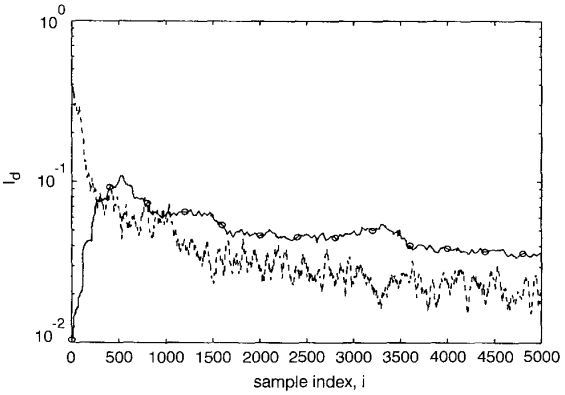

Fig. 4 *BM-1: evolution of error (I_d) (MRAN against ONSAHL)*

---- MRAN algorithm
-O-O-ONSAHL algorithm

quickly and in the steady state it is lower than that of ONSAHL.

3.2 BM-2: nonlinear SISO time-varying discrete system

The second example BM-2 is selected to identify a nonlinear SISO time-varying discrete system given below:

$$y(i) = \frac{29\beta(i)}{40} sin\left(\frac{16u(i-1) + 8y(i-1)}{\beta(i)(3 + 4u(i-1)^2 + 4y(i-1)^2}\right) + \frac{2}{10}u(i-1) + \frac{2}{10}y(i-1) \quad (18)$$

$\beta(i)$ is a time-varying parameter given in Table 1. The system input signal u_i used in this example is a random signal with uniform distribution in the interval $[-1, 1]$.

Figs. 5–7 present the identification results for this benchmark problem. From Figs. 5 and 6 the most important thing we can observe is the change of the hidden neurons together with β. Briefly, when the system dynamics change because of the variation in β, the network has to add new neurons to allow for the changes. When the change of β does not affect the dynamics and changes the complexity of the system after some time, the old neuron will be pruned. Compared to previous results, [16] MRAN uses only 11 hidden neurons, whereas the ONSAHL uses 25 hidden neurons. From Fig. 7 we can see that the MRAN algorithm also achieves a better approximation result, i.e. I_d is smaller. Furthermore, for this example, the MRAN starts with no hidden neuron, whereas ONSAHL starts with a trained network from BM-1 (23 hidden neurons).

The identification results for both MRAN and ONSAHL algorithms are compared in Table 2, where I_{dav} is the average value of the I_d calculated from the 1500th to 5000th sample. It is selected in this way because as you can see, after 1500 samples, the identification result converges to a comparatively smooth one.

Based on these two problems, it can be concluded that MRAN is able to perform better with a smaller network structure. Noted also that there is no pruning strategy in the ONSAHL algorithm. However, for practical use, MRAN's online implementation issues have to be analysed. Any improvements to MRAN to make it perform with less computation time are always welcome, and such a modification is described in the following.

Table 1: Evolution of β (BM-2)

Index(i)	0 to 1500	1501 to 2500	2501 to 5000
$\beta(i)$	1.0	0.9	0.8

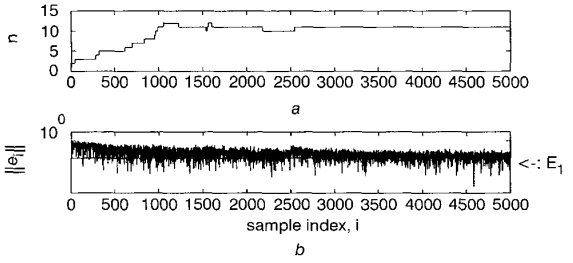

Fig. 5 *BM-2: performance of MRAN algorithm*

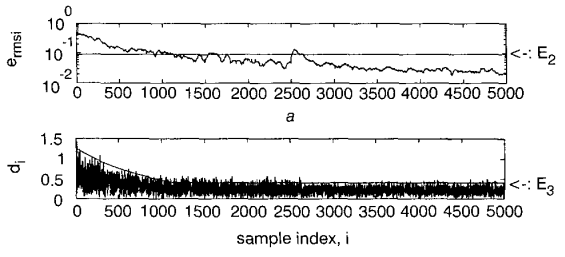

Fig. 6 *BM-2: performance of MRAN algorithm*

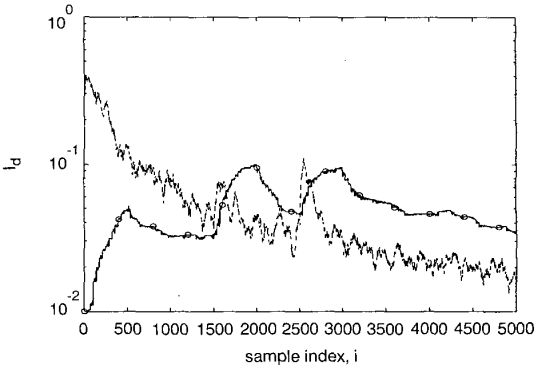

Fig. 7 *BM-2: evolution of error (I_d) (MRAN against ONSAHL)*
---- MRAN algorithm
-O-O-ONSAHL algorithm

Table 2: Comparison of identification results of MRAN and ONSAHL

	BM-1		BM-2	
Performance	I_{dav}	Hidden units	I_{dav}	Hidden units
ONSAHL	0.0437	23	0.0586	25
MRAN	0.0261	8	0.0326	11

4 Real-Time implementation and EMRAN algorithm

In practice, it should be possible to run any identification algorithm in real-time and MRAN is no exception. Since MRAN uses a sequential learning scheme, for real-time implementation, the learning time t_c for one set of input and output data must be less than the sampling time T selected for identification. In this Section, we estimate MRAN's learning time t_c for one cycle, based on its detailed breakdown into a number of computational steps in one cycle. All the time estimates are based on running MRAN on a Pentium 120 MHz computer under the environment of $VC^{++}5.0$.

Looking at MRAN equations (eqns. 1–13), it is clear that t_c mainly consists of five parts, corresponding to the Steps 1 to 5 described in Section 2.1. The computation time for step i is referred to as t_{c_i}, and the cycle time t_c is given by

$$t_c = \sum_{i=1}^{5} t_{c_i} \quad (19)$$

For this timing study, the benchmark problem BM-2 has been selected as the first candidate. The time for training

one sample of data is determined by factors such as the RBF network structure (number of inputs, number of outputs) and the number of hidden neurons. Therefore, for finding out the t_c, the network size was varied from 5 to 35 hidden neurons in an increment of 5, without worrying about the approximation accuracy.

The following question was posed: if the MRAN network had 10 hidden neurons for the problem of BM-2, what are the constituent times t_{c_i}s without worrying whether the 10 hidden neuron network produced a good approximation? This approach makes the timing analysis easier, as MRAN produces a network with a varying number of hidden neurons, and to calculate times based on this would be difficult. For the BM-2 benchmark problem, the breakdown for all the times for the 5 steps is given in Table 3.

In Table 3, the last row gives the total time t_c. Although it is not correct to use 'total' time as the training time for each sample of data, this calculation considers the 'worst case' scenario for calculating t_c (For example, if a new training pair x_i, y_i satisfies the condition for recruiting a new hidden unit, then after the hidden neuron is added, it directly goes to the pruning strategy of Step 5; therefore, t_c is only the sum of t_{c1}, t_{c2} and t_{c5}).

Even though the BM-2 problem gives some idea about the MRAN computation cycle times, the BM-2 problem is still a SISO nonlinear system, although time-varying. A realistic assessment can only be made if the selected problem is of a reasonably large size such as a MIMO problem. In this context, the 2-input 2-output nonlinear system identification problem is selected here as the BM-3 problem [15].

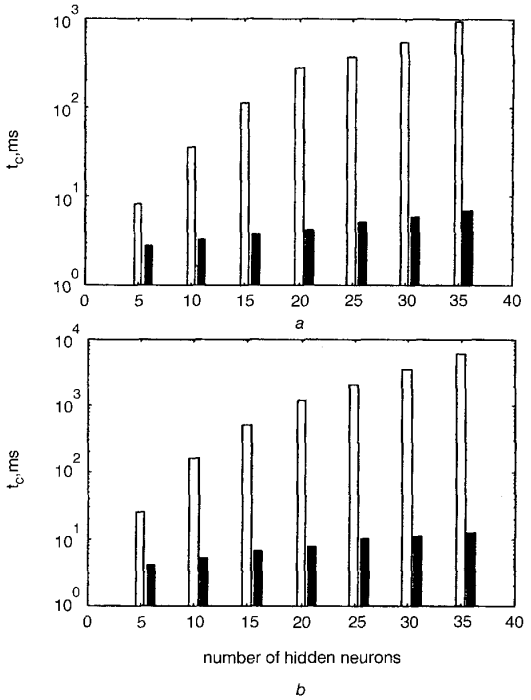

Fig. 8 *Comparison of cycle times for MRAN and EMRAN algorithms*
a BM-2
b BM-3
■ EMRAN algorithm
□ MRAN algorithm

4.1 BM-3: nonlinear MIMO time-invariant discrete system

The MIMO nonlinear dynamic system is given by

$$y_1(i) = \frac{15u_1(i-1)y_2(i-2)}{2+50u_1(i-1)^2} + \frac{1}{2}u_1(i-1) - \frac{1}{4}y_2(i-2) + \frac{1}{10}$$

$$y_2(i) = \frac{sin[\pi u_2(i-1)y_1(i-2)] + 2u_2(i-1)}{3} \quad (20)$$

The two random input signals $u_1(i)$, $u_2(i)$, uncorrelated and uniformly distributed in the interval $[-1, 1]$, are used to generate the online training set, together with the output. Once the plant is identified by the MRAN network, for testing the accuracy of the identified model, the signals shown in Fig. 8 are used as inputs. These signals have both frequency modulation and amplitude modulation so that they can test the RBFNN's generalisation ability and adaptability to the data's oscillation.

Table 4 gives the computation cycle times for the five steps with varying hidden neurons. It can be seen straightaway that because of the greater number of inputs and outputs, the cycle times for BM-3 are considerably higher, especially for a higher number of hidden neurons. It is also evident that Step 4 is the real bottle-neck and consumes a large chunk of computational overhead.

From Tables 3 and 4, it can be seen that with both the increase of the network inputs and outputs and also the number of hidden neurons, the amount of time consumed in Step 4 (tuning parameters using EKF) is large. For example, when the hidden neuron reaches 30, from Table 4 we can see that Step 4 consumes more than 99% of the total training time, and to satisfy $t_c < T$, T must be greater than 3.6 s, which is unacceptable for many real systems.

To find out why the amount of time consumed in Step 4 is large with the increase in the hidden neurons, we take a close look at the computations involved in Step 4, i.e. eqns.

Table 3: Computation cycle time t_c (ms) for MRAN (BM-2 problem)

Hidden units	5	10	15	20	25	30	35
t_{1c}	0.82	1.37	2.07	2.38	3.00	3.65	4.55
t_{2c}	0.003	0.004	0.004	0.004	0.005	0.005	0.005
t_{3c}	1.10	2.70	3.32	4.80	5.97	7.32	8.97
t_{4c}	6.14	30.97	105.5	269.7	360.9	533.2	936.0
t_{5c}	0.09	0.17	0.26	0.33	0.44	0.53	0.67
t_c	8.15	35.21	111.2	277.2	370.3	544.7	950.2

Table 4: Computation cycle time t_c (ms) for MRAN (BM-3 problem)

Hidden units	5	10	15	20	25	30	35
t_{1c}	1.51	2.63	3.87	5.13	7.11	8.07	9.37
t_{2c}	0.005	0.005	0.008	0.008	0.009	0.009	0.01
t_{3c}	1.60	3.67	5.87	8.13	9.70	11.98	14.28
t_{4c}	22.2	154.3	504.1	1202	2080	3548	6195
t_{5c}	0.10	0.18	0.26	0.36	0.46	0.55	0.68
t_c	25.42	160.8	514.1	1215	2097	3568	6219

8–10. As multiplication consumes more time than addition and subtraction, it is worth looking at the number of multiplications involved in Step 4. Note that when matrix $U_{c\times d}$ multiplies matrix $V_{d\times h}$, the total number of separate multiplications is $c \times d \times h$. The matrices involved in Step 4 (eqns. 8–10) and their sizes are K_i:(S, n_y); B_i:(S, n_y); P_i:(S, S); w_i:$(S, 1)$; e_i:$(n_y, 1)$; R_i:(n_y, n_y), and where S is defined as $S = N \times (n_x + n_y + 1) + n_y$;

With the sizes of the above matrices known, the following are the total number of multiplications needed

(i) to calculate $K_i B_i^T P_{i-1}$:$S^3 + S^2 \times n_y$
(ii) to calculate eqn. 9: suppose $n_y < 10$, and so the inverse calculation of the matrix (with the size $n_y \times n_y$) can be reasonably omitted) $S^2 \times n_y + S \times n_y^2$
(iii) to calculate $(K_i e_i)$:$S \times n_y$

The total number of multiplications in Step 4 is

$$sum(N, n_x, n_y) = S^3 + 2 \times S^2 \times n_y + S \times (n_y^2 + n_y) \quad (21)$$

From eqn. 21, the total time for Step 4 is a third-order polynomial and is a function of N, n_x, n_y. Hence if the number of neurons or inputs and outputs increases, the computational time for Step 4 will increase enormously.

If MRAN has to be modified for real-time implementation, this bottle-neck in Step 4 has to be overcome. Such a modification to MRAN is discussed below.

4.2 EMRAN algorithm

We see from eqn. 21 that the weakness in MRAN which increases its computational load is that all the parameters of the network, including all the hidden neuron's centres, widths and weights, have to be updated in every step; and so the size of the matrices to be updated becomes large as the number of hidden neurons increases.

To overcome this bottle-neck of MRAN for real-time implementation, an algorithm is proposed called EMRAN, which is an improved version of the MRAN algorithm. For this purpose, a 'winner neuron' strategy is incorporated similar to that described for the ONSAHL algorithm. The key idea of the EMRAN algorithm is that, in every step, only those parameters that are related to the selected winner neuron are updated by the EKF algorithm. The winner neuron is defined as that neuron in the network which is closest (in some norm sense) to the current input data [16]. EMRAN attempts to reduce the online computation time considerably and avoid the overflow of memory, at the same time retaining the good characteristics of MRAN, i.e. fewer hidden neurons, lower approximation error etc.

Basically, EMRAN has the same form as MRAN and all the equations are the same, and hence are not repeated here. Only the changes are highlighted below.

In eqn. 6, $\underline{\mu}_{ir}$ is defined as the hidden neuron that is the nearest to the current input data x_i in the input space. Here this special hidden neuron is the winner neuron, and the parameters related to this neuron are denoted as $\underline{\mu}^*$, δ^* and a^*. The criteria for adding and pruning the hidden neurons are all the same; the difference is that if the training sample does not meet the criteria for adding a new hidden neuron, the network parameters w^* related to only the winner neuron are updated, using the EKF as follows:

$$w_i^* = w_{i-1}^* + K_i^* e_i \quad (22)$$

Obviously, $w^* = [a_0^T, a^{*T}, \underline{\mu}^{*T}, \sigma^*] \subseteq w$.

In eqn. 22, K_i^* is the Kalman gain matrix with the size of $(2n_y + n_x + 1) \times n_y$:

$$K_i^* = P_{i-1}^* B_i^* [R_i + B_i^{*T} P_{i-1}^* B_i^*]^{-1} \quad (23)$$

$B_i^* = \nabla_{w^*} f(x_i)$ is the gradient matrix of the function $f(x_i)$, with respect to the parameter vector w^* evaluated at w_{i-1}^*:

$$B_i^* = \left[I, \phi^*(x_i) I, \phi^*(x_i) \frac{2a^*}{\sigma^{*2}} [x_i - \underline{\mu}^*]^T, \right.$$
$$\left. \phi^*(x_i) \frac{2a^*}{\sigma^{*3}} \|x_i - \underline{\mu}^*\|^2 \right]^T \quad (24)$$

R_i is the variance of the measurement noise. P_i^* is the error covariance matrix, which is updated by

$$P_i^* = (I - K_i B_i^{*T}) P_{i-1}^* + qI \quad (25)$$

$P_i^* \subseteq P_i$, which includes the corresponding columns and rows related to the winner neuron. The size of P_i^* is $(2n_y + n_x + 1) \times (2n_y + n_x + 1)$.

Using EMRAN, the cycle time and its break-up for one sample of data are given in Tables 5 and 6 for problems BM-2 and BM-3, respectively. From the Tables, it can be seen that there is a large reduction in t_{c_4}, and all other times remain same. This reduction in Step 4 time results in a major reduction for the cycle time t_c for EMRAN.

To look at the computational overhead reduction clearly, the above data are also displayed in Fig. 8 in a bar graph form. A logarithmic scale has been used for this purpose because the difference between MRAN and EMRAN is large. For the BM-2 problem for a typical network with 30 neurons, for example, the cycle time for MRAN is 545 ms, whereas for EMRAN it is 6ms, a significant reduction. This reduction is more for the BM-3 problem, as for a network of 30 neurons MRAN takes 3568 ms, whereas EMRAN takes only 11.15 ms, a significant two-order reduction. This behaviour is observed for the network with all sizes (number of neurons). It is evident from Fig. 8 that EMRAN produces a large reduction in compu-

Table 5: Computation cycle time t_c (ms) for EMRAN (BM-2 problem)

Hidden units	5	10	15	20	25	30	35
t_{1c}	0.82	1.37	2.07	2.38	3.00	3.65	4.55
t_{2c}	0.003	0.004	0.004	0.004	0.005	0.005	0.005
t_{3c}	0.22	0.27	0.23	0.25	0.24	0.25	0.26
t_{4c}	1.67	1.50	1.51	1.42	1.48	1.57	1.54
t_{5c}	0.09	0.17	0.26	0.33	0.44	0.53	0.67
t_c	2.80	3.31	4.07	4.38	5.17	6.00	7.03

Table 6: Computation cycle time t_c (ms) for EMRAN (BM-3 problem)

Hidden units	5	10	15	20	25	30	35
t_{1c}	1.51	2.63	3.87	5.13	7.11	8.07	9.37
t_{2c}	0.005	0.005	0.008	0.008	0.009	0.009	0.01
t_{3c}	0.35	0.38	0.40	0.42	0.40	0.42	0.45
t_{4c}	2.12	2.04	2.03	1.95	2.19	2.11	2.05
t_{5c}	0.10	0.18	0.26	0.36	0.46	0.55	0.68
t_c	4.08	5.23	6.57	7.87	10.17	11.16	12.56

tation time, and also that as the number of hidden neurons increases to more than 20, the cycle time remains approximately flat.

5 Performance comparison of MRAN and EMRAN

In Section 4 we looked at cycle times of MRAN and EMRAN without looking at their performances, i.e. the identification accuracy of MRAN and EMRAN has not been compared. In this Section, the identification accuracy comparison for both problems BM-2 and BM-3 are carried out using MRAN and EMRAN.

5.1 BM-2: nonlinear SISO time-variant dynamic system

The network input and output vector is the same as before. The parameters for the EMRAN are selected as $E_1 = 0.01$, $E_2 = 0.09$, $E_3 = max\{\varepsilon_{max} \times \gamma^i, \varepsilon_{min}\}$, $\varepsilon_{max} = 1.25$, $\varepsilon_{min} = 0.4$, $\gamma = 0.999$. We use $\delta = 0.001$ as the pruning threshold, and the size of the two sliding windows (M, N_w) is 48.

Figs. 9 and 10 show the neuron history and error history for this system. Compared to 11 neurons for MRAN, 12 hidden neurons are used in EMRAN. In Fig. 11 the approximation error between the EMRAN and MRAN is close, and whenever there is a change in the dynamics of the system MRAN gives lower errors than EMRAN.

Table 7 presents a comparison of MRAN and EMRAN in terms of the network size, approximation errors and computational times for problems BM-2 and BM-3. The error I_{dav} in Table 7 is based on the average error index I_d for 10 000 samples and, for t_c, the network has (no more than) 15 hidden neurons.

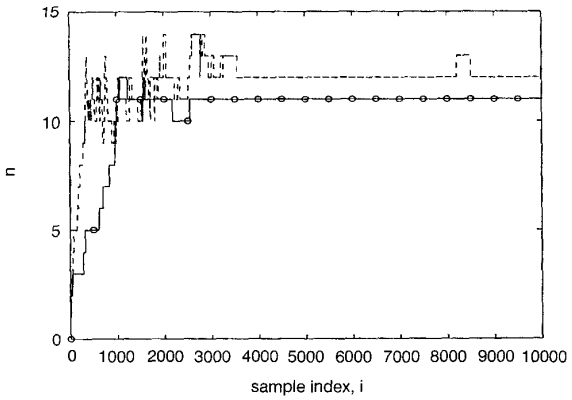

Fig. 9 *BM-2: evolution of hidden neurons (MRAN against EMRAN)*
---- EMRAN algorithm
-O-O-MRAN algorithm

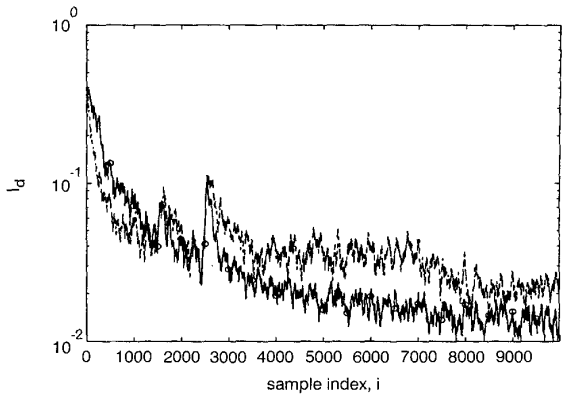

Fig. 10 *BM-2: evolution of error (I_d) (MRAN against EMRAN)*
---- EMRAN algorithm
-O-O-MRAN algorithm

5.2 BM-3: MIMO nonlinear dynamic system

After the online training, Fig. 12 shows the test result. We can also see from Fig. 13 that the adding and pruning capability of the EMRAN allows the RBF neural network to identify the high-dimension nonlinear system online. This time there are maximum 35 hidden neurons.

From the Tables it is clear that, with a slight increase of error in EMRAN, there is great advantage in computational time for EMRAN compared to MRAN.

Note that for both MRAN and EMRAN the parameters $E_1, E_2, E_3 \ldots$ etc. should be properly selected to ensure a good performance of the algorithm. Guidelines for selecting these thresholds as a trade-off between accuracy and network size have been presented previously [14]. For the above identification problems, the parameters were tuned largely by trial and error based on an initial sample size of 1 000, and then used further for online identification.

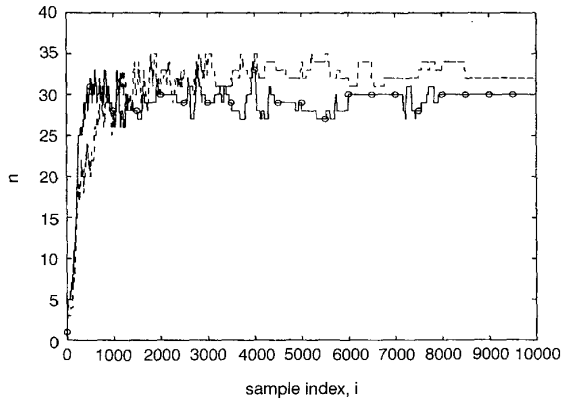

Fig. 11 *BM-3: evolution of hidden neurons (MRAN against EMRAN)*
---- EMRAN algorithm
-O-O-MRAN algorithm

Table 7: Comparison of identification results of MRAN and EMRAN

Performance	BM-2				BM-3			
	I'_{dav}	Hidden neurons	Overall time, s	t_c, ms	I'_{dav}	Hidden neurons	Overall time, s	t_c, ms
MRAN	0.0379	11	104.0	111	0.0349	30	28574	6219
EMRAN	0.0427	12	21.3	3.80	0.0392	32	54.8	12.56

10000 samples; t_c is calculated based on 35 hidden units;
overall time = total identification time for all samples

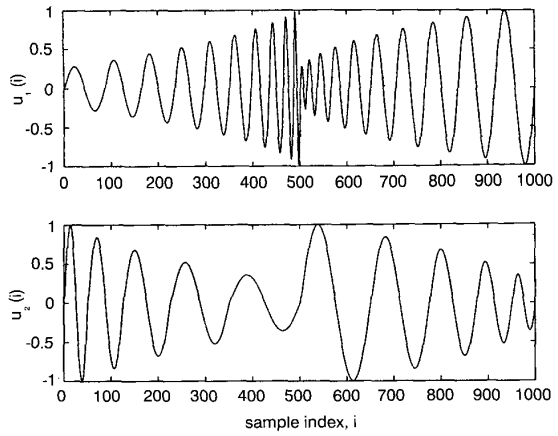

Fig. 12 *BM-3: test input signals*

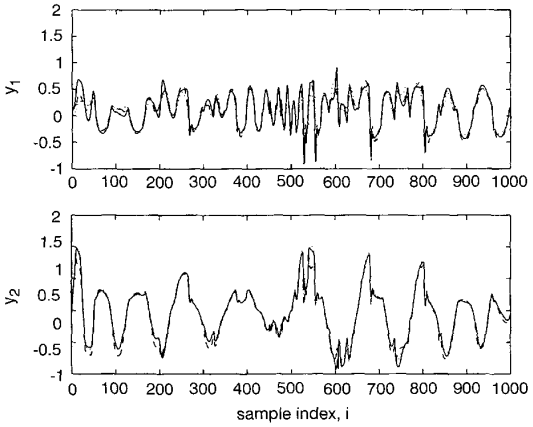

Fig. 13 *BM-3: test output data (MRAN against EMRAN)*

—— actual output
– – – MRAN output
····· EMRAN output

6 Conclusions

We have presented a performance analysis of the MRAN algorithm for online identification of nonlinear dynamic systems. Using nonlinear time-invariant and time-varying identification benchmark problems, MRAN's performance has been compared with the ONSAHL. The results indicate that MRAN realises networks with far fewer hidden neurons than ONSAHL and with better approximation accuracy.

The problems in real-time implementation of MRAN have been highlighted using detailed timing studies and analysis of the basic computations in MRAN. An extension to MRAN, EMRAN, which utilises a winner neuron strategy in MRAN, has been highlighted. This modification reduces the computation load for MRAN and leads to considerable reduction in the identification time, with only a slight increase in the approximation error. This also indicates the minimum sampling time we can select using EMRAN for identification problems. Using the same benchmark problems, the benefits of EMRAN show that, compared with other learning algorithms, EMRAN can 'adaptively track' the dynamics of the nonlinear system quickly without loss of accuracy, and is ideal for fast online identification of nonlinear plants.

7 References

1 AGARWAL, M.: 'A systematic classification of neural-network-based control', *IEEE Control Syst. Mag.*, 1997, **17**, (2), pp. 75–93
2 SADHUKHAN, D., and FETEIH, S.: 'F8 neurocontroller based on dynamic inversion', *AIAA J. Guid., Control, Dynam.*, 1996, **19**, (1), pp. 150–156
3 NARENDRA, K., and PARTHASARATHY, K.: 'Identification and control of dynamic systems using neural networks', *IEEE Trans., Neural Netw.*, 1990, **1**, (1), pp. 4–26
4 CHEN, S., BILLINGS, S.A., COUAN, C., and GRANT, P.M.: 'Practical identification of NARMAX models using radial basis function', *Int. J. Control*, 1990, **52**, pp. 1327–1350
5 CHEN, S., and BILLINGS, S.A.: 'Neural networks for system identification', *Int. J. Control*, 1992, **56**, pp. 319–346
6 CHEN, C.L., CHEN, W.C., and CHANG, F.Y.: 'Hybrid learning algorithm for Gaussian potential function networks', *IEE Proc., Control Theory Appl.*, 1993, **140**, (6), pp. 442–448
7 MOODY, J., and DARKEN, C.J.: 'Fast learning in network of locally tuned processing units', *Neural Computation*, 1989, **1**, pp. 281–294
8 LEE, S., and KIL, R.M.: 'A Gaussian potential function network with hierarchically self-organizing learning', *Neural Netw.*, 1991, **4**, pp. 207–224
9 MUSAVI, M.T., AHMED, W., CHAN, K.H., FARIS, K.B., and HUMMELS, D.M.: 'On training of radial basis function classifiers', *Neural Netw.*, 1992, **5**, pp. 595–603
10 PLATT, J.C.: 'A resource allocating network for function interpolation', *Neural Comput.*, 1991, **3**, pp. 213–225
11 KADIRKAMANATHAN, V., and NIRANJAN, M.: 'A function estimation approach to sequential learning with neural networks', *Neural Comput.*, 1993, **5**, pp. 954–975
12 LU, Y.W., SUNDARARAJAN, N., and SARATCHANDRAN, P.: 'A sequential learning scheme for function approximation and using minimal radial basis neural networks', *Neural Comput.*, 1997, **9**, (2), pp. 1–18
13 LU, Y.W., SUNDARARAJAN, N., and SARATCHANDRAN, P.: 'A sequential minimal radial basis function (RBF) neural network learning algorithm', *IEEE Trans., Neural Netw.*, 1998, **9**, (2), pp. 308–318
14 LU, Y.W., SUNDARARAJAN, N., and SARATCHANDRAN, P.: 'Identification of time- varying non-linear systems using minimal radial basis function neural networks', *IEE Proc., Control Theory Appl.*, 1997, **144**, (1), pp. 1–7
15 JUNGE, T.F., and UNBEHAUEN, H.: 'Off-line identification of nonlinear systems using structurally adaptive radial basis function networks'. Proc. 35th Conf. on *Decision and Control*, Kobe, Japan, pp. 943–948, December 1996
16 JUNGE, T.F., and UNBEHAUEN, H.: 'On-line identification of nonlinear time-variant systems using structurally adaptive radial basis function networks'. Proc. Amer. Control Conf., Albuquerque, New Mexico, pp. 1037–1041, 1997