Communicated by Scott Fahlman

A Sequential Learning Scheme for Function Approximation Using Minimal Radial Basis Function Neural Networks

Lu Yingwei
N. Sundararajan
P. Saratchandran
School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore

This article presents a sequential learning algorithm for function approximation and time-series prediction using a minimal radial basis function neural network (RBFNN). The algorithm combines the growth criterion of the resource-allocating network (RAN) of Platt (1991) with a pruning strategy based on the relative contribution of each hidden unit to the overall network output. The resulting network leads toward a minimal topology for the RBFNN.

The performance of the algorithm is compared with RAN and the enhanced RAN algorithm of Kadirkamanathan and Niranjan (1993) for the following benchmark problems: (1) hearta from the benchmark problems database PROBEN1, (2) Hermite polynomial, and (3) Mackey-Glass chaotic time series. For these problems, the proposed algorithm is shown to realize RBFNNs with far fewer hidden neurons with better or same accuracy.

1 Introduction

Radial basis function neural networks (RBFNN) are well suited for function approximation and pattern recognition due to their simple topological structure and their ability to reveal how learning proceeds in an explicit manner (Tao 1993). In the classical approach to radial basis function (RBF) network implementation, the basis functions are usually chosen as gaussian and the number of hidden units (i.e., centers and widths of the gaussian functions) is fixed a priori based on the properties of input data. The weights connecting the hidden and output units are estimated by a linear least squares method, as in the least mean square (LMS) method (Moody & Darken 1989; Musavi et al. 1992). The disadvantage with this approach is that it is not suitable for sequential learning and usually results in too many hidden units (Bors & Gabbouj 1994).

A significant contribution that overcomes these drawbacks was made by Platt (1991) through the development of an algorithm that adds hidden units to the network based on the novelty of the new data. The algorithm

is suitable for sequential learning and is based on the idea that the number of hidden units should correspond to the complexity of the underlying function as reflected in the observed data. The resulting network, called a resource-allocating network (RAN), starts with no hidden units and grows by allocating new hidden units based on the novelty in the observations that arrive sequentially. If an observation has no novelty, then the existing parameters of the network are adjusted by an LMS algorithm to fit that observation.

An interpretation of Platt's RAN from a function space approach was provided by Kadirkamanathan and Niranjan (1993), who also enhanced its performance by adopting an Extended Kalman Filter (EKF) instead of the LMS method for adjusting network parameters. They called the resulting network a Resource-Allocating Network via Extended Kalman Filter (RANEKF) and showed its superior performance to the RAN for the tasks of function approximation and time-series prediction (Kadirkamanathan & Niranjan 1993).

One drawback of both RAN and RANEKF is that once a hidden unit is created, it can never be removed. Because of this, both RAN and RANEKF could produce networks in which some hidden units, although active initially, may subsequently end up contributing little to the network output. If such inactive hidden units can be detected and removed as learning progresses, then a more parsimonious network topology can be realized. This has been the motivation for us to combine pruning to the RANEKF scheme. The following function approximation problem clearly illustrates how RANEKF could produce more numbers of hidden units than required.

1.1 Function Approximation Problem. In this problem, the RANEKF has to approximate a nonlinear function (Chen et al. 1993), which is a linear combination of six gaussian functions:

$$\begin{aligned} y(\mathbf{x}) = \quad & \exp[-\tfrac{(x_1-0.3)^2+(x_2-0.2)^2}{0.01}] + \exp[-\tfrac{(x_1-0.7)^2+(x_2-0.2)^2}{0.01}] \\ & + \exp[-\tfrac{(x_1-0.1)^2+(x_2-0.5)^2}{0.02}] + \exp[-\tfrac{(x_1-0.9)^2+(x_2-0.5)^2}{0.02}] \\ & + \exp[-\tfrac{(x_1-0.3)^2+(x_2-0.8)^2}{0.01}] + \exp[-\tfrac{(x_1-0.7)^2+(x_2-0.8)^2}{0.01}]. \end{aligned} \tag{1.1}$$

Theoretically we can say that to approximate this function, the RANEKF network should need only six hidden neurons. To test this, training samples are randomly chosen in the interval $(0, 1)$ and the observations sequentially presented to RANEKF. Figure 1 shows the results of applying the RANEKF algorithm to this problem. From Figure 1a, we can see the number of hidden neurons in the RANEKF network increasing with the number of observations and the network's finally settling to 15 hidden units instead of the desired 6 hidden units. Thus the RANEKF overfits and so the network realized by this method is not minimal. Figure 1b shows the desired 6 centers $(+)$ and the 15 centers produced by RANEKF $(\circ)$.

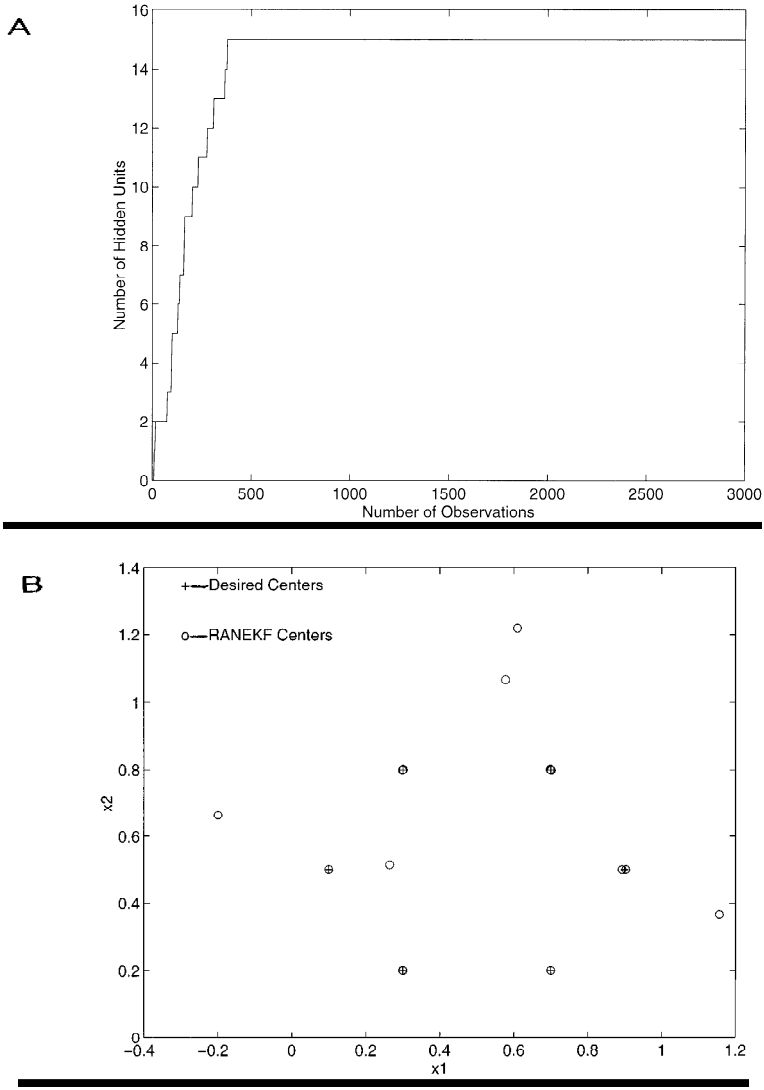

Figure 1: Number and positions of hidden unit centers achieved by RANEKF for the static function approximation problem.

Those hidden units in the network, which are far away from the desired centers, will have a significant output value only when the input is near their center. But for approximating the function of equation 1.1, a zero output is required for all inputs that are far from the desired centers. Therefore, the circles in Figure 1b, which are far from the desired centers, end up making little contribution to the network's outputs and should be removed from the network. We notice that corresponding to the desired centers (0.3,0.8), (0.7,0.8), (0.9,0.5), and (0.3,0.2), there are two overlapping circles. The two hidden units with the same center will provide no more information than one does, so one of them is superfluous and ought to be pruned. Generally, all those hidden units that contribute little to the network output are superfluous.

In this article we propose an algorithm that adopts the basic idea of RANEKF and augments it with a pruning strategy to obtain a minimal RBFNN network. The pruning strategy removes those hidden neurons that consistently make little contribution to the network output and together with an additional growth criterion ensures that the transition in the number of hidden neurons is smooth. Although theoretical proofs showing that the resulting RBFNN topology is in some sense minimal is still under investigation, we have referred it as a minimal RAN (M-RAN) based on detailed simulation studies carried out on a number of benchmark problems (Yingwei et al. 1995). For all those problems, the proposed approach produced a network with fewer hidden neurons than both the RAN and the RANEKF with the same or smaller approximation errors.

The article is organized as follows. Section 2 describes the the M-RAN algorithm. Sections 2.1 and 2.2 present the basic strategy for growth and pruning. The performance of M-RAN for the static function approximation problem of equation 1.1 is given in Section 2.3. An additional growth criterion based on the output error over a sliding window is introduced in Section 2.4 to smooth the transitions in the growth and pruning. Section 2.5 gives the final algorithm and its performance for the problem of equation 1.1. In Section 3 the performance of M-RAN is compared with RAN and RANEKF for two static function approximation problems (PROBEN1-**hearta** and Hermite polynomial) and the dynamic Mackey-Glass chaotic time-series prediction problem. Finally, conclusions are summarized in Section 4.

2 Proposed M-RAN Algorithm

In this section, a description of the M-RAN algorithm is presented. The notations and symbols used here are the same as those in Kadirkamanathan and Niranjan (1993).

2.1 Basic Growth Strategy. The basic growth strategy used in our algorithm is the same as that of RAN and RANEKF. The output of the network

for all three algorithms (RAN, RANEKF, and M-RAN) has the following form:

$$f(\mathbf{x}) = \alpha_0 + \sum_{k=1}^{K} \alpha_k \phi_k(\mathbf{x}) \,, \tag{2.1}$$

where $\phi_k(\mathbf{x})$ is the response of the kth hidden neuron to the input $\mathbf{x}$ and α_k is the weight connecting the kth hidden unit to the output unit. α_0 is the bias term. Here, K represents the number of hidden neurons in the network. $\phi_k(\mathbf{x})$ is a gaussian function given by

$$\phi_k(\mathbf{x}) = \exp\left(-\frac{1}{\sigma_k^2}\|\mathbf{x} - \mu_\mathbf{k}\|^2\right), \tag{2.2}$$

where $\mu_\mathbf{k}$ is the center and σ_k is the width of the gaussian function. $\|\|$ denotes the Euclidean norm.

The learning process of M-RAN involves allocation of new hidden units as well as adjusting network parameters. The network begins with no hidden units. As observations are received, the network grows by using some of them as new hidden units. The following two criteria must be met for an observation $(\mathbf{x_n}, y_n)$ to be used to add a new hidden unit to the network:

$$\|\mathbf{x_n} - \mu_\mathbf{nr}\| > \epsilon_n \,. \tag{2.3}$$

$$e_n = y_n - f(\mathbf{x_n}) > e_{min} \,, \tag{2.4}$$

where $\mu_\mathbf{nr}$ is the center (of the hidden unit) closest to $\mathbf{x_n}$. ϵ_n and e_{min} are thresholds to be selected appropriately. When a new hidden unit is added to the network, the parameters associated with the unit are $\alpha_{K+1} = e_n$, $\mu_{\mathbf{K+1}} = \mathbf{x_n}$, $\sigma_{K+1} = \kappa\|\mathbf{x_n} - \mu_\mathbf{nr}\|$. κ is an overlap factor, which determines the overlap of the responses of the hidden units in the input space.

When the observation $(\mathbf{x_n}, y_n)$ does not meet the criteria for adding a new hidden unit, the network parameters $\mathbf{w} = [\alpha_0, \alpha_1, {\mu_\mathbf{1}}^T, \sigma_1, \ldots, \alpha_K, {\mu_\mathbf{K}}^T, \sigma_K]^T$ are adapted using the EKF as follows:

$$\mathbf{w_n} = \mathbf{w_{(n-1)}} + e_n \mathbf{k_n} \,. \tag{2.5}$$

$\mathbf{k_n}$ is the Kalman gain vector given by

$$\mathbf{k_n} = [R_n + {\mathbf{a_n}}^T P_{n-1} \mathbf{a_n}]^{-1} P_{n-1} \mathbf{a_n} \,. \tag{2.6}$$

$\mathbf{a_n}$ is the gradient vector and has the following form:

$$\mathbf{a_n} = \left[1, \phi_1(\mathbf{x_n}), \phi_1(\mathbf{x_n})\frac{2\alpha_1}{\sigma_1^2}(\mathbf{x_n} - \mu_\mathbf{1})^T, \phi_1(\mathbf{x_n})\frac{2\alpha_1}{\sigma_1^3}\|\mathbf{x_n} - \mu_\mathbf{1}\|^2, \ldots,\right.$$

$$\left.\phi_K(\mathbf{x_n}), \phi_K(\mathbf{x_n})\frac{2\alpha_K}{\sigma_K^2}(\mathbf{x_n} - \mu_\mathbf{K})^T, \phi_K(\mathbf{x_n})\frac{2\alpha_K}{\sigma_K^3}\|\mathbf{x_n} - \mu_\mathbf{K}\|^2\right]^T. \quad (2.7)$$

R_n is the variance of the measurement noise. P_n is the error covariance matrix, which is updated by

$$P_n = [I - \mathbf{k_n}\mathbf{a_n}^T]P_{n-1} + QI\,, \quad (2.8)$$

where Q is a scalar that determines the allowed random step in the direction of gradient vector.[1] If the number of parameters to be adjusted is N, P_n is an $N \times N$ positive definite symmetric matrix. When a new hidden unit is allocated, the dimensionality of P_n increases to

$$P_n = \begin{pmatrix} P_{n-1} & 0 \\ 0 & P_0 I \end{pmatrix}, \quad (2.9)$$

and the new rows and columns are initialized by P_0, which is an estimate of the uncertainty in the initial values assigned to the parameters. The dimension of identity matrix I is equal to the number of new parameters introduced by the new hidden unit.

2.2 Pruning Strategy. This section presents a scheme to prune superfluous hidden units in the M-RAN network. To remove the hidden units that make little contribution to the network output, first consider the output o_k of the hidden unit k:

$$o_k = \alpha_k \exp\left(-\frac{\|\mathbf{x} - \mu_\mathbf{k}\|^2}{\sigma_k^2}\right). \quad (2.10)$$

If α_k or σ_k in equation 2.10 is small, o_k might become small. Alternately if $\|\mathbf{x} - \mu_\mathbf{k}\|$ is large, which means the input is far from the center of this hidden unit, the output becomes small. In order to decide whether a hidden neuron should be removed, the output values (o_k) for the hidden units are examined continuously. If the output of a hidden unit is less than a threshold over a number of M consecutive inputs, then that hidden unit is removed from

[1] According to Kadirkamanathan and Niranjan (1993), the rapid convergence of the EKF algorithm may prevent the model from adapting to future data, and QI is introduced to avoid this problem.

the network.[2] Since the use of absolute values can cause inconsistency in pruning, the outputs of hidden units are normalized, and the normalized output values are used in the pruning criterion. This pruning strategy is expressed as follows:

- For every observation (x_n, y_n), compute the outputs of all hidden units o_k^n $(k = 1, \ldots, K)$, using equation 2.10.
- Find the largest absolute hidden unit output value $\|o_{max}^n\|$. Compute the normalized output values r_k^n, $(k = 1, \ldots, K)$, for the hidden units where $r_k^n = \|\frac{o_k^n}{o_{max}^n}\|$.
- Remove the hidden units for which the normalized output is less than a threshold δ for M consecutive observations.
- Adjust the dimensionality of the EKF to suit the reduced network.

2.3 Performance of the Pruning Strategy for Function Approximation. The pruning strategy is added to the basic strategy of Section 2.1, and the resulting algorithm is tested on the function approximation problem of equation 1.1. Figure 2 shows the results of the algorithm. It can be clearly seen that the proposed pruning strategy helps to reduce the number of hidden neurons to 6 after 4500 samples. It can also be observed that the algorithm causes the number of hidden neurons to increase by 13 initially (0–600 samples), after which pruning begins to take effect and brings the number of hidden units to 7 at around 1000 samples, and stays with 7 until 4500 samples before reaching the correct value of 6 hidden units. This shows that pruning helps to realize a minimal RBFNN, although the transitions in the number of hidden neurons are not always smooth. Also the minimum network was realized after 4500 samples, which is rather long. This motivated us to include an extra criterion for growth to smooth the transitions.

2.4 Sliding Window RMS Criterion for Adding Hidden Neurons. Besides the novelty criteria given by equations 2.3 and 2.4, we include a third criterion, which uses the root mean square (RMS) value of the output error over a sliding data window before adding a hidden unit. The RMS value of network output error at nth observation e_{rmsn} is given by

$$e_{rmsn} = \sqrt{\frac{\sum_{i=n-(M-1)}^{n} (e_i^2)}{M}}, \tag{2.11}$$

[2] For detailed analysis of this pruning criterion, refer to Yingwei et al. (1995).

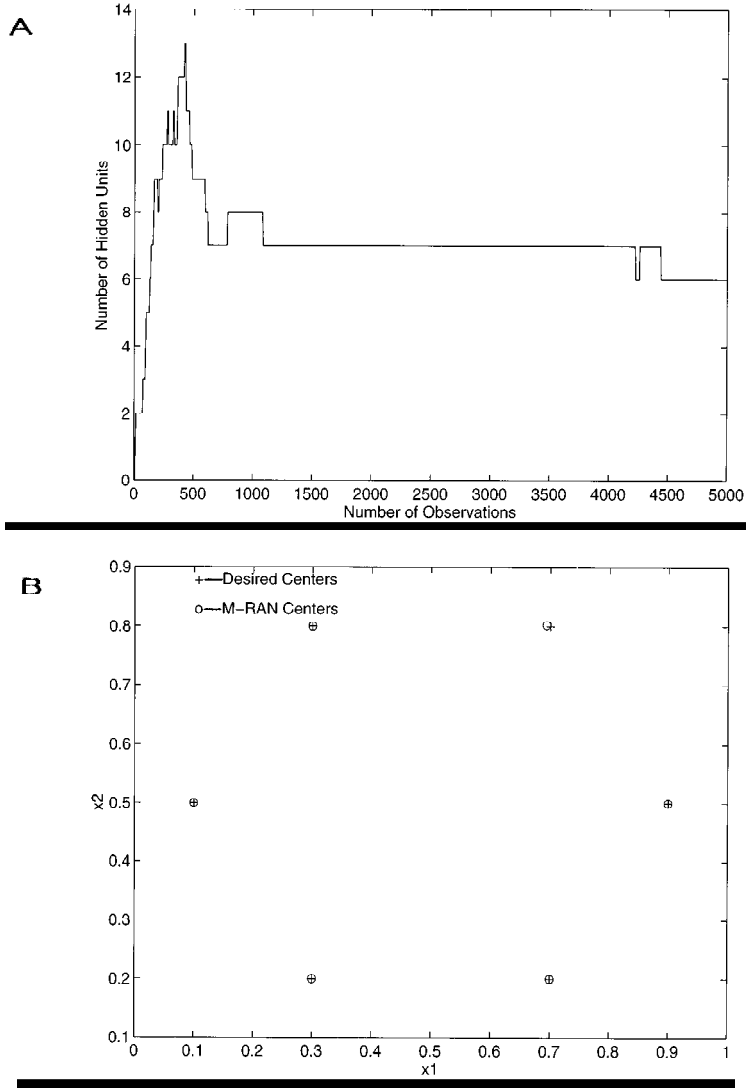

Figure 2: Number and positions of hidden unit centers obtained with pruning strategy on static function approximation problem.

where $e_i = y_i - f(\mathbf{x}_i)$. In equation 2.11, for each observation, a sliding data window is employed to include a certain number of output errors based on $y_n, y_{n-1}, \ldots, y_{n-(M-1)}$. When a new observation arrives, the data in this window are updated by including the latest data and eliminating the oldest. For example, at the nth observation, the data window includes $e_n, e_{n-1}, \ldots, e_{n-(M-1)}$; when the $(n+1)th$ observation arrives, the data window will contain $e_{n+1}, e_n, \ldots, e_{n-(M-2)}$. In this sense, it seems that this data window slides along with the data obtained on-line; thus we call it a data sliding window. The size of the window is determined by the value of M—that is, the number of samples used to calculate the RMS error. Thus, the extra growth criterion to be satisfied is $e_{rmsn} > e'_{min}$, where e'_{min} is a threshold value to be selected.

2.5 The Final M-RAN Algorithm. In this section, the final M-RAN sequential learning algorithm is presented.

Algorithm. For each input $\mathbf{x_n}$, compute:

$$\phi_k(\mathbf{x_n}) = \exp\left(-\frac{1}{\sigma_k^2}\|\mathbf{x_n} - \mu_\mathbf{k}\|^2\right),\ f(\mathbf{x_n}) = \alpha_0 + \sum_{k=1}^{K} \alpha_k \phi_k(\mathbf{x_n}),$$

$\epsilon_n = max\{\epsilon_{max}\gamma^n, \epsilon_{min}\}$, where $0 < \gamma < 1$ is a decay constant.[3]

$$e_n = y_n - f(\mathbf{x_n}),\ e_{rmsn} = \sqrt{\frac{\sum_{i=n-(M-1)}^{n}[y_i - f(\mathbf{x_i})]^2}{M}}.$$

If $e_n > e_{min}$, and $\|\mathbf{x_n} - \mu_\mathbf{nr}\| > \epsilon_n$, and $e_{rmsn} > e'_{min}$, then allocate a new hidden unit with $\alpha_{k+1} = e_n$, and $\mu_{\mathbf{k+1}} = \mathbf{x_n}$, and $\sigma_{k+1} = \kappa\|\mathbf{x_n} - \mu_\mathbf{nr}\|$.
Else

$$\mathbf{w_n} = \mathbf{w_{(n-1)}} + \mathbf{k_n} e_n,$$

$$\mathbf{k_n} = [R_n + \mathbf{a_n}^T P_{n-1} \mathbf{a_n}]^{-1} P_{n-1} \mathbf{a_n},$$

$$P_n = [I - k_n \mathbf{a_n}^T] P_{n-1} + QI.$$

Compute the outputs of all hidden units o_k^n, $(k = 1, \ldots, K)$.
Find the largest absolute hidden unit output value $\|o_{max}^n\|$.
Calculate the normalized value for each hidden unit

$$r_k^n = \left\| \frac{o_k^n}{o_{max}^n} \right\|, (k = 1, \ldots, K)$$

if $r_k^n < \delta$ for M consecutive observations, then prune the kth hidden neuron and reduce the dimensionality of P_n to fit for the requirement of EKF.

[3] The value for ϵ_n is decayed until it reaches ϵ_{min}.

This final algorithm is tested on the function approximation problem of section 1.1, and the results are presented in Figure 3. From Figure 3, it can be seen that the 6 hidden units on the desired positions are obtained within 600 samples. Also, the transitions during the build-up and drop-off of the hidden neurons are smooth compared to Figure 2.

3 Performance of the Algorithm on Benchmark Problems

In this section, the sequential learning strategy described in Section 2 is applied to three benchmark problems. The first two are static function approximation problems, and the third is a time-series prediction problem. The first problem is taken from the benchmark problems PROBEN1 (Prechelt 1994), set up recently and available from http://www.ipd.ira.uka.de/~prechelt/NIPS_bench.html. The problem selected for this study is the **hearta** data set, which is based on the "heart disease" diagnosis data sets from the University of California, Irvine, machine learning data set. The second problem is to approximate the Hermite polynomial, and the data in this case are randomly sampled consistent with the underlying function and presented sequentially to the network. The third example is on-line prediction of a chaotic time series given by the Mackey-Glass equation where an underlying function for the data does not exist. For all these problems, the performance of our algorithm is compared with the RANEKF and the RAN.

3.1 Function Approximation Problem: PROBEN1-hearta Benchmark. The **hearta** data set consists of 920 examples (690 training and 230 test examples) with each example described by 13 input attributes and 1 output attribute. Out of the 920 examples, only 299 are complete; the rest have missing values. This makes **hearta** a difficult function approximation problem. Our networks for RAN, RANEKF, and M-RAN used 13 input units and a single continuous output to represent the five different levels of 1 output attribute. According to the guidelines given in Prechelt (1994), the missing values were replaced with the mean of the nonmissing values for that attribute. The training data for the networks consist of both the training set and validation set, and the approximation error is calculated over the test data provided in the data set. The performance of the three algorithms is evaluated on the three different permutations of the **hearta** data set—**hearta1, hearta2**, and **hearta3**—available in PROBEN1. The values for the various parameters used in this experiment are as follows: $\epsilon_{max} = 4.0$, $\epsilon_{min} = 1.5$, $\gamma = 0.99$, $e_{min} = 0.1$, $\kappa = 0.4$, $P_0 = R_n = 1.0$, $Q = 0.00001$, and $\eta = 0.01$. The threshold value $\delta = 0.00001$, the size of sliding data window $M = 60$, and the threshold value of the RMS error in the additional growth criterion $e'_{min} = 0.25$.

Figure 4a shows the growth pattern for the three networks as they learn sequentially from the training data. Figure 4b shows the approximation error, measured by the given squared error percentage (E) in Prechelt (1994),

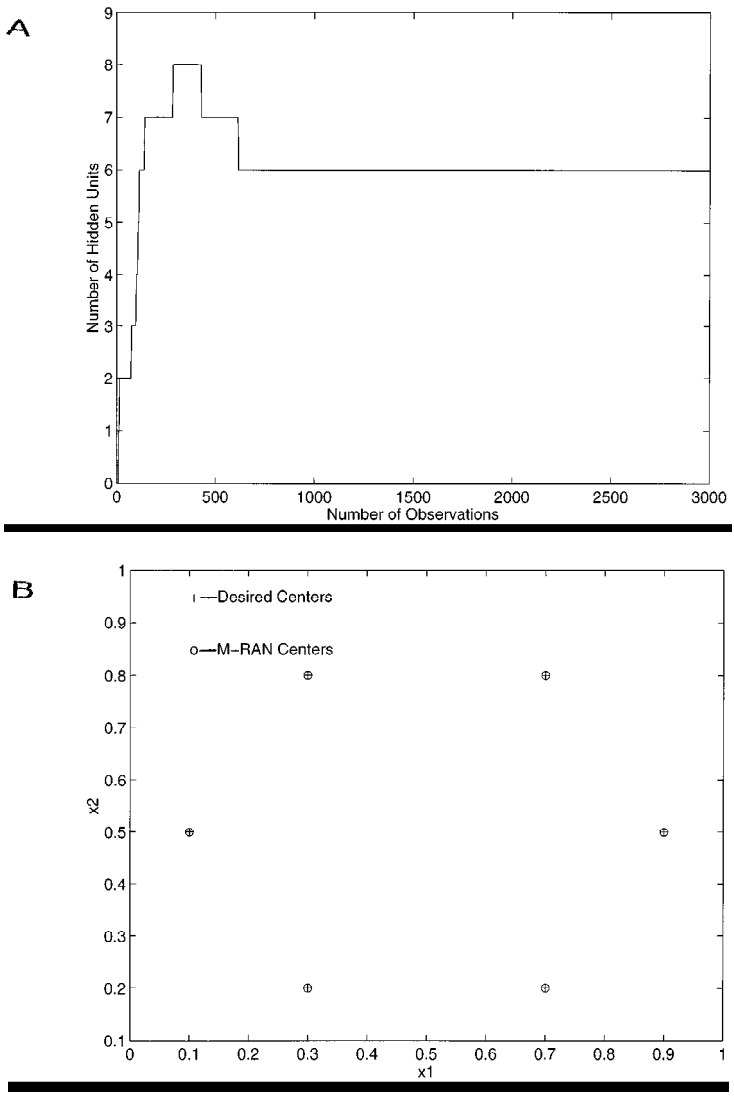

Figure 3: Number and positions of hidden unit centers achieved with M-RAN.

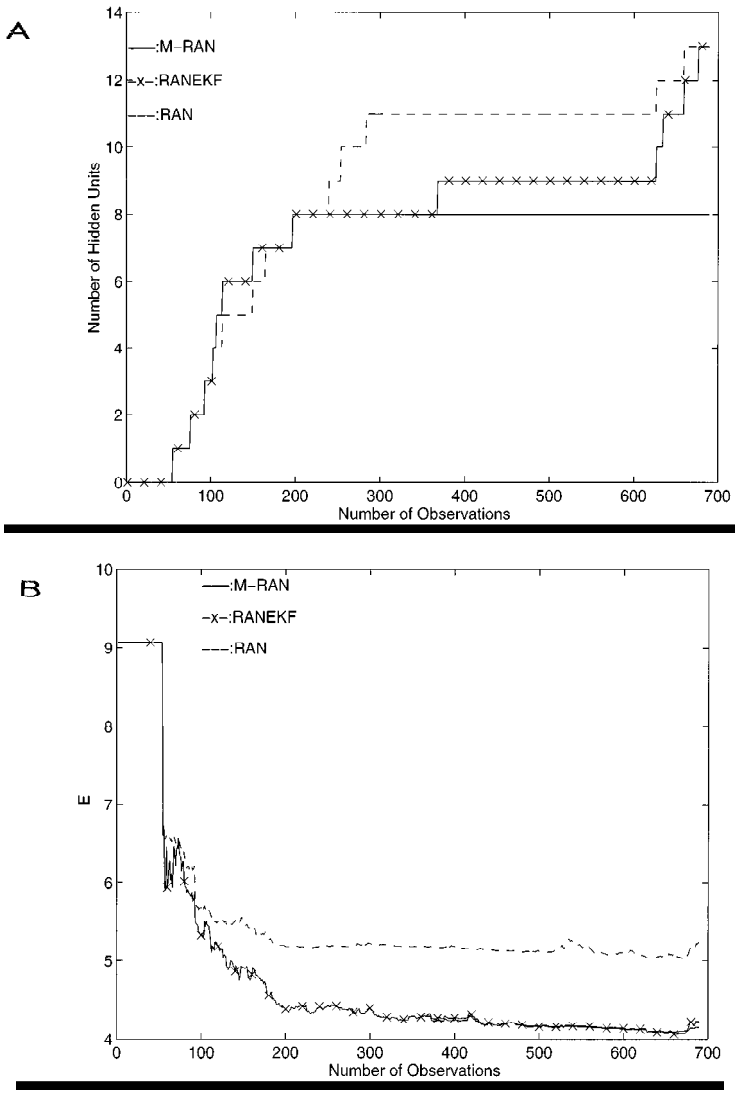

Figure 4: Performance of RAN, RANEKF, and M-RAN for **hearta1**.

Table 1: Performance of Different Networks for **hearta1**, **hearta2**, and **hearta3**.

Benchmark data	Network	Number of hidden neurons	Squared error percentage for testing set	Number of training epochs
hearta1	M-RAN	8	4.14	1
	RANEKF	13	4.21	1
	RAN	13	5.20	1
	BP	32	4.55	47
hearta2	M-RAN	9	3.81	1
	RANEKF	11	3.81	1
	RAN	11	5.71	1
	BP	16	4.33	54
hearta3	M-RAN	7	3.80	1
	RANEKF	11	3.91	1
	RAN	14	5.63	1
	BP	32	4.89	46

over the test data. E is expressed as follows:

$$E = 100 * \frac{o_{max} - o_{min}}{N * P} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_{pi} - t_{pi})^2, \tag{3.1}$$

where o_{min} and o_{max} are the minimum and the maximum values of the output coefficients in the problem. N is the number of output nodes of the network, and P is the number of patterns (examples) in the data set considered. o_{pi} and t_{pi} are the actual and desired output values at the ith output node and pth pattern, respectively.

Table 1 shows the results from the three algorithms together with the performance of multilayer backpropagation (BP) networks for pivot architectures (Prechelt 1994). It can be seen from the table that our algorithm achieves the same or a lower approximation error than RAN and RANEKF and BP with fewer hidden units. The values quoted for pivot architectures are from Table 11 of Prechelt (1994).

3.2 Function Approximation Problem: **Hermite Polynomial.** In this section, results for approximating the Hermite Polynomial are presented. The Hermite Polynomial is given by the following equation:

$$f(x) = 1.1(1 - x + 2x^2) \exp\left(-\frac{1}{2}x^2\right). \tag{3.2}$$

A random sampling of the interval $[-4, 4]$ is used in obtaining 40 input-output data for the training set. All parameter values are chosen the same as

those in Kadirkamanathan and Niranjan (1993)—that is, $\epsilon_{max} = 2.0, \epsilon_{min} = 0.2, \gamma = 0.977, e_{min} = 0.02, \kappa = 0.87, P_0 = R_n = 1.0, Q = 0.02, \eta = 0.05$. The additional parameters required for our algorithm are as follows: the threshold value $\delta = 0.01$, the size of sliding data window $M = 25$, and the threshold value of the RMS error in the additional growth criterion $e'_{min} = 0.3$.

Figure 5 shows the results from the three algorithms. It can be seen from the figure that our algorithm achieves a slightly lower approximation error, measured by the RMS error[4] calculated over 200 uniformly sampled data, than RANEKF with only 7 hidden units compared to the 13 hidden units of RANEKF. RAN needed 18 hidden units and achieved an RMS error of nearly an order of magnitude larger than both our algorithm and the RANEKF. With no noise in the training set, the M-RAN needed only about half the number of hidden units of RANEKF.

The effect of noise on the performance of the three algorithms was analyzed by adding different levels of gaussian white noise to the training patterns, and here too M-RAN always realized more compact networks with smaller approximation errors (Yingwei et al. 1995). As an example, when the noise variance was 0.07, the number of hidden units for M-RAN was 11, for RANEKF 40, and for RAN 20.

3.3 Prediction of Chaotic Time Series. The chaotic time series used is the Mackey-Glass series, which is governed by the following time-delay ordinary differential equation:

$$\frac{ds(t)}{dt} = -bs(t) + a\frac{s(t-\tau)}{1+s(t-\tau)^{10}}, \tag{3.3}$$

with $a = 0.2, b = 0.1$, and $\tau = 17$. Integrating the equation over the time interval $[t, t+\Delta t]$ by the trapezoidal rule yields

$$x(t+\Delta t) = \frac{2-b\Delta t}{2+\Delta t}x(t) + \frac{a\Delta t}{2+b\Delta t}\left[\frac{x(t+\Delta t-\tau)}{1+x^{10}(t+\Delta t-\tau)} + \frac{x(t-\tau)}{1+x^{10}(t-\tau)}\right]. \tag{3.4}$$

Setting $\Delta t = 1$, the time series is generated under the condition $x(t-\tau) = 0.3$ for $0 \le t \le \tau$ ($\tau = 17$). The initial 4000 data points are discarded for the initialization transients to decay.[5]

The series is predicted with $v = 50$ sample steps ahead using four past samples: $s_{n-v}, s_{n-v-6}, s_{n-v-12}, s_{n-v-18}$. Hence, the nth input-output data for

[4] RMSE is used for comparison to be consistent with Kadirkamanathan and Niranjan (1993).

[5] These conditions are the same as those in Platt (1991).

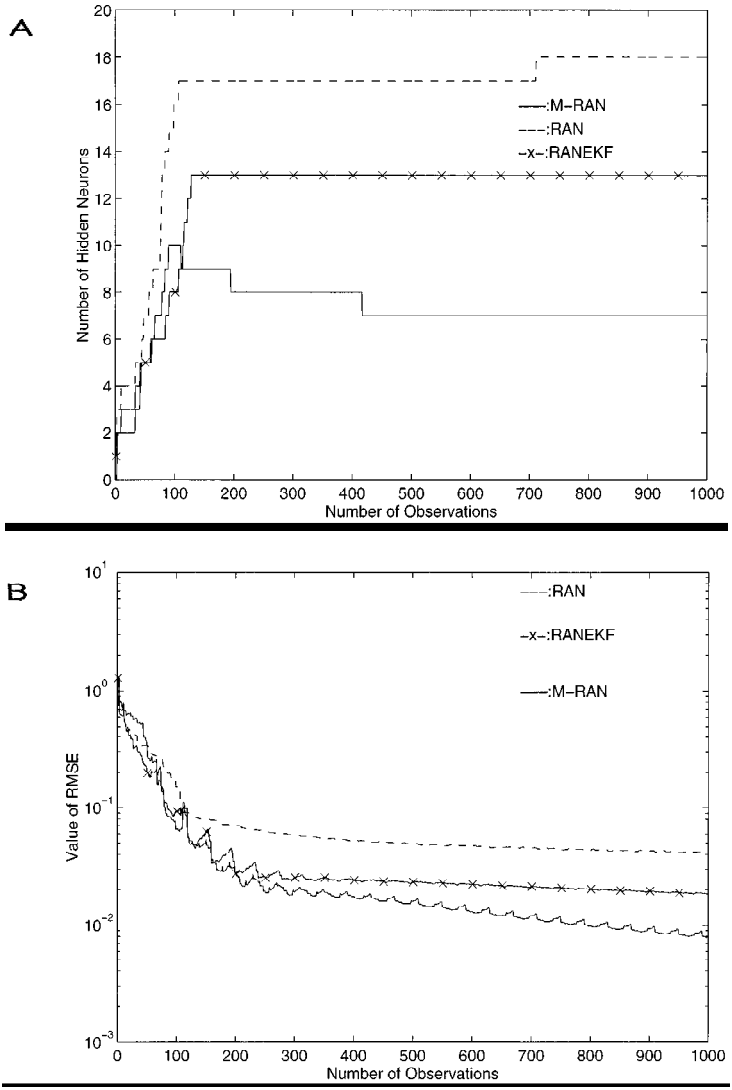

Figure 5: Performance of RAN, RANEKF, and M-RAN for Hermite Polynomial without noise.

the network to learn are

$$\mathbf{x_n} = [s_{n-\nu}, s_{n-\nu-6}, s_{n-\nu-12}, s_{n-\nu-18}]^T,$$

$$y_n = s_n$$

whereas the step-ahead predicted value at time $n - \nu$ is given by

$$z_n = f^{(n-\nu)}(\mathbf{x}_n), \tag{3.5}$$

where $f^{(n-\nu)}(\mathbf{x}_n)$ is the network at time $(n - \nu)$. The ν step-ahead prediction error is

$$\epsilon_n = s_n - z_n. \tag{3.6}$$

The exponentially weighted prediction error (WPE) is chosen as a measure of the performance of the network prediction. WPE at time (n) can be recursively computed as

$$WPE^{(n)^2} = \lambda WPE^{(n-1)^2} + (1 - \lambda)\|\epsilon_n\|^2, \tag{3.7}$$

where $0 < \lambda < 1$. Here λ is chosen as $\lambda = 0.95$. The parameter values are selected as follows: $\epsilon_{max} = 0.7, \epsilon_{min} = 0.07, \gamma = 0.999, e_{min} = 0.05, \kappa = 0.87, P_0 = R_n = 1.0, Q = 0.0002, \eta = 0.02, \delta = 0.0001; M = 90$; and $e'_{min} = 0.04$.

Figure 6c displays the approximation curve of the M-RAN during the sample time between 4500 and 5000. The desired output curve is plotted with a slash-dotted line, while the output of the minimal network is plotted with a solid line. This figure shows that the approximation curve of our M-RAN network fits the desired curve at most places. The number of hidden units and the network performance are shown in Figures 6a and 6b, respectively. In this case, the value of WPE is almost same for all three algorithms, while the M-RAN needed only 29 hidden units, compared with 39 hidden units achieved by RANEKF, and 81 hidden units of RAN.

4 Conclusions

In this article, a sequential learning algorithm for realizing a minimal RBF neural network is developed. This algorithm augments the growth criteria of RANEKF and combines with it a scheme to remove those hidden units that make insignificant contributions to the output of the network.

Using benchmark problems covering both static (**hearta**, Hermite Polynomial) and dynamic (prediction for a chaotic time series) approximations, the algorithm developed is shown to produce an RBF neural network with smaller complexity than both RANEKF and RAN, with equal or better approximation accuracy.

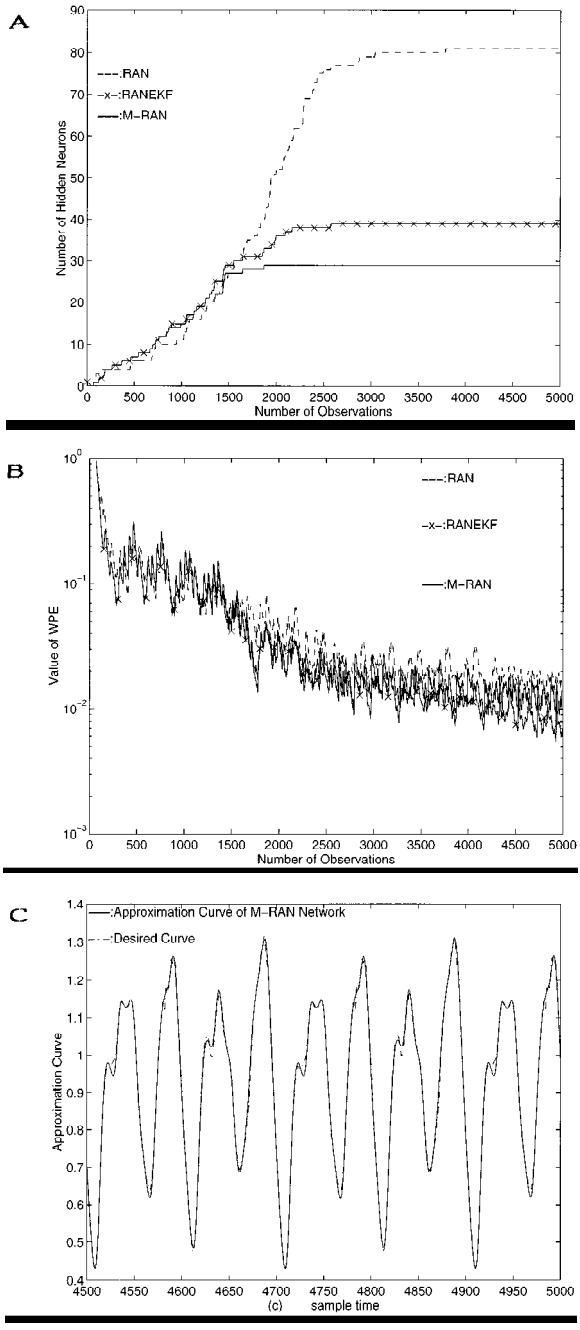

Figure 6: Performance of RAN, RANEKF, and M-RAN for Mackey-Glass time-series prediction problem.

Acknowledgments

We thank the anonymous reviewers for their constructive comments and also for pointing out the PROBEN1 benchmark database.

References

Bors, A. G., & Gabbouj, M. (1994). Minimal topology for a radial basis functions neural network for pattern classification. *Digital Signal Processing*, **4**, 173–188.

Chen, C., Chen, W., & Cheng, F. (1993). Hybrid learning algorithm for Gaussian potential function network. *IEE Proceedings-D*, **140**, 442–448.

Kadirkamanathan, V., & Niranjan, M. (1993). A function estimation approach to sequential learning with neural network. *Neural Computation*, **5**, 954–975.

Moody, J., & Darken, C. J. (1989). Fast learning in network of locally-tuned processing units. *Neural Computation*, **1**, 281–294.

Musavi, M., Ahmed, W., Chan, K., Faris, K., & Hummels, D. (1992). On training of radial basis function classifiers. *Neural Networks*, **5**, 595–603.

Platt, J. (1991). A resource allocating network for function interpolation. *Neural Computation*, **3**, 213–225.

Prechelt, L. (1994). Proben1—a set of neural network benchmark problems and benchmarking rules. (Tech. Rep. 21/94). Fakultat fur Informatik, University Karlsruhe.

Tao, K. (1993). A closer look at the radial basis function (RBF) networks. In A. L. A. Singh (Ed.), *Conference Record of 27th Asilomar Conference on Signals, System and Computers*, pp. 401–405. New York: IEEE.

Yingwei, L., Sundararajan, N., & Saratchandran, P. (1995). A sequential learning scheme for fuction approximation using minimal radial basis function (RBF) neural networks. (Tech. Rep. CSP/9505). Singapore: Center for Signal Processing, School of Electrical and Electronic Engineering, Nanyang Technological University.

Received December 11, 1995; accepted June 21, 1996.