A Generalized Growing and Pruning RBF (GGAP-RBF) Neural Network for Function Approximation

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

***Abstract*—This paper presents a new sequential learning algorithm for radial basis function (RBF) networks referred to as generalized growing and pruning algorithm for RBF (GGAP-RBF). The paper first introduces the concept of *significance* for the hidden neurons and then uses it in the learning algorithm to realize parsimonious networks. The growing and pruning strategy of GGAP-RBF is based on linking the required learning accuracy with the *significance* of the *nearest* or intentionally added new neuron. Significance of a neuron is a measure of the average information content of that neuron. The GGAP-RBF algorithm can be used for any arbitrary sampling density for training samples and is derived from a rigorous statistical point of view. Simulation results for bench mark problems in the function approximation area show that the GGAP-RBF outperforms several other sequential learning algorithms in terms of learning speed, network size and generalization performance regardless of the sampling density function of the training data.**

***Index Terms*—Growing, neuron's significance, pruning, radial basis networks, sequential learning.**

I. Introduction

Radial basis function (RBF) networks have gained much popularity in recent times due to their ability to approximate complex nonlinear mappings directly from the input–output data with a simple topological structure. Several learning algorithms have been proposed in the literature for training RBF networks [1]–[12]. Selection of a learning algorithm for a particular application is critically dependent on its accuracy and speed. In practical online applications, sequential learning algorithms are generally preferred over batch learning algorithms as they do not require retraining whenever a new data is received. Compared with the batch learning algorithms, the sequential learning algorithms that we will discuss in this paper have the following distinguishing features:

1) all the training observations are *sequentially* (one-by-one) presented to the learning system;
2) at any time, *only one* training observation is seen and learned;
3) a training observation is *discarded* as soon as the learning procedure for that particular observation is completed;
4) the learning system has no *prior* knowledge as to how many total training observations will be presented.

Manuscript received January 21, 2003; revised September 12, 2003.

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/TNN.2004.836241

Thus, if one strictly applies the above features of the sequential algorithms, many of the existing algorithms are not sequential. One major bottleneck seems to be that they need the entire training data ready for training before the training procedure starts and, thus, they are not really sequential. This point is highlighted in a brief review of the existing algorithms given below.

The algorithms proposed by Chen *et al.* [4], [5] and Chng *et al.* [6] can add neurons to the network one by one and obtain more compact networks than conventional RBF networks. However, they need to select a subset network from a N-term full network based on some orthogonal subset selection scheme (orthogonal least square (OLS) [4] or regularized OLS [5]), where the N-term full network is formed after all the N training observations are presented.

Based on the subset selection technique [4], [5], Orr [7] proposed the regularized forward selection (RFS) algorithm for RBF networks, that combines forward subset selection and zeroth-order regularization and achieves better generalization. Unlike other approaches involving several preset parameters and thresholds (used for adding new centers and performing gradient descent) that must be tuned with each new problem, RFS has only one preset parameter, the basis function width. The implementation of Orr [7] still depends on the data being available all at the same time and, hence, is strictly not a sequential one. The computation cost (number of floating point operations) for parameter adjustment at each learning cycle can be up to $O(n^3)$, where n is the number of training data which is usually very large.

The pattern classification algorithm proposed by Bors and Gabbouj [8] initializes the centers and widths of RBF neurons and updates the weights based on backpropagation learning where a cost function based on all the training data is required [8, eqs. (13) and (19)]. Obviously, all these algorithms including support vector machine (SVM) and its variants [13]–[15] depend on the data being available all at a time. Thus, strictly speaking, all the above learning algorithms are not sequential learning algorithms but variations of batch algorithms only.

A major contribution to sequential learning was made by Platt [1] by proposing resource allocation network (RAN) in which hidden neurons were added sequentially based on the novelty of the new data. Enhancement of RAN, known as RANEKF was proposed by Kadirkamanathan and Niranjan [2] in which extended Kalman filter (EKF) rather than least mean square (LMS) algorithm was used for updating the network parameters (centers, widths, and weights of Gaussian neurons) to improve the accuracy and obtain a more compact network. RAN and RANEKF can only add neurons to the network and can not prune insignificant neurons from the network. The networks obtained by RAN and RANEKF could become extremely large in some large-scale complex applications.

A significant improvement to RAN and RANEKF was made by Yingwei *et al.* [9], [10] 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 [16]. MRAN uses a sliding data window in the growing and pruning criteria to identify the neurons that contribute relatively little to the network output. Selection of the appropriate sizes for these windows critically depend on the distribution of the input samples. In MRAN, choosing proper window sizes can only be done by trial and error based on exhaustive simulation studies.

In [11], QR factorization and singular value decomposition methods are used for determining the structure as well as for pruning the network. It has been shown [11] that the size of the network is more compact than RAN. However, these methods require additional computational efforts. Quite a large number of (around 15) parameters/variables need to be preset (by trial and error) and training data need to be stored and reused for pruning purposes.

To realize a compact RBF network similar to [11] and [12] presents a pruning scheme which checks the pruning criteria for *all* hidden neurons *only after all the* training observations have been presented and learned. Pruning is not conducted during the learning stage and, thus, these two algorithms do not generate compact network during the learning stage. In most practical sequential applications, the learning system may not know when all the observations have been presented and whether there are some more observations to be input to the system. Thus, it may not be easy for users to determine when pruning should be conducted in practical online applications. Similar to MRAN, selection of the appropriate values for those parameters introduced in these two algorithms critically depend on the distribution of the input data and can only be done by exhaustive trial and error based on simulation studies as well. The algorithm proposed by Rojas *et al.* [12] is also computationally expensive.

One important point to be noted in all these sequential algorithms is that they do not link the required learning accuracy directly to the 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. For a function approximation problem, the required approximation or learning accuracy can be defined as the q-norm Euclidian distance (L_q-distance) between the true function output vector $(\mathbf{y}_1, \ldots, \mathbf{y}_n)^T$ and the approximating network's output vector $(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n))^T$ for n training observations $(\mathbf{x}_i, \mathbf{y}_i)$, i.e., $((\sum_{i=1}^{n} \|f(\mathbf{x}_i) - \mathbf{y}_i\|_q^q)/n)^{1/q}$. (Refer to [17] for the definition of q-norm for vectors.)

This paper introduces the concept of "significance" for the hidden neurons and directly links the required learning accuracy to the significance of neurons in the learning algorithm so as to realize a compact RBF network. *Significance of a neuron gives a measure of the information content in the neuron about the function to be learned and is defined as the contribution made by that neuron to the network output averaged (in the q-norm sense) over all the input data received so far.* A neuron will not be added to the network if its significance is going to be little. To our best knowledge, this is different from all the existing sequential learning algorithms including RAN, RANEKF, HSOL [18], and the sequential growing and pruning algorithm proposed by Todorović and Stanković [19] since all of them add new neurons based on their novelty to the individual instant observations. However in our proposed algorithm, a neuron can be only added when it is statistically significant to all the observations including the observations which have been learned but discarded already. Likewise, in our proposed algorithm, a hidden neuron with little significance—the contribution made by that neuron to the network output averaged over *all* the observations received—is simply removed from the network. In contrast, both HSOL [18] and the sequential growing and pruning algorithm proposed in [19] prune neurons mainly based on their significance computed based on that individual instant input observations.

Our concept of significance is also wholly different from and much simpler than that of [19] where it was defined based on the sensitivity of a neuron's width and connection weight to the output error for the current input data. To put it simply, the significance proposed in this paper is defined as a neuron's statistical contribution to the overall performance of the network.

In the generalized growing and pruning RBF (GGAP-RBF) algorithm proposed in this paper, this significance is used in growing and pruning strategies. A new neuron will be added only if its significance is more than the chosen learning accuracy. If during training the significance for a neuron becomes less than the learning accuracy, then that neuron will be pruned. The GGAP-RBF algorithm is truly sequential and can be used for on-line learning in real time applications where the training observations are sequentially (one-by-one) presented and discarded after being learned, and the learning (parameter adjustment, network growing or pruning) is conducted whenever a new observation is presented.

The main difference between the work of Salmerón *et al.* [11] and Rojas *et al.* [12] and that in this paper is: [11] [12] do not conduct pruning during the learning stage. As shown in Section IV, the algorithms without pruning functions during learning stage may result in a large network leading to learning failure. In our proposed algorithm, the pruning is checked and conducted through the whole learning phase and the network architecture remains compact and, thus, avoids the above problem.

In this paper, the performance comparison of the GGAP-RBF with RAN, RANEKF, and MRAN in terms of learning accuracy, learning speed and compactness of the network are presented for two benchmark problems, namely California Housing and chaotic time series prediction problems. The results indicate the superior performance of GGAP-RBF for all the problems studied. Recently a fast implementation of SVM for regression (SVR) based on sequential minimal optimization (SMO) [15] has become popular. We have compared our GGAP-RBF with this SMO even though GGAP-RBF is a sequential algorithm whereas SMO is not a truly sequential algorithm. SMO sequentially adjusts only the Lagrange multipliers and the data are processed batch by batch. The learning phase starts when all data are ready and no new data are added during the learning phase. The simulation results on real large complex applications show that the proposed GGAP-RBF algorithm is faster and provides much smaller networks than SMO, and provides a comparable generalization performance.

The rest of this paper is organized as follows. Section II introduces the definition of significance of neurons and then gives a simple estimation scheme for the significance. Section III describes the proposed "significance"-based generalized growing and pruning RBF (GGAP-RBF) learning algorithm derived for arbitrary input sampling density functions. Section IV presents the performance comparison results for GGAP-RBF along with RAN, RANEKF, and MRAN for two bench-mark problems, i.e., California Housing which is a real large-scale complex function approximation problem and Mackey–Glass chaotic time series prediction problem. Section V summarizes the conclusions from this study.

II. Definition and Estimation of Significance of Neurons

This section first introduces the notion of *significance* for the hidden neurons based on their statistical average contribution over all inputs seen so far, although those inputs are discarded and not stored in the system after being learned.

The output of a RBF network with K neurons for an input vector $\mathbf{x} = (x_1, \ldots, x_l)^T \in X \subseteq \mathbf{R}^l$, where l is the dimension of input observation space $X \subseteq \mathbf{R}^l$ and T means the transpose of vectors, is given by

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

where α_k is the weight connecting the kth hidden neuron to the output neuron and $\phi_k(\mathbf{x})$ is the response of the kth hidden neuron for an input vector $\mathbf{x}$

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

where $\mu_k = (\mu_{k,1}, \ldots, \mu_{k,l})^T \in \mathbf{R}^l$ and σ_k are the center and width of the kth hidden neuron, respectively, $k = 1, \ldots, K$.

In sequential learning, a series of training samples are randomly drawn and presented to, and learned by the network one by one. Let a series of training samples $(\mathbf{x}_i, \mathbf{y}_i), i = 1, 2, \ldots,$ be drawn sequentially and randomly from a range X with a sampling density function of $p(\mathbf{x})$, where X is a subset of an l-dimensional Euclidian space. The sampling density function $p(\mathbf{x})$ is defined as

$$\int \cdots \int_X p(\mathbf{x})\, d\mathbf{x} = 1. \tag{3}$$

For simplicity, in this paper, $\int \cdots \int_X$ is denoted by $\int_X$. The size of the range X can be denoted by $S(X) = \int_X 1\, d\mathbf{x}$. 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

$$f_1(\alpha_1, \mu_1, \sigma_1, \ldots, \alpha_K, \mu_K, \sigma_K, \mathbf{x}_i) = \sum_{j=1}^{K} \alpha_j \phi_j(\mathbf{x}_i). \tag{4}$$

If the neuron k is removed, the output of the RBF network with the remaining $K-1$ neurons for the input $\mathbf{x}_i$ is

$$\begin{aligned} &f_2(\alpha_1, \mu_1, \sigma_1, \ldots, \alpha_{k-1}, \mu_{k-1}, \sigma_{k-1}, \alpha_{k+1} \\ &\quad \mu_{k+1}, \sigma_{k+1}, \ldots, \alpha_K, \mu_K, \sigma_K, \mathbf{x}_i) \\ &\quad = \sum_{j=1}^{k-1} \alpha_j \phi_j(\mathbf{x}_i) + \sum_{j=k+1}^{K} \alpha_j \phi_j(\mathbf{x}_i). \end{aligned} \tag{5}$$

Thus, for an observation $\mathbf{x}_i$, the error resulted from removing neuron k is given by

$$\begin{aligned} E(k,i) &= \|f_1(\alpha_1, \mu_1, \sigma_1, \ldots, \alpha_K, \mu_K, \sigma_K, \mathbf{x}_i) \\ &\quad - f_2(\alpha_1, \mu_1, \sigma_1, \ldots, \alpha_{k-1}, \mu_{k-1}, \sigma_{k-1} \\ &\quad \alpha_{k+1}, \mu_{k+1}, \sigma_{k+1}, \ldots, \alpha_K, \mu_K, \sigma_K, \mathbf{x}_i)\|_q \\ &= \|\alpha_k\|_q \phi_k(\mathbf{x}_i), \quad i = 1, \ldots, n \end{aligned} \tag{6}$$

where $\|\cdot\|_q$ is the q-norm of vectors, indicating the L_q-distance between two points in Euclidian space.

In theory, the q-norm of the error E_{q} for all n sequentially learned observations caused by removing the neuron k is

$$E_q(k) = \|(E(k,1), \ldots, E(k,n))^T\|_q. \tag{7}$$

This can be further written as

$$E_q(k) = \left(\frac{\sum_{i=1}^{n} E^q(k,i)}{n}\right)^{1/q} = \|\alpha_k\|_q \left(\frac{\sum_{i=1}^{n} \phi_k^q(\mathbf{x}_i)}{n}\right)^{1/q}. \tag{8}$$

However, the computation complexity of $E_q(k)$ would be very high if it were calculated based on all learned observations and if n is large. On the other hand, in the sequential learning implementation after learning the training observations $(\mathbf{x}_i, \mathbf{y}_i), i = 1, \ldots, n,$ are no longer stored in the system and the value n may be unknown and not recorded either. In fact, there may possibly have some more observations to be input further. Thus, there must be some simpler and better way to calculate $E_q(k)$ as stated in (8) without the prior knowledge of each specified observations $(\mathbf{x}_i, \mathbf{y}_i)$.

Suppose that the observations $(\mathbf{x}_i, \mathbf{y}_i), i = 1, 2, \ldots,$ are drawn from a sampling range X with a sampling density function $p(\mathbf{x})$. Suppose that at an instant of time, n observations $(\mathbf{x}_i, \mathbf{y}_i)$ have been learned by the sequential learning system. Let the sampling range X be divided into N small spaces $\Delta_j, j = 1, \ldots, N$. The size of Δ_j is represented by $S(\Delta_j)$. Since the sampling density function is $p(\mathbf{x})$ there are about $n \cdot p(\mathbf{x}_j) \cdot S(\Delta_j)$ samples in each Δ_j, where $\mathbf{x}_j$ is any point chosen in Δ_j. From (8), we have

$$\begin{aligned} E_q(k) &\approx \|\alpha_k\|_q \left(\frac{\sum_{j=1}^{N} \phi_k^q(\mathbf{x}_j) \cdot np(\mathbf{x}_j) \cdot S(\Delta_j)}{n}\right)^{1/q} \\ &= \|\alpha_k\|_q \left(\sum_{j=1}^{N} \phi_k^q(\mathbf{x}_j) p(\mathbf{x}_j) S(\Delta_j)\right)^{1/q}. \end{aligned} \tag{9}$$

When the number of input observations n is large and Δ_j is small, we have

$$\begin{aligned}\lim_{n\to+\infty} E_q(k) &\approx \lim_{N\to+\infty} \|\alpha_k\|_q \left(\sum_{j=1}^{N} \phi_k^q(\mathbf{x}_j)p(\mathbf{x}_j)S(\Delta_j)\right)^{1/q}\\ &= \|\alpha_k\|_q \left(\int_X \phi_k^q(\mathbf{x})p(\mathbf{x})\,d\mathbf{x}\right)^{1/q}\\ &= \|\alpha_k\|_q \left(\int_X \exp\left(-\frac{q\|\mathbf{x}-\mu_k\|^2}{\sigma_k^2}\right)p(\mathbf{x})\,d\mathbf{x}\right)^{1/q}.\end{aligned} \tag{10}$$

This is the statistical contribution of neuron k to the overall output of the RBF network and we define this as the "significance" of a specified neuron k, $E_{\text{sig}}(k)$, and is given by

$$\begin{aligned}E_{\text{sig}}(k) &= \lim_{n\to+\infty} E_q(k)\\ &= \|\alpha_k\|_q \left(\int_X \exp\left(-\frac{q\|\mathbf{x}-\mu_k\|^2}{\sigma_k^2}\right)p(\mathbf{x})\,d\mathbf{x}\right)^{1/q}.\end{aligned} \tag{11}$$

If the significance of neuron k is less than the required learning accuracy $e_{\min}$, then neuron k should be deemed insignificant and removed, otherwise, neuron k is significant and should be retained.

More interestingly, if the distributions of the l attributes $(x_1,\ldots,x_i,\ldots,x_l)^T$ of observations $\mathbf{x}$'s are independent from each other, the density function $p(\mathbf{x})$ of $\mathbf{x}$ can be written as: $p(\mathbf{x}) = \prod_{i=1}^{l} p_i(x_i)$, where $p_i(x)$ is the density function of the ith attribute x_i of observations. Thus, in this case, "significance" (11) can be rewritten as

$$\begin{aligned}&E_{\text{sig}}(k)\\ &= \|\alpha_k\|_q \left(\int\cdots\int_X \exp\left(-\frac{q\|\mathbf{x}-\mu_k\|^2}{\sigma_k^2}\right)p(\mathbf{x})\,d\mathbf{x}\right)^{1/q}\\ &= \|\alpha_k\|_q \prod_{i=1}^{l}\left(\int_{a_i}^{b_i} \exp\left(-\frac{q(x-\mu_{k,i})^2}{\sigma_k^2}\right)p_i(x)\,dx\right)^{1/q}\end{aligned} \tag{12}$$

where l is the dimension of the input space X and (a_i, b_i) the interval of the ith attribute x_i of observations, $i = 1,\ldots,l$.

The above equation involves an integration of the probability density function $p(\mathbf{x})$ in the sampling range X. This can be done analytically for some simple but popularly used $p(\mathbf{x})$ functions like uniform, normal, exponential and Rayleigh functions, etc.

A. Uniform Sampling Distribution

When the input samples are uniformally drawn from a range X, the sampling density function $p(\mathbf{x})$ is given by $p(\mathbf{x}) = (1/S(X))$, where $S(X)$ is the size of the range X given by $S(X) = \int_{\mathbf{x}} 1\,d\mathbf{x}$. Substituting for $p(\mathbf{x})$ in (11) we get

$$E_{\text{sig}}(k) = \|\alpha_k\|_q \left(\int_X \exp\left(-\frac{q\|\mathbf{x}-\mu_k\|^2}{\sigma_k^2}\right)\frac{1}{S(X)}\,d\mathbf{x}\right)^{1/q}. \tag{13}$$

Note that in general the width σ_k of a neuron k is much less than the size of range X, the above equation can be approximated as

$$\begin{aligned}E_{\text{sig}}(k) &= \|\alpha_k\|_q \left(\int_X \exp\left(-\frac{q\|\mathbf{x}-\mu_k\|^2}{\sigma_k^2}\right)\frac{1}{S(X)}\,d\mathbf{x}\right)^{1/q}\\ &\approx \frac{\|\alpha_k\|_q}{S(X)^{1/q}}\left(2\int_0^{+\infty}\exp\left(-\frac{qx^2}{\sigma_k^2}\right)dx\right)^{l/q}\\ &= \|\alpha_k\|_q \left(\frac{\pi}{q}\right)^{l/2q}\left(\frac{\sigma_k^l}{S(X)}\right)^{1/q}.\end{aligned} \tag{14}$$

For the uniform sampling density case, one needs to know the size $S(X)$ of the sampling range X. In most applications, $S(X)$ may be known or can be simply estimated. In fact, as we do in some of our simulations, one may just normalize the inputs to the range $[0,1]^l$ and get $S(X) = 1$.

B. Normal Sampling Distribution

When the input samples are drawn from a normal sampling distribution, $p(\mathbf{x})$ can be expressed as

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{15}$$

Thus, the "significance" (11) can be rewritten as

$$\begin{aligned}E_{\text{sig}}(k) &= \frac{\|\alpha_k\|_q}{(\sqrt{2\pi}\sigma)^{1/q}}\\ &\quad\times\left(\int_X \exp\left(-\frac{q(x-\mu_k)^2}{\sigma_k^2}-\frac{(x-\mu)^2}{2\sigma^2}\right)dx\right)^{1/q}\\ &= \frac{\|\alpha_k\|_q}{(\sqrt{2\pi}\sigma)^{1/q}}\exp\left(-\frac{(\mu-\mu_k)^2}{2q\sigma^2+\sigma_k^2}\right)\\ &\quad\times\left(\int_X \exp\left(-\frac{2q\sigma^2+\sigma_k^2}{2\sigma^2\sigma_k^2}\right.\right.\\ &\quad\times\left.\left.\left(x-\frac{\mu_k+\frac{\sigma_k^2}{2q\sigma^2}\mu}{1+\frac{\sigma_k^2}{2q\sigma^2}}\right)^2\right)dx\right)^{1/q}.\end{aligned} \tag{16}$$

In general, $\sigma_k \ll \sigma$ since the samples are drawn from the whole range of X and the neuron k impacts only part area of X. Thus, the significance can be expressed as

$$\begin{aligned}E_{\text{sig}}(k) &\approx \frac{\|\alpha_k\|_q}{(\sqrt{2\pi}\sigma)^{1/q}}\exp\left(-\frac{(\mu-\mu_k)^2}{2q\sigma^2+\sigma_k^2}\right)\\ &\quad\times\left(\int_{-\infty}^{+\infty}\exp\left(-\frac{2q\sigma^2+\sigma_k^2}{2\sigma^2\sigma_k^2}x^2\right)dx\right)^{1/q}\\ &= \|\alpha_k\|_q\left(\frac{\sigma_k}{\sqrt{2q\sigma^2+\sigma_k^2}}\right)^{1/q}\exp\left(-\frac{(\mu-\mu_k)^2}{2q\sigma^2+\sigma_k^2}\right)\\ &\approx \|\alpha_k\|_q\left(\frac{\sigma_k}{\sqrt{2q}\sigma}\right)^{1/q}\exp\left(-\frac{(\mu-\mu_k)^2}{2q\sigma^2}\right).\end{aligned} \tag{17}$$

C. Rayleigh Sampling Distribution

For the case of Rayleigh sampling distribution, the sampling density function is

$$p(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ \frac{x}{\mu^2}\exp\left(-\frac{x^2}{2\mu^2}\right), & \text{otherwise} \end{cases}. \tag{18}$$

In this case, the "significance" (11) can be estimated as

$$\begin{aligned} E_{\text{sig}}(k) &= \frac{\|\alpha_k\|_q}{\mu^{2/q}}\left(\int_X x\exp\left(-\frac{q(x-\mu_k)^2}{\sigma_k^2}-\frac{x^2}{2\mu^2}\right)dx\right)^{1/q} \\ &= \frac{\|\alpha_k\|_q}{\mu^{2/q}}\exp\left(\frac{\mu_k^2}{2q\mu^2+\sigma_k^2}\right) \\ &\quad\times\left(\int_X x\exp\left(-\frac{2q\mu^2+\sigma_k^2}{2\mu^2\sigma_k^2}\left(x-\frac{2q\mu^2\mu_k}{2q\mu^2+\sigma_k^2}\right)^2\right)dx\right)^{1/q} \\ &\approx \frac{\|\alpha_k\|_q}{\mu^{2/q}}\exp\left(\frac{\mu_k^2}{2q\mu^2+\sigma_k^2}\right) \\ &\quad\times\left(\frac{4\pi\mu^2\sigma_k^2}{2q\mu^2+\sigma_k^2}\right)^{1/2q}\left(\frac{2\mu^2\sigma_k^2}{2q\mu^2+\sigma_k^2}\right)^{1/q} \\ &= \|\alpha_k\|_q\cdot\left(\frac{2}{\mu}\right)^{2/q}\cdot\pi^{1/2q} \\ &\quad\cdot\frac{(\mu\sigma_k)^{3/q}}{(2q\mu^2+\sigma_k^2)^{3/2q}}\exp\left(\frac{\mu_k^2}{2q\mu^2+\sigma_k^2}\right). \end{aligned} \tag{19}$$

D. Exponential Sampling Distribution

For the case of exponential sampling distribution, the sampling density function is

$$p(\mathbf{x}) = \frac{1}{\mu}\exp\left(\frac{x}{\mu}\right). \tag{20}$$

In this case, the "significance" (11) can be estimated as

$$\begin{aligned} E_{\text{sig}}(k) &= \frac{\|\alpha_k\|_q}{\mu^{1/q}}\left(\int_X x\exp\left(-\frac{q(x-\mu_k)^2}{\sigma_k^2}+\frac{x}{\mu}\right)dx\right)^{1/q} \\ &= \frac{\|\alpha_k\|_q}{\mu^{1/q}}\exp\left(-\frac{4q\mu_k\mu+\sigma_k^2}{4q^2\mu^2}\right) \\ &\quad\times\left(\int_X \exp\left(-\frac{q}{\sigma_k^2}\left(x-\frac{2q\mu_k\mu+\sigma_k^2}{2q\mu}\right)^2\right)dx\right)^{1/q} \\ &\approx \|\alpha_k\|_q\cdot\left(\sqrt{\frac{\pi}{q}}\cdot\frac{\sigma_k}{\mu}\right)^{1/q}\exp\left(-\frac{\mu_k}{\mu}-\frac{\sigma_k^2}{4q\mu^2}\right). \end{aligned} \tag{21}$$

In GGAP-RBF algorithm, this definition of significance of a neuron (formula (11) or (12), or its special cases such as uniform sampling case (14), normal sampling case (17), Rayleigh sampling case (19), and exponential sampling case (21)) are used for the growing and pruning criteria for hidden neurons as indicated below.

Note: In real practical applications, the sampling distribution $p(\mathbf{x})$ could be one of these distributions or a combination of some of these distributions according to (12). For example, as shown in Section IV, the sampling distribution of a real large-scale complex application "California Housing" can be simply approximated by a combination of all these four typical distributions.

III. Proposed GGAP-RBF Learning Algorithm

In this section, the main ideas behind the GGAP-RBF algorithm are described first, followed by the actual algorithm in a pseudo code form.

A. Growing Criterion

The learning process of GGAP-RBF involves the allocation of new hidden neurons as well as adaptation of network parameters. The RBF network begins with no hidden neurons. As inputs are received sequentially during training, some of them may initiate new hidden neurons based on a growing criterion as follows:

$$\begin{cases} \|\mathbf{x}_n-\mu_{n\mathbf{r}}\| > \epsilon_n \\ \|e_n\|_q\left(\int_X \exp\left(-\frac{q\|\mathbf{x}-\mathbf{x_n}\|^2}{\kappa^2\|\mathbf{x}_n-\mu_{n\mathbf{r}}\|^2}\|\right)p(\mathbf{x})\,d\mathbf{x}\right)^{1/q} > e_{\min} \end{cases} \tag{22}$$

where $\mathbf{x}_n$ is the latest input received, $\mu_{n\mathbf{r}}$ is the center of the hidden neuron nearest (in the Euclidean distance sense) to $\mathbf{x}_n$. $e_{\min}$ is the expected approximation accuracy and ϵ_n is a threshold to be selected appropriately.

When a new neuron is added, the parameters associated with it are given by

$$\begin{cases} \alpha_{K+1} = e_n \\ \mu_{K+1} = \mathbf{x}_n \\ \sigma_{K+1} = \kappa\|\mathbf{x}_n-\mu_{n\mathbf{r}}\| \end{cases} \tag{23}$$

where $e_n = \mathbf{y}_n - f(\mathbf{x}_n)$ and κ is an overlap factor that determines the overlap of the responses of the hidden neurons in the input space.

Remark: The first criterion ensures that a new neuron is only added if the input data is sufficiently far from the existing neurons. The second criterion ensures that the significance of the newly added neuron obtained by substituting (23) in (11) is greater than the required approximation accuracy $e_{\min}$. Also the second criterion subsumes the novelty criterion used in RAN, RANKEF, and MRAN where the condition $\|e_n\| > e_{\min}$ is obviously implied in our enhanced growing criterion $\|e_n\|_q(\int_X \exp(-(q\|\mathbf{x}-\mathbf{x_n}\|^2/\kappa^2\|\mathbf{x}_n-\mu_{n\mathbf{r}}\|^2\|)p(\mathbf{x})\,d\mathbf{x})^{1/q} > e_{\min}$ based on the significance of newly intentionally added neuron.

B. Pruning Criterion

If the significance of neuron k is less than the approximation accuracy $e_{\min}$, neuron k is insignificant and should be removed, otherwise, neuron k is significant and should be retained. Given the approximation accuracy $e_{\min}$, neuron k will be pruned if

$$\begin{aligned}E_{\text{sig}}(k) &= \lim_{n\to+\infty} E_{\text{ave}}(k)\\ &= \|\alpha_k\|_q \left(\int_X \exp\left(-\frac{q\|\mathbf{x}-\mu_k\|^2}{\sigma_k^2}\right) p(\mathbf{x})\,d\mathbf{x}\right)^{1/q} < e_{\min}.\end{aligned} \tag{24}$$

The above condition implies that after learning each observation, the significance for all neurons should be computed and checked for possible pruning. This will be a computationally intensive task. But it is shown in the next subsection that only the nearest neuron is possibly insignificant and need to be checked for pruning and there is no need to compute the significance of all neurons in the network.

C. Nearest Neuron for Parameter Adjustment and Pruning

In order to increase the learning speed further it can be shown that, one needs only to adjust parameters of the neuron nearest (in the Euclidean distance sense) to the most recently received input if no new neuron is added and only needs to check the nearest (most recently adjusted) neuron for pruning. It is neither necessary to adjust the parameters for all neurons nor necessary to check all neurons for possible pruning. At any time instant, only the *single* nearest neuron needs to be adjusted or needs to be checked for pruning. The rationale for this is given as follows.

Compared to the Gaussian function $\phi(x) = \exp(-(x^2/\sigma^2))$, its first and second derivatives will approach zero faster when $x > \sigma$. Thus, in the gradient vector $\mathbf{a}_n$ of EKF [9] all elements except $\phi_{n\mathbf{r}}(\mathbf{x}_n), \phi_{n\mathbf{r}}(\mathbf{x}_n)(2\alpha_{n\mathbf{r}}/\sigma_{n\mathbf{r}}^2)(\mathbf{x}_n - \mu_{n\mathbf{r}})^T, \phi_{n\mathbf{r}}(\mathbf{x}_n)(2\alpha_{n\mathbf{r}}/\sigma_{n\mathbf{r}}^3)\|\mathbf{x}_n - \mu_{n\mathbf{r}}\|^2$ will approach zero quickly when $x > \sigma$.

This would dramatically increase the learning speed by removing the "curse of dimensionality" of RANEKF and MRAN but without losing learning performance in most cases. In fact, when an observation is presented, RANEKF and MRAN needs to calculate Kalman gain vector and error covariance matrix by using a $3K$ gradient vector, where K is the number of neurons obtained so far. When K becomes large, this would result in a larger learning time and possibly make ordinary computers incapable of learning because of lower memory configuration, etc. However, since only one neuron's parameters are adjusted at any time, GGAP-RBF only needs to do simple matrix operation on a very small matrix.

Suppose that after sequentially learning n observations, a RBF network with K neurons has been obtained. Obviously all these K neurons should be significant since insignificant neurons would have been pruned after learning the nth observation. If a new $(n+1)$th observation $(\mathbf{x}_{n+1}, \mathbf{y}_{n+1})$ arrives and the growing criteria (22) is satisfied, a new significant neuron $K+1$ will be added. The parameters of all the rest neurons remain unchanged and those neurons will remain significant after learning the $(n+1)$th 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 $(\mathbf{x}_{n+1}, \mathbf{y}_{n+1})$ arrives and the growing criteria (22) is not satisfied, no new neuron will be added and only the parameters of the nearest neuron will be adjusted. Since the parameters of all the neurons except for the nearest one remain unchanged, those neurons except for the nearest one will remain significant after learning the $(n+1)$th observation. After the parameters of the nearest neuron are adjusted, if the nearest neuron becomes insignificant it should be removed. That means, if only the parameters of the nearest neuron are adjusted after each observation during sequential learning, one needs to check only whether the nearest neuron becomes insignificant after adjustment. As for the parameter adjustment, the adjustment will be done for the parameters of the nearest neuron using the EKF algorithm. In EKF algorithm, the Kalman gain vector computation becomes dramatically simpler because we are adjusting only one neuron. (Refer to [9] for EKF equation details.)

Thus, a new simple efficient GGAP-RBF algorithm suitable for multi-input–multi-output (MIMO) applications and with any sampling density function $p(\mathbf{x})$ is summarized below:

Proposed GGAP-RBF Algorithm: For each observation $(\mathbf{x}_n, \mathbf{y}_n)$ presented to the network, where $\mathbf{x}_n \in X \subseteq \mathbf{R}^l$ and $n = 1, 2, \ldots$, do

1. **compute** the overall network output

$$f(\mathbf{x}_n) = \sum_{k=1}^{K} \alpha_k \exp\left(-\frac{1}{\sigma_k^2}\|\mathbf{x}_n - \mu_k\|^2\right) \tag{25}$$

where K is the number of hidden neurons.

2. **calculate** the parameters required in the growth criterion

$$\begin{aligned}\epsilon_n &= \max\{\epsilon_{\max}\gamma^n, \epsilon_{\min}\}, \quad (0 < \gamma < 1)\\ e_n &= \mathbf{y}_n - f(\mathbf{x}_n)\end{aligned} \tag{26}$$

3. **apply** the criterion for adding neurons

If

$$\|\mathbf{x}_n - \mu_{n\mathbf{r}}\| > \epsilon_n \quad \text{and}$$
$$\|e_n\|_q \left(\int_X \exp\left(-\frac{q\|\mathbf{x}-\mathbf{x}_n\|^2}{\kappa^2\|\mathbf{x}_n - \mu_{n\mathbf{r}}\|^2}\right) p(\mathbf{x})\,d\mathbf{x}\right)^{1/q} > e_{\min}$$

allocate a new hidden neuron $K+1$ with

$$\begin{aligned}\alpha_{K+1} &= e_n\\ \mu_{K+1} &= \mathbf{x}_n\\ \sigma_{K+1} &= \kappa\|\mathbf{x}_n - \mu_{n\mathbf{r}}\|.\end{aligned} \tag{27}$$

Else

adjust the network parameters $\alpha_{n\mathbf{r}}, \mu_{n\mathbf{r}}, \sigma_{n\mathbf{r}}$ for the nearest neuron only.

check the criterion for pruning the adjusted hidden neuron:

If

$$\begin{aligned}&E_{\text{sig}}(n\mathbf{r})\\ &= \|\alpha_{n\mathbf{r}}\|_q \left(\int_X \exp\left(-\frac{q\|\mathbf{x}-\mu_{n\mathbf{r}}\|^2}{\sigma_{n\mathbf{r}}^2}\right) p(\mathbf{x})\,d\mathbf{x}\right)^{1/q} < e_{\min}\end{aligned}$$

remove the $n\mathbf{r}$th hidden neuron
reduce the dimensionality of EKF

Endif

Endif

IV. Benchmark Problems Performance Comparison

In this section, the performance comparison of GGAP-RBF with three other popular sequential algorithms (RAN, RANEKF, and MRAN) and the popular SVM algorithm SVR are presented for two benchmark problems in the function approximation area: 1) California Housing and 2) Mackey–Glass chaotic time series prediction. The sampling density function for California Housing[1] can be approximated by a combination of the four typical distributions discussed in Section II of this paper, namely: uniform, normal, exponential and Rayleigh sampling distributions. For the Mackey–Glass problem, the input distribution was assumed as uniform. All the simulations on RAN, RANEKF, GGAP-RBF, and MRAN are carried out in Matlab 6.5 environment running in an ordinary PC.

Two learning accuracy measurements are used in all the simulation experiments. mean arithmetic error (MAE) is defined as

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} \|f(\mathbf{x}_i) - \mathbf{y}_i\|_1}{n} \tag{28}$$

and root mean square error (RMSE) is defined as

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} \|f(\mathbf{x}_i) - \mathbf{y}_i\|_2^2}{n}}. \tag{29}$$

Simulation results are presented for both 1-norm growing and pruning approach [$q = 1$ in "significance" (11)], and 2-norm growing and pruning approach [$q = 2$ in "significance" (11)]. In all simulations, MAE and RMSE of training and testing samples are calculated and compared for both (1-norm) and (2-norm) GGAP-RBF.

A. California Housing

California Housing is a dataset obtained from the StatLib repository. There are 20640 observations for predicting the price of houses in California. Information on the variables were collected using all the block groups in California from the 1990 Census. In this sample a block group on average includes 1425.5 individuals living in a geographically compact area. Naturally, the geographical area included varies inversely with the population density. Distances among the centroids of each block group were computed as measured in latitude and longitude. All the block groups reporting zero entries for the independent and dependent variables were excluded. The final data contained 20640 observations on nine variables, which consists of eight continuous inputs (median income, housing median age, total rooms, total bedrooms, population, households, latitude, and longitude) and one continuous output (median house value).[2]

For simplicity, the eight input attributes and one output have been normalized to the range $[0, 1]$ in our experiment. The expected approximation accuracy is $e_{\min} = 0.0001$. For this problem, 8000 training data and 12640 testing data are randomly generated from the California Housing database. Performance among GGAP-RBF, RAN, RANEKF, MRAN, and SVR are compared for this problem. The parameters for GGAP-RBF, RAN, RANEKF, and MRAN algorithms are chosen as: $\epsilon_{\max} = 1.15, \epsilon_{\min} = 0.04, \kappa = 0.10$, and $\gamma = 0.999$. In addition to these parameters, for GGAP-RBF,

[1]http://www.niaad.liacc.up.pt/~ltorgo/Regression/cal_housing.html
[2]http://www.niaad.liacc.up.pt/~ltorgo/Regression/cal_housing.html

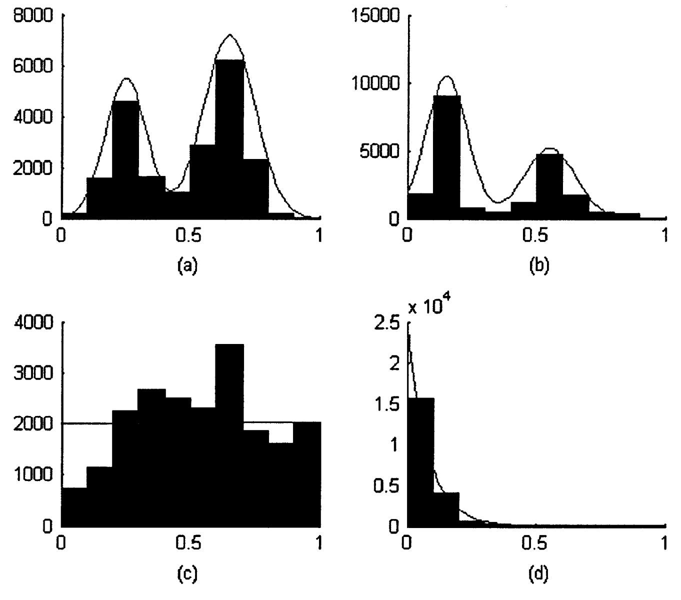

Fig. 1. Histogram shows the distribution of California Housing training samples: attributes 1–4. (a) Histogram of attribute 1. (b) Histogram of attribute 2. (c) Histogram of attribute 3. (d) Histogram of attribute 4.

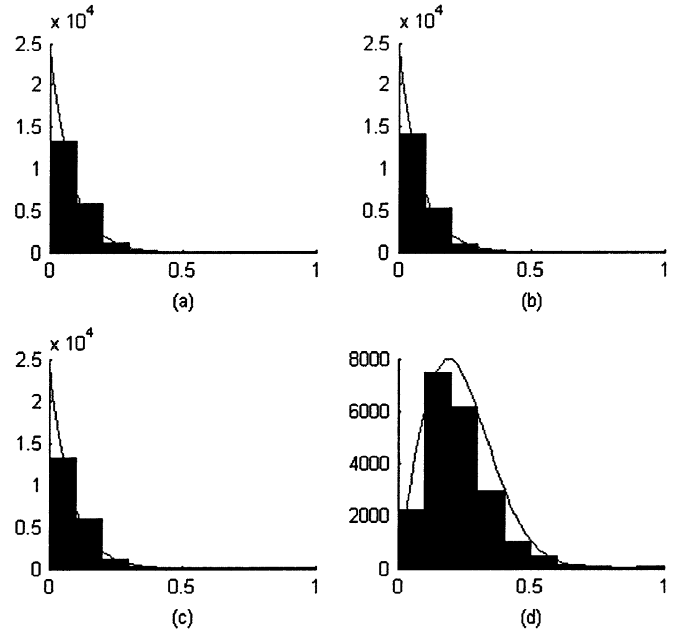

Fig. 2. Histogram shows the distribution of California Housing training samples: attributes 5–8. (a) Histogram of attribute 5. (b) Histogram of attribute 6. (c) Histogram of attribute 7. (d) Histogram of attribute 8.

RANEKF, and MRAN we set $Q_0 = 0.000\,01$ and $P_0 = 0.9$. The simulations for SVR are carried out using well-known compiled C-coded SVM packages: LIBSVM [20] running in the same PC.

For SVR algorithm, we set $C = 500$ after several trial-and-error runs. For 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 $M = 60$ after several trial-and-error runs. After analyzing the distributions of the eight input attributes as shown in Figs. 1 and 2, it was found that the distribution of the first two input attributes are binormal, and the distribution of the third can be roughly approximated by a uniform one. The distributions of the input attributes 4–7 can be approximated by exponential distributions and the distribution of the last attribute was approximated by a Rayleigh distribution. In fact, after simple estimation, the sampling density functions $p_i(x)$ $(i = 1,\ldots,8)$ of these eight attributes can simply be approximated, respectively, as follows:

$$\begin{aligned}
p_1(x) &= 0.5\left(\frac{1}{\sqrt{2\pi}\times 0.08}\exp\left(-\frac{(x-0.25)^2}{2\times 0.08^2}\right)\right.\\
&\quad\left.+\frac{1}{\sqrt{2\pi}\times 0.10}\exp\left(-\frac{(x-0.65)^2}{2\times 0.10^2}\right)\right)\\
p_2(x) &= 0.5\left(\frac{1}{\sqrt{2\pi}\times 0.08}\exp\left(-\frac{(x-0.15)^2}{2\times 0.08^2}\right)\right.\\
&\quad\left.+\frac{1}{\sqrt{2\pi}\times 0.10}\exp\left(-\frac{(x-0.55)^2}{2\times 0.10^2}\right)\right)\\
p_3(x) &= 1\\
p_4(x) &= \frac{1}{0.0675}\exp\left(\frac{x}{0.0675}\right)\\
p_5(x) &= \frac{1}{0.0838}\exp\left(\frac{x}{0.0838}\right)\\
p_6(x) &= \frac{1}{0.0501}\exp\left(\frac{x}{0.0501}\right)\\
p_7(x) &= \frac{1}{0.0825}\exp\left(\frac{x}{0.0825}\right)\\
p_8(x) &= \frac{x}{0.1880^2}\exp\left(-\frac{x^2}{2\times 0.1880^2}\right).
\end{aligned}$$

During this simulation, we assume that the eight attributes are independent. Thus, according to (12), the significance of neuron k defined in the GGAP-RBF algorithm

$$E_{\mathrm{sig}}(k) = \|\alpha_k\|_q \prod_{i=1}^{8}\left(\int_0^1 \exp\left(-\frac{q(x-\mu_{k,i})^2}{\sigma_k^2}\right)p_i(x)\,dx\right)^{1/q} \tag{30}$$

can be further easily calculated by estimating $(\int_0^1 \exp(-(q(x-\mu_{k,i})^2/\sigma_k^2))p_i(x)\,dx)^{1/q}, i = 1,\ldots,8$, according to normal sampling case (17) (if $i = 1, 2$), uniform sampling case (14) (if $i = 3$), exponential sampling case (19) (if $i = 4, 5, 6, 7$), and Rayleigh sampling case (21) (if $i = 8$), respectively.

Table I shows that GGAP, MRAN, RANEKF, RAN, and SVR achieve comparable generalization performance. SVR spent 164.8440 s (running in C executable environment faster than MATLAB environment)[3] obtaining 2429 support vectors. Although by setting different values for parameter C, SVR could achieve higher learning speed for this application, but the learning accuracy would become worse and the number of support vectors would become higher. RAN spent 3505.2 s obtaining a network with 3552 neurons and MRAN spent 2891.5 s obtaining a smaller network with 64 neurons. During the simulations for this application, it was found that it was difficult to get all the data trained by RANEKF. Similar to RAN, RANEKF obtained large number of neurons for this application. Since only one neuron's parameters are adjusted at any time, GGAP-RBF needs to do simple matrix operation on a very small matrix. GGAP-RBF for 1-norm and 2-norm growing and pruning approaches, respectively, spent only 115.34 s and 191.23 s obtaining much smaller networks with 18 and 24 neurons only. If one notices that C executable implementation is in general much faster than MATLAB implementation by a factor up to 50, it then becomes clear that for this application GGAP-RBF runs much faster than SVR. Fig. 3 shows the learning speed comparison of different learning algorithms (except for SVR) for this large scale complex problem and Fig. 4 shows the neuron updating history. Since SVR is not a sequential algorithm, unlike the rest of the sequential algorithms, the per-observation learning information is not applicable.

[3]Refer to http://www.mathworks.com/products/compiler/examples/example2.shtml for compiler execution speed comparison.

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT ALGORITHMS FOR A REAL LARGE-SCALE COMPLEX APPLICATION: CALIFORNIA HOUSING

Algorithms	CPU Time(s)	Training Error		Testing Error		No. of
		MAE	RMSE	MAE	RMSE	Neurons/SVs
GGAP (1-norm)[a]	115.34	0.10468	0.14169	0.1025	0.13861	18
GGAP (2-norm)[a]	191.23	0.10208	0.13905	0.10361	0.14038	24
MRAN[a]	2891.5	0.11452	0.15984	0.11397	0.15859	64
RANEKF[a,c]	14181	0.029104	0.073576	0.11159	0.14952	200
RAN[a]	3505.2	0.071877	0.10829	0.11103	0.15305	3552
SVR[b]	164.8440	-	0.1252	-	0.1282	2429

[a] run in MATLAB environment. [b] run in C executable environment.

[c] when 2337 samples were learned.

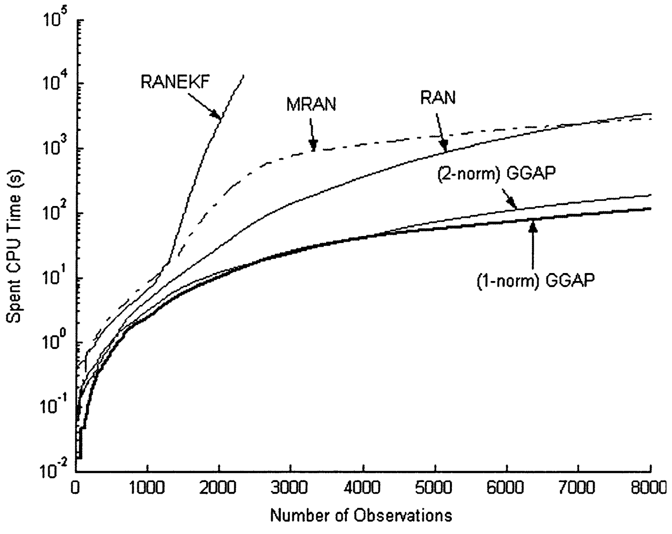

Fig. 3. Learning speed comparison of different learning algorithms for California Housing application.

B. Chaotic Time Series (Mackey–Glass) Prediction

One possible application of GGAP-RBF is to predict complex time series, a special function approximation problem. As mentioned by Platt [1], the need to time series arises in such real-world problems as detecting arrhythmia in heartbeats. The chaotic Mackey–Glass differential delay equation is recognized

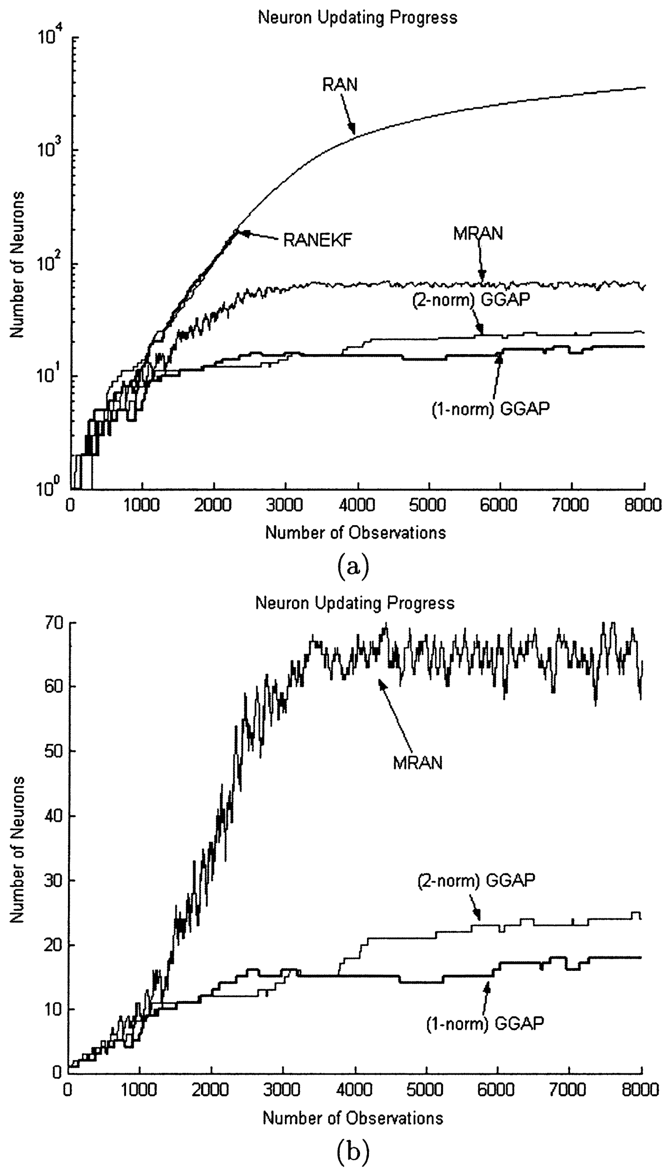

Fig. 4. Neuron updating progress for California Housing application. (a) Different algorithms. (b) Details for GGAP-RBF and MRAN algorithms only.

as one of the benchmark time series problems, which is generated from the following delay differential equation:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \tag{31}$$

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

$$x(t+\Delta t) = \frac{2-b\Delta t}{2+\Delta t}x(t) + \frac{a\Delta t}{2+b\Delta t}\left[\frac{x(t+\Delta t-\tau)}{1+x^{10}(t+\Delta t-\tau)} + \frac{x(t-\tau)}{1+x^{10}(t-\tau)}\right]. \tag{32}$$

As given by Yingwei *et al.* [9], setting $\Delta t = 1$, the time series is generated under the condition $x(t-\tau) = 0.3$ for $0 \le t \le \tau$ $(\tau = 17$ in our test case). The series is predicted

TABLE II
PERFORMANCE COMPARISON OF DIFFERENT ALGORITHMS FOR A TIME SERIES APPLICATION: MACKEY–GLASS

Algorithms	CPU Time(s)	Training Error		Testing Error		No. of
		MAE	RMSE	MAE	RMSE	Neurons
GGAP (1-norm)	24.326	0.034363	0.069952	7.1962e-5	0.036771	13
GGAP (2-norm)	28.868	0.029703	0.066737	1.085e-4	0.031202	19
MRAN	57.205	0.040308	0.11012	1.0867e-4	0.03373	16
RANEKF	62.674	0.026464	0.072586	6.2671e-5	0.023962	23
RAN	58.127	0.047319	0.1006	1.3257e-4	0.046623	39

with $v = 50$ sample steps ahead using the four past samples: $s_{n-v}, s_{n-v-6}, s_{n-v-12}$, and s_{n-v-18}. Hence, the nth input–output data for the network to learn are

$$\mathbf{x}_n = [s_{n-v}, s_{n-v-6}, s_{n-v-12}, s_{n-v-18}]^T$$
$$y_n = s_n \tag{33}$$

whereas the step-ahead predicted value (the output of the network) at time $n-v$ is given by

$$z_n = f(\mathbf{x}_n). \tag{34}$$

The v step-ahead prediction error is

$$e_n = s_n - z_n. \tag{35}$$

Similar to [1] and [9], the parameter values are selected as follows: $e_{\min} = 0.01, \kappa = 0.87, \epsilon_{\max} = 0.7, \epsilon_{\min} = 0.07$. For MRAN, the growing threshold and growing/pruning window size are chosen as 0.04 and 90, respectively. For GGAP-RBF, simply assume that the observations are drawn uniformly from the range $[0.4, 1.4]^4$ and, thus, set $S(X) = 1$.

In this test case, the network was trained by the observations up to time $t = 4000$ and the unknown observations from time $t = 4001$ to $t = 4500$ were used as testing observation to test the prediction of the network. Table II shows the learning speed, training accuracy, prediction performance and the obtained neurons. It can be seen that GGAP-RBF, MRAN, and RANEKF can get comparable prediction performance on the long-term prediction. However, GGAP-RBF learns much faster than other algorithms and obtains smaller network architectures. Figs. 5 and 6 display the neuron updating history during the online learning phase and the approximated time series curve for both 1-norm and 2-norm cases.

Remark: It has been found that the networks obtained by RAN and RANEKF become large in some simulations, indicating that RAN and RANEKF without pruning approaches may not be able to cope with real practical applications in an ordinary computing environment. Compared with RAN and RANEKF, MRAN can obtain a more compact network. But its learning accuracy and generalization performance may not be very good unless the growing-pruning window size is chosen properly. Choosing the proper size for the window can only be done by trial and error based on exhaustive simulation studies. If several trials are conducted, the total time spent on whole

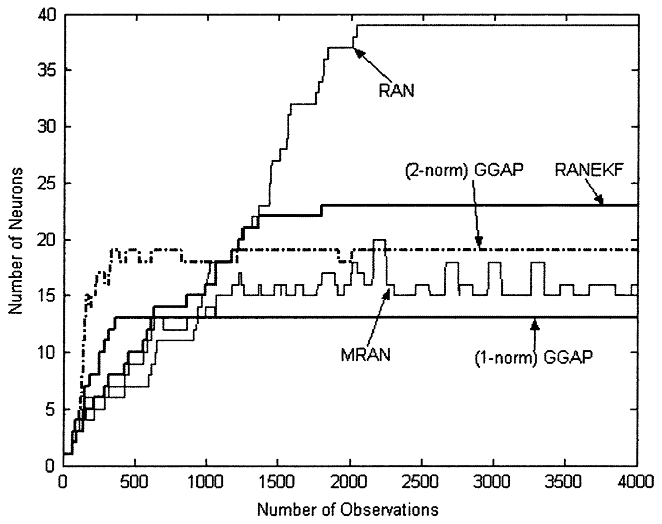

Fig. 5. Neuron updating progress for Mackey–Glass time series prediction.

training stage would be possibly large. For example, for the California Housing application, one trial of MRAN would take up to 50 min. In order to get proper parameters, at least six trials were conducted in our experiment and the actual training time we spent was much more than 300 min, compared to the extremely short training time (<200 s) spent for our new proposed algorithm.

V. Conclusion

In this paper, a sequential learning algorithm called GGAP-RBF for any arbitrary distribution of input training samples/data is presented for function approximation. Although RANEKF may obtain large networks in some applications, EKF based parameter adjustment method introduced by Kadirkamanathan and Niranjan [2] still plays the key role in both MRAN and GGAP-RBF. The key difference among the parameter adjustments of RANEKF, MRAN and ours is that GGAP-RBF reduces the computation complexity by only adjusting the parameters of the nearest neuron every time instead of all neurons without losing learning performance.

Using the idea of significance of neurons, which is **quantitatively** defined from a statistical viewpoint as the average information content of a neuron and also the contribution of that neuron to the overall performance of the RBF network, the proposed algorithm adds and prunes hidden neurons smoothly and produces much more compact networks than other popular sequential learning algorithms. The GGAP-RBF algorithm can be used for any arbitrary input sampling distribution and the simulation results have been provided for uniform, normal, Rayleigh, and exponential sampling cases.

Although Gaussian RBF has been used and tested in this paper, it should be noted that the "significance" concept [(11) and (12)] introduced in this paper can be used for other radial basis functions as well. The convergence and generalization performance of these "significance" based radial basis function networks is an interesting topic for further research.

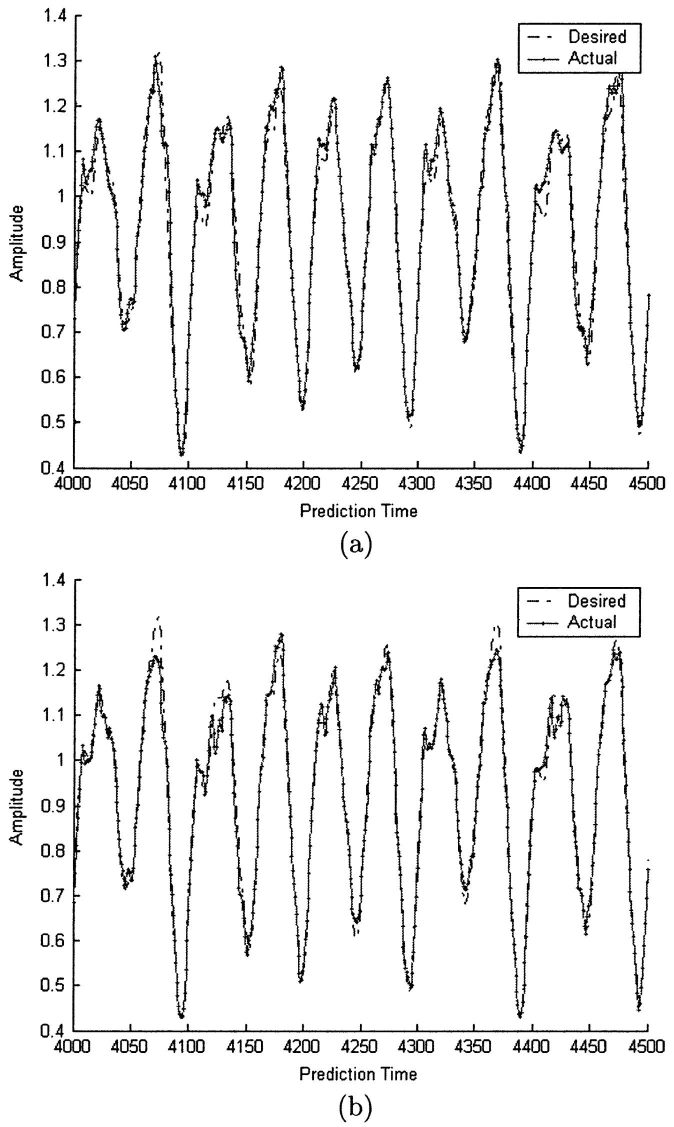

Fig. 6. Approximated time series curves. (a) (1-norm) GGAP-RBF. (b) (2-norm) GGAP-RBF. The network was trained by the observations up to time $t = 4000$ and used to predict the time series curve from time $t = 4001$ to $t = 4500$.

References

[1] J. Platt, "A resource-allocating network for function interpolation," *Neural Computat.*, vol. 3, pp. 213–225, 1991.
[2] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Computat.*, 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 Netw.*, vol. 8, no. 6, pp. 1492–1506, Nov. 1997.
[4] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. Neural Netw.*, vol. 2, no. 2, pp. 302–309, Mar. 1991.
[5] S. Chen, E. S. Chng, and K. Alkadhimi, "Regularized orthogonal least squares algorithm for constructing radial basis function networks," *Int. J. Control*, vol. 64, no. 5, pp. 829–837, 1996.
[6] E. S. Chng, S. Chen, and B. Mulgrew, "Gradient radial basis function networks for nonlinear and nonstationary time series prediction," *IEEE Trans. Neural Netw.*, vol. 7, no. 1, pp. 190–194, Jan. 1996.
[7] M. J. L. Orr, "Regularization on the selection of radial basis function centers," *Neural Computat.*, vol. 7, pp. 606–623, 1995.

[8] A. G. Bors and M. Gabbouj, "Minimal topology for a radial basis functions neural network for pattern classification," *Dig. Signal Process.*, vol. 4, pp. 173–188, 1994.

[9] L. Yingwei, N. Sundararajan, and P. Saratchandran, "A sequental learning scheme for function approximation using minimal radial basis function (RBF) neural networks," *Neural Computat.*, vol. 9, pp. 461–478, 1997.

[10] ——, "Performance evaluation of a sequental minimal radial basis function (RBF) neural network learning algorithm," *IEEE Trans. Neural Netw.*, vol. 9, no. 2, pp. 308–318, Mar. 1998.

[11] M. Salmerón, J. Ortega, C. G. Puntonet, and A. Prieto, "Improved RAN sequential prediction using orthogonal techniques," *Neurocomput.*, vol. 41, pp. 153–172, 2001.

[12] 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," *Neurocomput.*, vol. 42, pp. 267–285, 2002.

[13] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," in *Neural Information Processing Systems 9*, M. Mozer, J. Jordan, and T. Petscbe, Eds. Cambridge, MA: MIT Press, 1997, pp. 155–161.

[14] S. Vijayakumar and S. Wu, "Sequential support vector classifiers and regression," in *Proc. Int. Conf. Soft Computing (SOCO'99)*, Genoa, Italy, pp. 610–619.

[15] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Microsoft, Res. Tech. Rep. MSR-TR-98-14, 1998.

[16] N. Sundararajan, P. Saratchandran, and L. Yingwei, *Radial Basis Function Neural Networks with Sequential Learning: MRAN and Its Applications*, Singapore: World Scientific, 1999.

[17] J. N. Franklin, "Determinants," in *Matrix Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1968, pp. 1–25.

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

[19] B. Todorović and M. Stanković, "Sequential growing and pruning of radial basis function network," in *Proc. Int. Joint Conf. Neural Networks (IJCNN'01)*, Washington, DC, Jul. 15–19, pp. 1954–1959.

[20] C.-C. Chang and C.-J. Lin. (2003) LIBSVM—A library for support vector machines. Dept. Comput. Sci. Inform. Eng., National Taiwan Univ., Taiwan, R.O.C. [Online]. Available: Available 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, P.R. China, in 1991 and 1994, respectively, and the Ph.D. degree in electrical engineering from Nanyang Technological University, Singapore, in 1999.

From 1998 to 2001, he worked as Research Fellow at the Singapore Institute of Manufacturing Technology (formerly known as Gintic Institute of Manufacturing Technology) where he has led/implemented several key industrial projects. Since 2001, he has been working as 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 in electrical engineering from Regional Engineering College, Calicut, India, the M.Tech. degree in electrical engineering from the Indian Institute of Technology, Kharagpur, India, the M.Sc. degree in systems enginering from City University, London, U.K., and the Ph.D. degree in control engineering from Oxford University, Oxford, U.K.

He worked for two years as 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 the Nanyang Technological University, Singapore, where he is now an Associate Professor. He has several publications in refereed journals and has authored four books titled *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). He is also an editor for the journal *Neural Parallel and Scientific Computations.* His research interests are in neural networks, parallel computing, and control.

Narasimhan Sundararajan (S'73–M'74–SM'84–F'96) received the B.E. degree in electrical engineering from the University of Madras, India, in 1966, the M.Tech. degree from the Indian Institute of Technology, Madras, India, in 1968, and the Ph.D. degree in electrical engineering from the University of Illinois, Urbana-Champaign, in 1971.

From 1972 to 1991, he worked at the Indian Space Research Organization and NASA on aerospace problems. Since 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 from 2003 to 2004 at the Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India. He has published more than 100 papers and four books in the area of neural networks: *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). He has served as as Associate Editor of the *IFAC Journal on Control Engineering Practice*. His research interests are in the areas of neural networks and control with aerospace applications.

Dr. Sundararajan is a an Associate Fellow of AIAA and has served as an Associate Editor for the IEEE TRANSACTIONS on CONTROL SYSTEMS TECHNOLOGY.