An Efficient Sequential Learning Algorithm for Growing and Pruning RBF (GAP-RBF) Networks

Guang-Bin Huang, *Senior Member, IEEE*, P. Saratchandran, *Senior Member, IEEE*, and Narasimhan Sundararajan, *Fellow, IEEE*

***Abstract*—This paper presents a simple sequential growing and pruning algorithm for radial basis function (RBF) networks. The algorithm referred to as growing and pruning (GAP)-RBF uses the concept of "Significance" of a neuron and links it to the learning accuracy. "Significance" of a neuron is defined as its contribution to the network output averaged over all the input data received so far. Using a piecewise-linear approximation for the Gaussian function, a simple and efficient way of computing this significance has been derived for uniformly distributed input data. In the GAP-RBF algorithm, the growing and pruning are based on the significance of the "nearest" neuron. In this paper, the performance of the GAP-RBF learning algorithm is compared with other well-known sequential learning algorithms like RAN, RANEKF, and MRAN on an artificial problem with uniform input distribution and three real-world nonuniform, higher dimensional benchmark problems. The results indicate that the GAP-RBF algorithm can provide comparable generalization performance with a considerably reduced network size and training time.**

***Index Terms*—Growing and pruning (GAP-RBF), radial basis function (RBF) networks, sequential learning.**

I. Introduction

RADIAL BASIS function (RBF) networks offer an efficient mechanism for approximating complex nonlinear mappings from the input–output data. Selection of a learning algorithm for a particular application is dependent on its accuracy and speed [1]–[5]. Sequential learning algorithms are better than batch learning algorithms as they do not require retraining whenever a new data is received. A significant contribution to sequential learning algorithm was made by Platt [1] through the development of a resource allocation network (RAN), in which hidden neurons were added sequentially based on the novelty of the new data. Enhancement of RAN, known as a RAN extended Kalman filter (RANEKF), was proposed by Kadirkamanathan and Niranjan [2] in which an extended Kalman filter (EKF), rather than the least-mean square (LMS) algorithm, was used for updating the network parameters. A significant improvement to RANEKF was made by Yingwei *et al.* [4] by introducing a pruning strategy based on the relative contribution of each hidden neuron to the overall network output. The resulting network referred to as MRAN has been used in a number of applications [6]. Other methods for pruning in RBF networks have been proposed in Salmerón *et al.* [7], [8] and Rojas *et al.* [9].

Manuscript received January 30, 2004; revised May 7, 2004. This paper was recommended by Associate Editor J. B. Oommen.

The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (e-mail: egbhuang@ntu.edu.sg; epsarat@ntu.edu.sg; ensundara@ntu.edu.sg).

Digital Object Identifier 10.1109/TSMCB.2004.834428

It is interesting to note that none of the above algorithms link the required accuracy directly to the learning algorithm. Instead, they all have various thresholds which have to be selected using exhaustive trial-and-error studies. This issue of specifying the various thresholds based on the needed accuracy has been raised in MRAN for determining the inactive neurons.

This paper attempts to directly link the desired accuracy to the significance of neurons and uses it in the learning algorithm to realize a compact RBF network. By significance of a neuron, we mean the contribution made by that neuron to the network output averaged over all the input data received so far. This requires the knowledge of the input data distribution. In this paper the growing and pruning (GAP)-RBF algorithm is derived based on significance of a neuron when the input data is uniformly distributed.

Furthermore, the calculation of the significance here is simplified by using a piecewise linear approximation to the Gaussian function, thereby reducing the computational efforts. In GAP-RBF algorithm, this significance is used in growing and pruning strategies. For growing (apart from the novelty and distance criteria used in MRAN), this algorithm uses the criterion based on the significance of the neuron (instead of the root mean square (RMS) criterion in MRAN). If the significance is more than the chosen learning accuracy, only then a neuron will be added. If the significance is less than the learning accuracy, then that neuron will be pruned. Furthermore, for both growing and pruning, it is shown that one needs to check only the nearest neuron (based on the Euclidean distance to the current input data) for its significance. If the input data does not require a new hidden neuron to be added, then the parameters of only the nearest neuron are adjusted, resulting in a reduction in the overall computations and thereby increasing the learning speed. If the significance is less than the learning accuracy, then that neuron will be pruned.

In this paper, the performance of the GAP-RBF learning algorithm is compared with other well-known sequential learning algorithms like RAN, RANEKF and MRAN on some real benchmark problems in the function approximation area. A comparison with the support vector regression (SVR) is also included as SVR is a popular (although not sequential) kernel based method. The results indicate that GAP-RBF algorithm can provide comparable generalization performance with reduced computational complexity.

The paper is organized as follows. Section II describes the principal ideas behind the GAP-RBF learning algorithm, followed by a summary of the algorithm. Section III presents a critical comparison of GAP-RBF with other popular algorithms.

Section IV presents quantitative performance comparisons for GAP-RBF with the other algorithms based on the benchmark problems. Section V summarizes the conclusions from this study.

II. Principal Ideas Behind the GAP-RBF Learning Algorithm

In this section, the main ideas behind the GAP-RBF for growing and pruning are described.

The output of a RBF network with K neurons is given by

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

where $\phi_k(\mathbf{x})$ is the response of the kth hidden neuron for an input vector $\mathbf{x} \in \mathbf{R}^l$

$$\phi_k(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}-\mu_k\|^2}{\sigma_k^2}\right). \tag{2}$$

α_k is its connecting weight to the output neuron, and $\mu_k \in \mathbf{R}^l$ and σ_k are the center and width of the kth hidden neuron, respectively, $k = 1, \cdots, K$. The algorithm introduces the notion of *significance* for the hidden neurons based on their average contribution over all inputs seen so far. This is described in the following paragraph.

In sequential learning, a series of training samples $(\mathbf{x}_i, y(\mathbf{x}_i))$, $i = 1, 2, \cdots$ are randomly drawn and presented one by one to the network. We assume that these input samples $\mathbf{x}_i$ have a uniform distribution and the range for the input is X. After sequentially learning n observations, assume that a RBF network with K neurons has been obtained. The network output for an input $\mathbf{x}_i$ is given by

$$f_1(\mathbf{x}_i) = \sum_{j=1}^{K} \alpha_j \phi_j(\mathbf{x}_i). \tag{3}$$

If the neuron k is removed, the output of the RBF network with the remaining $K-1$ neurons is

$$f_2(\mathbf{x}_i) = \sum_{j=1}^{k-1} \alpha_j \phi_j(\mathbf{x}_i) + \sum_{j=k+1}^{K} \alpha_j \phi_j(\mathbf{x}_i). \tag{4}$$

Thus, for an observation $\mathbf{x}_i$, the error resulting from removing neuron k is the absolute difference between $f_1(\mathbf{x}_i)$ and $f_2(\mathbf{x}_i)$; that is

$$E(k,i) = |f_1(\mathbf{x}_i) - f_2(\mathbf{x}_i)| = |\alpha_k| \phi_k(\mathbf{x}_i), \quad i = 1, \cdots, n. \tag{5}$$

Then, the average error for all n sequentially learned inputs due to removing neuron k will be

$$E_{\text{ave}}(k) = \frac{\sum_{i=1}^{n} E(k,i)}{n} = \frac{|\alpha_k|}{n} \cdot \sum_{i=1}^{n} \phi_k(\mathbf{x}_i). \tag{6}$$

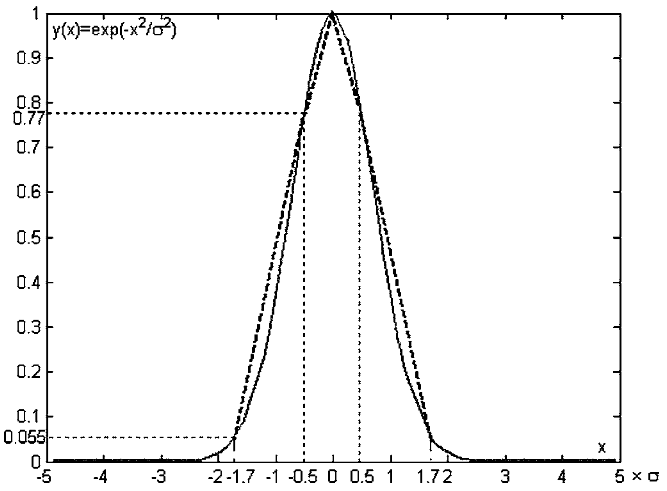

Fig. 1. Approximation of a Gaussian function by piecewise-linear functions.

Significance of a neuron is defined as its average output over all the input samples it has seen, as given by the above equation.[1] Calculation of $E_{\text{ave}}(k)$ in a sequential learning scheme would require storing all the past inputs and their corresponding outputs. This will be a time-consuming operation and is not ideal for sequential real-time applications. A simplified approach to calculate $E_{\text{ave}}(k)$ without storing all the past inputs and their corresponding outputs is given below.

When $\|\mathbf{x} - \mu_k\| > 1.7\sigma_k$, $\exp((-1/\sigma_k^2)\|\mathbf{x}_i - \mu_k\|^2) < 0.0556$. Then, the average error $E_{\text{ave}}(k)$ mainly results from losing outputs for the observations $\mathbf{x}_i$ located in the impact area $\{\mathbf{x} : \|\mathbf{x} - \mu_k\| \le 1.7\sigma_k\}$ of neuron k. The average error can then be approximated as

$$E_{\text{ave}}(k) \approx \frac{|\alpha_k|}{n} \cdot \sum_{i=1}^{m} \phi_k(\mathbf{x}_i), \quad \mathbf{x}_i \in \{\mathbf{x} : \|\mathbf{x} - \mu_k\| \le 1.7\sigma_k\}. \tag{7}$$

The above equation for significance of a neuron requires a computation involving exponential function which may be time consuming. In order to simplify this, a piecewise-linear approximation to the Gaussian function as shown in Fig. 1 is made use of here. For simplicity, the computation of the $E_{\text{ave}}(k)$ when x_i and μ_k are scalars (for the one-dimensional input case) is given as follows:

$$\phi_k(\mathbf{x}_i) \approx \begin{cases} \frac{0.77}{1.2\sigma_k}(x_i - \mu_k + 1.7\sigma_k), & \text{if } \mu_k - 1.7\sigma_k \le x_i < \mu_k - 0.5\sigma_k \\ 0.77 + \frac{0.23}{0.5\sigma_k}(x_i - \mu_k + 0.5\sigma_k), & \text{if } \mu_k - 0.5\sigma_k \le x_i < \mu_k \\ 0.77 - \frac{0.23}{0.5\sigma_k}(x_i - \mu_k - 0.5\sigma_k), & \text{if } \mu_k \le x_i < \mu_k + 0.5\sigma_k \\ -\frac{0.77}{1.2\sigma_k}(x_i - \mu_k - 1.7\sigma_k), & \text{if } \mu_k + 0.5\sigma_k \le x_i < \mu_k + 1.7\sigma_k \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Suppose that there are m input samples located in the impact area $(\mu_k - 1.7\sigma_k, \mu_k + 1.7\sigma_k)$ of neuron k and those observations are uniformly drawn from X, there are $((1)\sigma_k/(3.4)\sigma_k)m$

[1]From functional space [2] point of view, the "significance" can be considered as the average distance between the two RBF network functions f_1 and f_2 before and after removing a neuron.

observations located in the area $(\mu_k - 0.5\sigma_k, \mu_k + 0.5\sigma_k)$ and $((2.4)\sigma_k/(3.4)\sigma_k)m$ input samples located in the area $(\mu_k - 1.7\sigma_k, \mu_k - 0.5\sigma_k) \cup (\mu_k + 0.5\sigma_k, \mu_k + 1.7\sigma_k)$. Then, by approximations (7) and (8), we have

$$E_{\text{ave}}(k) \approx \frac{|\alpha_k|}{n} \cdot \left(\frac{1}{3.4}m \cdot \frac{1+0.77}{2} + \frac{2.4}{3.4}m \cdot \frac{0.77}{2}\right) \approx 0.53\frac{m}{n} \cdot |\alpha_k|. \tag{9}$$

Extending the above for the case when $\mathbf{x}$ and μ are vectors of l-dimension, we have

$$\begin{aligned} E_{\text{ave}}(k) &\approx \frac{|\alpha_k|}{n} \cdot \left(\frac{1}{3.4}m \cdot \frac{1+0.77}{2} + \frac{2.4}{3.4}m \cdot \frac{0.77}{2}\right) \\ &\approx 0.53^l \cdot \frac{m}{n} \cdot |\alpha_k|. \end{aligned} \tag{10}$$

Since all n observations are uniformly distributed in the range X, $(m/n) = (3.4\sigma_k)^l/S(X)$, where $S(X)$ is the size of the range X the training samples are drawn from, we have $E_{\text{ave}}(k) \approx |((1.8\sigma_k)^l\alpha_k/S(X))|$, which is the contribution of neuron k to the overall performance of the RBF network. Thus, the "significance" of neuron k can be quantified as

$$E_{\text{sig}}(k) = \left|\frac{(1.8\sigma_k)^l\alpha_k}{S(X)}\right| \tag{11}$$

where l is the dimension of the input space. The significance given by the above equation assumes a uniform distribution for the input data. For other general distributions like normal, Rayleigh, etc., the expressions for significance will be highly complex and are presented in Huang, *et al.* [10].

The learning process of RAN, RANEKF, and MRAN involves the allocation of new hidden neurons as well as adaptation of network parameters. The RBF network begins with no hidden neurons. As observations are received during training, some of them may initiate new hidden neurons based on a growing criterion. For the newly added neuron $K+1$ if $E_{\text{sig}}(K+1) = |((1.8\sigma_{K+1})^l\alpha_{K+1}/S(X))| < e_{\min}$, it means that the newly added neuron makes insignificant contribution to the overall performance of the whole network and hence this neuron should not be added at all. Therefore, an enhanced growing criterion to make the growing process smooth should be: if for new observation $(\mathbf{x}_n, y_n)$

$$\begin{cases} \|\mathbf{x}_n - \mu_{n\mathbf{r}}\| > \epsilon_n \\ |e_n| > e_{\min} \\ \frac{(1.8\cdot\kappa\|\mathbf{x}_n - \mu_{n\mathbf{r}}\|)^l|e_n|}{S(X)} > e_{\min} \end{cases} \tag{12}$$

a new neuron $K+1$ should be added and the parameters associated with the new hidden neuron are taken as follows:

$$\begin{cases} \alpha_{K+1} = e_n \\ \mu_{K+1} = \mathbf{x}_n \\ \sigma_{K+1} = \kappa\|\mathbf{x}_n - \mu_{n\mathbf{r}}\| \end{cases} \tag{13}$$

where $e_n = y_n - f(\mathbf{x}_n)$ and $\mu_{n\mathbf{r}}$ is the center which is nearest to $\mathbf{x}_n$.

If the significance of neuron k to the overall performance of the RBF network is less than the expected accuracy $e_{\min}$, neuron k is insignificant and should be removed; otherwise, neuron k is significant. Thus, we have a new pruning criterion, which is independent of the true function to be learned. For neuron k, given the desired approximation accuracy $e_{\min}$, if

$$E_{\text{sig}}(k) = \left|\frac{(1.8\sigma_k)^l\alpha_k}{S(X)}\right| < e_{\min} \tag{14}$$

then the average contribution made by neuron k in the whole range X is less than the expected accuracy $e_{\min}$ and the neuron k is insignificant; thus, neuron k can be removed. That means, after learning each observation, for all neurons, check whether $|((1.8\sigma_k)^l\alpha_k/S(X))| < e_{\min}$, $k = 1, \cdots, K$, and all the neurons with $|((1.8\sigma_k)^l\alpha_k/S(X))| < e_{\min}$ should be removed.

For the Gaussian function $\phi(x)$, its first and second derivatives will approach zero much faster as x moves away from zero. Thus, in the gradient vector $\mathbf{a}_n$ of EKF [4] all elements except the parameters of the nearest neuron will approach zero quickly. In order to increase the learning speed further, instead of adjusting the parameters and conducting pruning checking for all neurons after each observation, one may only need to adjust parameters for the nearest neuron and check the nearest neuron for pruning if no new neuron is added. It is neither necessary to adjust the parameters for all neurons nor necessary to check all neurons for possible pruning. If a new observation arrives and the growing criteria (12) is satisfied, a new significant neuron will be added. Since the parameters of all the rest neurons remain unchanged those neurons will remain significant after learning the new observation, and the new added neuron is also significant, thus, pruning checking need not be done after a new neuron is added. If a new observation arrives and the growing criteria (12) is not satisfied, no new neuron will be added and only the parameters of the nearest neuron will be adjusted. Thus (for the sake of increasing the learning speed), only the nearest neuron needs to be checked for growing and pruning.

Thus, we have a simple and efficient growing and pruning RBF algorithm as follows.

```
GAP-RBF Algorithm:
  Given an approximation error $e_{\min}$, for
each observation $(\mathbf{x}_n, y_n)$, where $\mathbf{x}_n \in \mathbf{R}^l$, do
1. compute the overall network output
        $f(\mathbf{x}_n) = \sum_{k=1}^{K}\alpha_k \exp\left(-\frac{1}{\sigma_k^2}\|\mathbf{x}_n - \mu_k\|^2\right)$    (15)
where $K$ is the number of hidden neurons.
2. calculate the parameters required in
the growth criterion
        $\epsilon_n = \max\{\epsilon_{\max}\gamma^n, \epsilon_{\min}\}, \quad (0 < \gamma < 1)$
        $e_n = y_n - f(\mathbf{x}_n)$    (16)
3. apply the criterion for adding neurons
  If $|e_n| > e_{\min}$ and $\|\mathbf{x}_n - \mu_{n\mathbf{r}}\| > \epsilon_n$ and
$((1.8\cdot\kappa\|\mathbf{x}_n - \mu_{n\mathbf{r}}\|)^l|e_n|/S(X)) > e_{\min}$
    allocate a new hidden neuron $K+1$ with
            $\alpha_{K+1} = e_n$
            $\mu_{K+1} = \mathbf{x}_n$
            $\sigma_{K+1} = \kappa\|\mathbf{x}_n - \mu_{n\mathbf{r}}\|$.    (17)
  Else
    adjust the network parameters $\alpha_{n\mathbf{r}}$, $\mu_{n\mathbf{r}}$, $\sigma_{n\mathbf{r}}$ for the nearest neuron only, using the EKF method.
    check the criterion for pruning the hidden neuron:
      If $|((1.8\sigma_{n\mathbf{r}})^l \alpha_{n\mathbf{r}}/S(X))| < e_{\min}$, where $S(X)$ is the estimated size of the range where the training samples are drawn from,
        remove the $n$rth hidden neuron
        reduce the dimensionality of EKF
      Endif
  Endif
```

III. Comparison of GAP-RBF With Other Algorithms

In this section, we present a critical comparison of GAP-RBF with the earlier RBF sequential learning schemes. The learning algorithms compared are RAN, RANEKF, MRAN, and also the algorithms developed by Salmerón *et al.* [7] and Rojas *et al.* [9].

A. Selection of Algorithm Parameters

In order to achieve a compact RBF network, MRAN introduces several algorithm parameters such as average expected error (for smooth neuron growth) $e'_{\min}$, pruning threshold δ, and growing and pruning sliding window size M. But there is no guideline as to how one chooses the right values for these parameters. *These parameters critically depend on the true functions (which we are trying to approximate) and the right values for these parameters can not be selected intuitively.* Also these parameters depend implicitly on the specified approximation error.

The algorithms proposed in Salmerón *et al.* [7] and Rojas *et al.* [9] also encounter similar difficulty of selecting appropriate parameter values. For example, the algorithm proposed in Salmerón *et al.* [7] needs tuning of more than 15 parameters as shown in Salmerón *et al.* [7, Tables I and II]. However, the only parameter required by the GAP-RBF may be $S(X)$, since the rest of the parameters could be fixed in most cases, as was done in all our simulations.

B. Smooth Growth of Neurons

In order to make the neuron growth smooth, MRAN augments the basic novelty criteria of RAN and RANEKF with an additional condition based on the RMS value of the output error over a sliding data window. This condition checks if the RMS value of the output error e_{rms} over a sliding window (M) is greater than a threshold $(e'_{\min})$. It is given by

$$e_{\mathrm{rms}} = \sqrt{\frac{\sum_{i=n-(M-1)}^{n} e_i^2}{M}} > e'_{\min} \tag{18}$$

where e_i is the network output error at ith instant and $e'_{\min}$ is the threshold selected for this criterion. This additional condition was introduced to ensure that the transition in the number of hidden neurons due to growing is smooth. It is not straightforward to determine these values for the parameters $e'_{\min}$ and M "a priori". But in *the GAP-RBF algorithm the neuron growth is inherently smooth as it is based on the significance of the neuron over all the input samples seen so far, although it does not store all the input samples.* Smooth growth of neurons can also be seen from the simulation results shown in Section IV.

C. Role of Significance of Neurons

In MRAN, Salmerón *et al.* [7] and Rojas *et al.* [9], there are no fixed relationships between the pruning threshold and the expected approximation accuracy. The pruning strategies used in MRAN, Salmerón *et al.* [7] and Rojas *et al.* [9] are not *accuracy-dependent.*

In GAP-RBF, the growing and pruning is based on the relationship between the significance of a neuron and the approximation accuracy. Based on the proposed significance determination criterion (14), the GAP-RBF changes neurons very smoothly, since it is guaranteed that the newly added neuron is significant and at most only one neuron (the nearest one) can be pruned at a time.

Mozer and Smolensky [11], [12] have used the concept of "relevance" to reduce network size automatically. It is to be noted that there are salient differences between the significance defined in this paper and Mozer and Smolensky's relevance of neurons. Conceptually speaking, the significance of a neuron is defined as the overall contribution of that neuron to the output of the network while the relevance of a neuron is defined as the difference between training errors before and after removing that neuron from the (fully trained) network. As discussed in Mozer and Smolensky [11], [12], the cost of computing ρ_k is $O(nK)$ stimulus presentations, where K is the number of neurons in the network and n is the number of observations in the training set. "If the training set is not fixed or is not known to the experimenter, additional difficulties arise in computing ρ_k." The approximation for ρ_k is $\rho_k \approx -(\partial E/\partial\alpha)$, which fluctuates "strongly" in time [11], [12]. In sequential learning, neither the training set nor the training errors E are known in advance, and it is not straightforward to calculate a neuron's relevance. The significance defined in this paper is not related to the training errors of the whole training set and only depends on the neurons' own parameters such as centers and impact widths and its computing is extremely simple.

D. Computational Complexity

Compared with RAN, for each observation, MRAN needs additional $O(M)$ operations to smoothen the growing of neurons, and $O(MK)$ operations for sorting and comparison in pruning phase, where M and K are the sliding window size and the number of hidden neurons respectively. As M and K increase, the computational complexity of MRAN will also increase. Since the EKF algorithm is used for the update, the complexity of the matrix operations in EKF increase at the rate of K^2. The computational complexity of Salmerón *et al.* [7] and Rojas *et al.* [9] are even worse than MRAN.

The complexity of GAP-RBF is much less since only the nearest neuron is checked for significance. GAP-RBF needs only one step (times and division) operation for pruning checking and a simple sparse matrix operation for parameter adjustment at each step using the EKF. It should be noted that in GAP-RBF when the parameters of the nearest neuron are adjusted all the cross correlation terms between the nearest neuron and all other neurons are also updated. All remaining elements of the error covariance matrix P are unchanged from their previous values. This is different to the philosophy used in the decoupled EKF (DEKF) algorithm (for MLP) proposed by Puskorius and Feldkamp [13], [14] to reduce the computational complexity. The key feature of DEKF is to ignore the interdependencies of mutually exclusive groups of neurons, i.e, the cross correlation terms of the error covariance matrix P. When a neuron's parameters are adjusted in the DEKF based learning algorithm, the cross correlation elements of that neuron to all the other neurons as well as all the other elements of the error covariance matrix P are assumed zero. Although only the nearest neuron is adjusted in our proposed learning algorithm, the interdependencies of different neurons are still maintained in order to achieve higher learning accuracy.

E. Memory Requirements

The implementation of the algorithm introduced in Rojas *et al.* [9] requires remembering all the input observations. MRAN has to remember the past M instantaneous errors and MK consecutive normalized outputs. The GAP-RBF algorithm does not have any special memory requirements since it only needs to remember the basic information of the network, i.e, the centers, widths, and weights of neurons.

Thus, with the introduction of the significance of neurons, which is *accuracy-dependent*, GAP-RBF can efficiently achieve a more compact network smoothly with higher accuracy and speed.

IV. Performance Evaluation of GAP-RBF Algorithm

In this section, the performance of the GAP-RBF learning algorithm is compared with other well-known sequential learning algorithms like RAN, RANEKF and MRAN on benchmark problems in the function approximation area. We also compare the performance of GAP-RBF with support vector regression (SVR). SVR is a kernel based regression algorithm with a rigorous mathematical basis. However, it should be noted that SVR is a batch learning algorithm and it needs all the data before training phase can commence, whereas GAP-RBF is a sequential learning algorithm in which all the training data may not be available at the same time. The comparison with SVR has been done for one artifical and one real world problems.

The benchmark problems are: 1) approximation of a rapidly varying continuous function; 2) approximation of the age of abalone using the abalone database; 3) Boston housing prediction—prediction of median house prices in the Boston area; and 4) auto-mpg—prediction of the fuel consumption of different models of cars. The first two problems are single-input single-output problems with uniform input distribution. The last three problems available in the UCI machine learning repository [15] are from the real world and are of higher dimensions and nonuniform input distribution. For each problem (except RANEKF for the abalone database), 50 trials have been done and the mean and standard deviation of achieved training/testing errors, number of neurons and CPU time have been compared. The training and testing observations are randomly generated for each trial of simulation and thus, for the same application different trial has different set of training and testing observations. All the simulations are carried out in MATLAB 6.5 environment running in a Pentium 4, 1.7-GHZ CPU. The simulations for SVR are carried out using compiled C-coded SVM packages: LIBSVM [16] running in the same PC. It should be noted that a C implementation would usually be much faster than a MATLAB implementation as done for GAP-RBF.

During all the experiments including those shown in this section, we have fixed the values as $\epsilon_{\max} = 1.15$, $\epsilon_{\min} = 0.04$, $\kappa = 0.10$, and $\gamma = 0.999$. The expected approximation accuracy $e_{\min}$ chosen for all cases is 0.0001.

A. Artificial Problem

In this case, GAP-RBF, MRAN, RANEKF, RAN, and SVR are used to approximate the following rapidly changing continuous function referred to as "SinE"

$$y(x) = 0.8\exp(-0.2x)\sin(10x). \tag{19}$$

For each trial, a training set (x_i, y_i) with 3000 data is created, where x_i's are uniformly randomly distributed in the interval (0,10). There are 1500 testing data (x_i, y_i) with x_i randomly distributed in the same range (0,10). For the MRAN algorithm, the pruning threshold is chosen as $e'_{\min} = 0.06$. After trial and error, an appropriate size of sliding window for growing and pruning is chosen as $M = 100$. Since training samples are drawn from the range $X = (0, 10)$, $S(X) = 10$. For SVR, the parameter C was tuned to $C = 1000$ along with other parameters.

Figs. 2 and 3 show that compared to RAN, RANEKF and MRAN, the GAP-RBF achieves better generalization performance and realizes the most compact network in the shortest training time. It can be further seen from Fig. 3 that GAP-RBF algorithm changes neurons much more smoothly compared to MRAN. These figures are for a typical trial and similar trends were observed for other trials and other applications as well. Since SVR is not really a "sequential" learning algorithm like GAP-RBF, SVR has to start only after all the training samples are received. Thus, we are not able to give the per-observation CPU time and error information, as shown in Fig. 2, nor the neuron/support vector evolution as in Fig. 3.

The final results for all the algorithms are shown in Table I. It can be seen from Table I that GAP-RBF performs better than all other algorithms.

B. Real-World Benchmark Problems

1) Abalone Age Prediction: The abalone problem [15] has 4177 cases predicting the age of abalone from physical measurements. Each observation consists of eight continuous input attributes and one integer output. There are three values for the first attribute: male, female, and infant, and this is an extremely nonuniform problem. For simplicity, the eight input attributes and the output can be normalized to the range $[0, 1]^9$. The parameters of SVR algorithm are tuned as $C = 300$ and $\gamma = 5$.

For the MRAN algorithm, the growing and pruning threshold is chosen as $e'_{\min} = 0.0001$ and an appropriate size of sliding window for growing and pruning is chosen as $M = 50$. Since the eight attributes of training samples are within the range $[0, 1]^8$, we set $S(X) = 1$ for GAP-RBF.

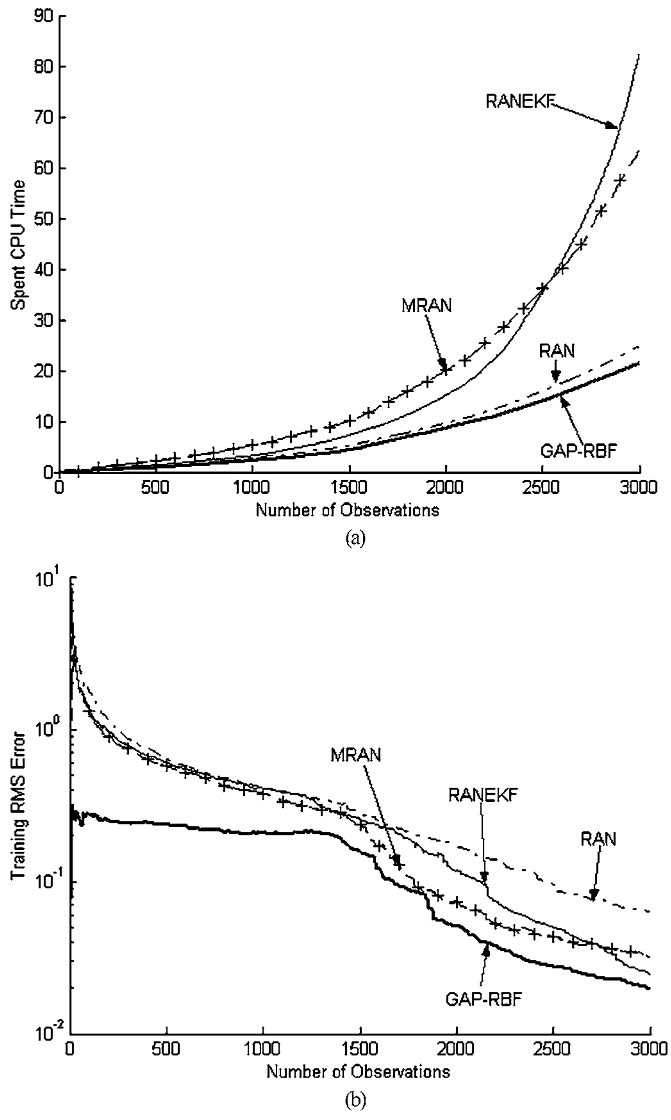

Fig. 2. Performance comparison for learning true function: Sin E. (a) Spent CPU time. (b) Rms error for training observations.

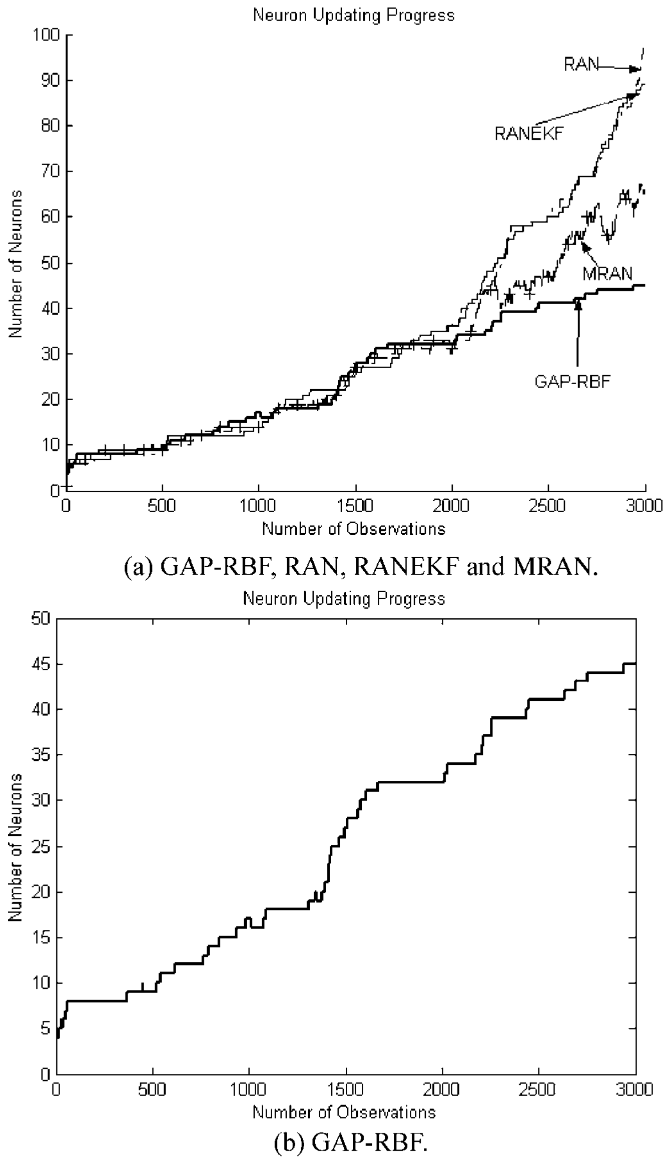

Fig. 3. Neuron updating progress for learning true function: SinE. (a) GAP-RBF, RAN, RANEKF, MRAN. (b) GAP-RBF.

For this problem, 3000 training data and 1177 testing data are randomly generated from the abalone database. RANEKF was not able to successfully learn abalone dataset if the memory of the computer was capped at 256 MB as the simulations always stop due to large number of neurons involving very large matrix computations. Thus, additional 1 GB memory was provided to the system when RANEKF was run for the abalone database. Since the learning time of RANEKF for the abalone database is much larger than the rest of algorithms, only 10 trials have been done for RANEKF.

Table II shows performance comparison of MRAN, RAN, GAP-RBF, RANEKF and SVR algorithms. It can be seen from the table that GAP-RBF produces comparable testing errors with that of the other algorithms, with the reduced network size and computation time.

2) *Boston Housing Prediction:* Boston housing database [15] has 506 cases concerning housing values in suburbs of Boston. Each observation consists of 13 input attributes (12 continuous attributes and one binary-valued attribute) and one continuous output (median value of owner-occupied homes). The 13 input attributes and the output can be simply normalized to the range $[0, 1]^{14}$. After simply analyzing the distributions of the 13 input attributes, it was found that the 13 input attributes are mainly nonuniformly distributed in the subspace $[0, 0.1] \times [0, 0.1] \times ([0, 0.4] \cup [0.6, 0.8]) \times [0, 0.1] \times ([0, 0.8] \cup [0.9, 1.0]) \times [0.3, 0.7] \times [0.1, 1.0] \times [0, 0.6] \times ([0, 0.3] \cup [0.9, 1.0]) \times ([0, 0.5] \cup [0.9, 1.0]) \times ([0.2, 0.3] \cup [0.4, 0.9]) \times [0.9, 1.0] \times [0, 0.8]$. Thus, for this case, $S(X)$ can be simply estimated for the GAP-RBF algorithm as: $S(X) = 0.1 \times 0.1 \times 0.6 \times 0.1 \times 0.9 \times 0.4 \times 0.9 \times 0.6 \times 0.4 \times 0.6 \times 0.6 \times 0.1 \times 0.8 = 1.5676 \times 10^{-6}$.

For the MRAN algorithm, the growing and pruning threshold is chosen as $e'_{\min} = 0.0001$ and an appropriate size of sliding window for growing and pruning is chosen as $M = 80$. For each trial, 481 training data and 25 testing data are randomly generated from the Boston housing database with random sequence. As shown in Table III, the GAP-RBF achieves a smaller network and takes less training time than other algorithms with comparable generalization performance to MRAN and RANEKF, but is much better than RAN.

TABLE I
PERFORMANCE COMPARISON FOR LEARNING TRUE FUNCTION: SinE

Algorithms	CPU Time(s)		Training Error (RMS)		Testing Error (RMS)		No. of Neurons	
	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev
GAP-RBF[a]	24.808	1.3989	0.0261	0.0085	0.0269	0.0087	45.32	3.3101
MRAN[a]	78.572	8.7595	0.0477	0.0205	0.0496	0.0212	43.64	13.718
RANEKF[a]	105.72	4.1335	0.0265	0.0117	0.0275	0.0124	89.36	2.8839
RAN[a]	45.514	0.5900	0.0659	0.0107	0.0673	0.0107	93.92	3.1741
SVR[b]	486.34	56.845	0.2591	0.0033	0.2629	0.0056	2088.4	26.871

[a] run in MATLAB environment.

[b] run in C executable environment which is faster than MATLAB by about 10-50 times.

TABLE II
PERFORMANCE COMPARISON FOR LEARNING REAL-WORLD DATA: ABALONE

Algorithms	CPU Time(s)		Training Error (RMS)		Testing Error (RMS)		No. of Neurons	
	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev
GAP-RBF[a]	83.784	73.401	0.0963	0.0061	0.0966	0.0068	23.62	9.5081
MRAN[a]	1500.4	134.08	0.0836	0.0039	0.0837	0.0042	87.571	7.1147
RANEKF[a]	90806	18193	0.0738	0.0042	0.0794	0.0053	409	22.485
RAN[a]	105.17	6.1714	0.0931	0.0091	0.0978	0.0092	345.58	12.578
SVR[b]	392.60	27.7235	0.0703	0.0006	0.0899	0.0050	564.92	12.0642

[a] run in MATLAB environment.

[b] run in C executable environment which is faster than MATLAB by about 10-50 times.

TABLE III
PERFORMANCE COMPARISON FOR LEARNING REAL-WORLD DATA: BOSTON HOUSING PREDICTION

Algorithms	CPU Time(s)		Training Error (RMS)		Testing Error (RMS)		No. of Neurons	
	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev
GAP-RBF	1.2399	0.2812	0.1507	0.0128	0.1418	0.0466	3.5	0.6468
MRAN	12.731	2.2585	0.1440	0.0108	0.1356	0.0411	13.58	1.8962
RANEKF	22.572	6.4159	0.1328	0.0086	0.1437	0.0464	19.98	1.8349
RAN	4.2664	0.4846	0.3449	0.0620	0.3432	0.0770	18.8	1.6413

3) Fuel Consumption Prediction of Automobiles: The auto-mpg problem [15] is a fuel consumption prediction of different models of cars based on the displacement, horsepower, weight and acceleration of the cars. There are 398 observations for predicting the fuel consumption (miles per gallon) of different models of cars. It consists of seven inputs (four continuous ones: displacement, horsepower, weight, acceleration; and three multivalues discrete ones: cylinders, model year and origin) and one continuous output (the fuel consumption).[2] For this problem, for each trial of simulations, 320 training data and 78 testing data are randomly generated from the Auto-Mpg database. Performance among GAP-RBF, RAN, RANEKF and MRAN are compared for this problem.

[2][Online] Available at: http://www.niaad.liacc.up.pt/~ltorgo/Regression/autompg.html

TABLE IV
PERFORMANCE COMPARISON FOR LEARNING REAL-WORLD DATA: AUTO-mpg

Algorithms	CPU Time(s)		Training Error (RMS)		Testing Error (RMS)		No. of Neurons	
	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev
GAP-RBF	0.4520	0.0786	0.1144	0.0132	0.1404	0.0270	3.12	0.7462
MRAN	1.4644	0.2453	0.1086	0.0100	0.1376	0.0226	4.46	0.7343
RANEKF	1.0103	0.1694	0.1088	0.0117	0.1387	0.0289	5.14	0.9037
RAN	0.8042	0.1417	0.2923	0.0808	0.3080	0.0915	4.44	0.8369

For simplicity, the eight input attributes and one output have been normalized to the range [0,1] in our experiment. After analyzing the distributions of the eight input attributes, they were found to be mainly nonuniformly distributed in some subspace and its $S(X)$ can be simply estimated as $S(X) = 0.0225$. For the MRAN algorithm, the growing and pruning threshold was chosen as $e'_{\min} = 0.0001$ and an appropriate size of sliding window for growing and pruning was chosen as $M = 50$ after several trial-and-error runs.

Table IV presents the generalization performance obtained by GAP-RBF, MRAN, and RANEKF which are comparable and is much better than that of RAN. GAP-RBF runs faster than all the rest algorithms for training and obtains the smallest network size.

V. CONCLUSIONS

In this paper, a simple idea to define and quantify the significance of a neuron is introduced and this is linked to the learning accuracy. Based on the definition of the significance of a neuron, a new growing and pruning RBF network called GAP-RBF has been developed. The growing and pruning criteria allows adding and pruning of neurons only when it is significant to the overall performance of the network. This results in a smooth growth of neurons and a compact network. Further, only the nearest neuron needs to be checked for growing and pruning. This makes the algorithm computationally efficient.

Performance of the GAP-RBF learning algorithm has been compared with other sequential learning algorithms like RAN, RANEKF and MRAN on four benchmark problems in the function approximation area. Also, a comparison with SVR is done for an artificial and one real world problem. The results indicate better performance of GAP-RBF algorithm in terms of generalization, network size and training speed when the input data is uniformly distributed (Problem 1). When the input data is not uniformly distributed (abalone, Boston housing and auto-mpg), GAP-RBF gives comparable generalization performance with a much smaller network size and much less training time.

REFERENCES

[1] J. Platt, "A resource-allocating network for function interpolation," *Neural Comput.*, vol. 3, pp. 213–225, 1991.
[2] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Comput.*, vol. 5, pp. 954–975, 1993.
[3] N. B. Karayiannis and G. W. Mi, "Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques," *IEEE Trans. Neural Networks*, vol. 8, no. 6, pp. 1492–1506, 1997.
[4] L. Yingwei, N. Sundararajan, and P. Saratchandran, "A sequental learning scheme for function approximation using minimal radial basis function (RBF) neural networks," *Neural Comput.*, vol. 9, pp. 461–478, 1997.
[5] L. Yingwei, N. Sundararajan, and P. Saratchandran, "Performance evaluation of a sequental minimal radial basis function (RBF) neural network learning algorithm," *IEEE Trans. Neural Networks*, vol. 9, pp. 308–318, Mar. 1998.
[6] N. Sundararajan, P. Saratchandran, and L. Yingwei, *Radial Basis Function Neural Networks With Sequential Learning: MRAN and Its Applications*. Singapore: World Scientific, 1999.
[7] M. Salmerón, J. Ortega, C. G. Puntonet, and A. Prieto, "Improved RAN sequential prediction using orthogonal techniques," *Neurocomputing*, vol. 41, pp. 153–172, 2001.
[8] M. Salmerón, J. Ortega, C. G. Puntonet, A. Prieto, and I. Rojas, "SSA, SVD, QR-cp, and RBF model reduction," in *Lecture Notes in Comput. Sci.*, vol. 2415, 2002, pp. 589–594.
[9] I. Rojas, H. Pomares, J. L. Bernier, J. Ortega, B. Pino, F. J. Pelayo, and A. Prieto, "Time series analysis using normalized PG-RBF network with regression weights," *Neurocomputing*, vol. 42, pp. 267–285, 2002.
[10] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Trans. Neural Networks*, to be published.
[11] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Advances in Neural Inform. Processing Syst. (NIPS 1988)*, 1988, pp. 107–115.
[12] M. C. Mozer and P. Smolensky, "Using relevance to reduce network size automatically," *Connection Sci.*, vol. 1, no. 1, pp. 3–16, 1989.
[13] G. V. Puskorius and L. A. Feldkamp, "Decoupled extended kalman filter training of feedforward layered networks," in *Proc. Int. Joint Conf. Neural Networks*, Seattle, WA, July 1991, pp. 771–777.
[14] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of nonlinear dynamics systems with Kalman filter trained recurrent networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 279–297, Mar. 1994.
[15] C. Blake and C. Merz, UCI Repository of Machine Learning Databases, Dept. Inform. Comput. Sci., Univ. California, Irvine, 1998.
[16] C.-C. Chang and C.-J. Lin. (2003) LIBSVM—A Library for Support Vector Machines. Dept. Comput. Sci. Inform. Eng., National Taiwan Univ., Taiwan. [Online]http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Guang-Bin Huang (M'98–SM'04) received the B.Sc degree in applied mathematics and the M.Eng degree in computer engineering from Northeastern University, China, in 1991 and 1994, respectively, and the Ph.D degree in electrical engineering from Nanyang Technological University, Singapore, in 1999.

From June 1998 to May 2001, he was a Research Fellow with Singapore Institute of Manufacturing Technology (formerly known as Gintic Institute of Manufacturing Technology), where he has led/implemented several key industrial projects. Since May 2001, he has been an Assistant Professor in the Information Communication Institute of Singapore (ICIS), School of Electrical and Electronic Engineering, Nanyang Technological University. His current research interests include soft computing and networking.

P. Saratchandran (M'87–SM'96) received the B.Sc. (Eng.) degree from the Regional Engineering College, Calicut, India, and the M.Tech. degree from the Indian Institute of Technology, Kharagpur, India, both in electrical engineering. He received the M.Sc. degree in systems enginering from City University, London, U.K., and the Ph.D degree in the area of control engineering from Oxford University, Oxford, U.K.

For two years he was a Scientist with the Indian Space Research Organization and spent five years as a Senior Design Engineer with the Hindustan Aeronautics Ltd., India, designing defense-related avionics systems. From 1984 to 1990, he worked in Australia for various defense industries as a Systems Consultant and Manager, developing real-time software/systems in Ada for the Australian Defense forces. During this period, he was also a Visiting Fellow at the Department of Mathematics and Computer Science, Macquarie University, Sydney, Australia. Since 1990, he has been with Nanyang Technological University, Singapore, where he is now an Associate Professor. He has several publications in refereed journals and has authored four books: Fully Tuned Radial Basis Function Neural Networks for Flight Control (Boston, MA: Kluwer, 2001), Radial Basis Function Neural Networks with Sequential Learning (Singapore: World Scientific, 1999), Parallel Architectures for Artificial Neural Networks (Los Alamitos, CA: IEEE Computer Society Press, 1998), and Parallel Implementations of Backpropagation Neural Networks (Singapore: World Scientific, 1996). His research interests are in neural networks, parallel computing, and control.

Dr. Saratchandran is an Editor for the journal *Neural Parallel and Scientific Computations*.

Narasimhan Sundararajan (M'74–SM'84–F'96) received the B.E. degree in electrical engineering (with first-class honors) from the University of Madras, Madras, India, in 1966, the M.Tech degree from the Indian Institute of Technology, Madras, in 1968, and the Ph.D. degree in electrical engineering from the University of Illinois, Urbana-Champaign, in 1971.

During 1972 to 1991, he worked for the Indian Space Research Organization and NASA on aerospace problems. Since February 1991, he has been with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, where he is currently a Professor. He was an IG Sarma Memorial ARDB Chaired Professor during December 2003–Feburary 2004 at the Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India. His research interests are in the areas of neural networks and control, with aerospace applications. He has published more than 100 papers and four books (in the area of neural networks titled): Radial Basis Function Neural Networks with Sequential Learning (Singapore: World Scientific, 1999), Parallel Architectures for Artificial Neural Networks (Los Alamitos, CA: IEEE Computer Society Press,1998), Parallel Implementations of Backpropagation Neural Networks (Singapore: World Scientific, 1996), and Fully Tuned Radial Basis Function Neural Networks for Flight Control (Boston, MA: Kluwer, 2001).

Dr. Sundararajan is an Associate Fellow of AIAA. Presently, he is an Associate Editor for the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY and the *IFAC Journal on Control Engineering Practice*.