

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

Guang-Bin Huang\*, *Member, IEEE*, P. Saratchandran\*, *Senior Member, IEEE*,  
and Narashiman 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 is applicable 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 and also the support

---

Revised and resubmitted to *IEEE Trans. on Neural Networks*. Paper no.:TNN# **CE758-Rev**

\*The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798. E-Mail: {egbhuang, epsarat, ensundara}@ntu.edu.sg

vector regression method in terms of learning speed, network size and generalization performance regardless of the sampling density function of the training data.

## 1 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. Selection of a learning algorithm for a particular application is crucially 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 batch learning algorithms, sequential learning algorithms we will discuss in this paper have the following distinguishing features:

- 1) All the training observations will be *sequentially one by one* presented to the learning system.
- 2) At any time, *only one* training observation can be 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.

Thus, based on the abovementioned major features of sequential algorithms, many of the existing algorithms are not sequential if one takes a close look at the schemes. One

major bottleneck seems to be that they need the entire training data ready for training before a training procedure starts and thus not really sequential. This is highlighted with a brief review of these algorithms as given below.

Although algorithms proposed by Chen, et al[1, 2] and Chng, et al[3] can add neurons to the network one by one and obtain more compact networks than conventional RBF networks, it needs to select a subset network from a  $N$ -term full network based on the some orthogonal subset selection implementation (**Orthogonal Least Square (OLS)**[1] or **Regularized Orthogonal Least Square (OLS)**[2]), where the  $N$ -term full network is formed when all the  $N$  training observations are presented.

Based on the subset selection technique [1, 2], Orr[4] proposed the **Regularized Forward Selection (RFS)** algorithm for RBF network, which combines forward subset selection and zero-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 to each new problem, RFS has only one preset parameter, the basis function width. The implementation of Orr[4] still depends on the data being available all at once and hence 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 and usually is very large. Thus, it is computationally intensive in most practical applications.

The pattern classification algorithm proposed by Bors and Gabbouj[5] 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 (refer to equations (13) and (19) of Bors and Gabbouj[5]). Obviously, all these algorithms including **Support Vector Machine (SVM)** and its variants[6, 7, 8] depend on the data being available all

at a time. Thus, strictly speaking, all the learning algorithms mentioned above are not sequential learning algorithms but variations of batch learning algorithms only.

A significant contribution to sequential learning was made by Platt[9] through the development of **R**esource **A**llocation **N**etwork (RAN) in which hidden neurons are added sequentially based on the novelty of the new data. Key enhancement of RAN, known as RANEKF was proposed by Kadiramanathan and Niranjana[10] in which **E**xtended **K**alman **F**ilter (EKF) rather than **L**east **M**ean **S**quare (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. Similar to HSOL, RAN and RANEKF can only add neurons to the network and can't prune insignificant neurons from the network. A significant improvement to RAN and RANEKF was made by Yingwei, et al[11, 12] 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[13]. Other methods for pruning in RBF networks have been proposed in Salmerón, et al[14] and Rojas, et al[15].

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 Salmerón, et al[14] QR factorization and singular value decomposition methods are used for determining the structure as well as for pruning the network. It has been shown[14] that the size of the network is more compact than RAN. However, these methods require additional computational efforts. Quite a large number of parameters/variables (around

15) need to be preset (by trial and error) and training data need to be stored and reused for pruning purpose.

To realize a compact RBF network, similar to Salmerón, et al[14], Rojas, et al[15] presents a pruning scheme which checks the pruning criteria for *all* hidden neurons *only after all* 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 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[15] is also computationally expensive.

One important point to be noted in all of 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  $(\mathbf{y}_1, \dots, \mathbf{y}_n)^T$  and the approximating network's output vector  $(f(\mathbf{x}_1), \dots, f(\mathbf{x}_n))^T$  for  $n$  training observations  $(\mathbf{x}_i, \mathbf{y}_i)$ , i.e.,  $(\frac{\sum_{i=1}^n \|f(\mathbf{x}_i) - \mathbf{y}_i\|_q^q}{n})^{1/q}$ . (Refer to Franklin[16] 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[17], and the sequential growing and pruning algorithm proposed by Todorović and Stanković[18] since all of them add new neurons based on their novelty to the individual instant input 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[17] and the sequential growing and pruning algorithm proposed by Todorović and Stanković[18] prune neurons mainly based on their significance to the individual instant input observations, and are complicated in both theory and implementation.

Our concept of significance is also wholly different from and much simpler than that of Todorović and Stanković[18] where it is defined based on the sensitivity of a neuron's width and connection weight to the output error for the current input data. However, the significance proposed in this paper is simply 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. For growing (apart

from the distance criteria used in MRAN), this algorithm uses a criterion based on the significance of the neuron. 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. Further, for both growing and pruning, it is shown that one needs to check only the nearest neuron (based on the Euclidean distance to the latest 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. 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 learned, and the learning (parameter adjustment, network growing or pruning) is conducted whenever a new observation is presented.

A significance difference between the work of Salmerón, et al[14] and Rojas, et al[15] and ours is: Salmerón, et al[14] and Rojas, et al[15] do not conduct pruning during learning stage. As shown in Section 3, algorithms without pruning functions during learning stage could possibly not be able to learn all the observations in some large scale complex applications since very large networks could possibly be obtained during the learning stage, which results in lacking memory and lower learning speed and could causes learning failure. In our new proposed algorithm, the pruning is checked and conducted through the whole learning phase and the network architecture is remained as compact and necessary as possible during the entire learning phase, thus, large scale complex applications with the proposed algorithm can be implemented in ordinary computers as well.

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 a number of problems. The results indicate the superior performance of GGAP-RBF for all the problems studied. Also, recently a fast implementation of SVM for Regression (SVR) based on Sequential Minimal Optimization (SMO)[8] has become popular. We have compared out GGAP-RBF with this SMO even though GGAP-RBF is a sequential algorithm whereas SMO is not a truly sequential algorithm defined in this paper. SMO only sequentially adjusts Lagrange multipliers and data are still input batch by batch. The learning phase can only start when all data are ready and no new data are added during learning phase. The simulation results on real large complex applications show that the new proposed GGAP-RBF algorithm seems to be faster and to provide much smaller networks than SMO, yet provides comparable generalization performance.

The rest of this paper is organized as follows. Section 2 describes the new **G**eneralized **G**rowing **A**nd **P**runing RBF (GGAP-RBF) learning algorithm derived for arbitrary sampling density functions and general  $q$ -norm learning accuracy. Section 3 presents the performance comparison results for GGAP-RBF along with RAN, RANEKF, and MRAN for a number of typical artificial and real large scale complex function approximation problems. The performance comparison of GGAP-RBF and SVR on real large scale complex benchmarking problems is also presented. Section 4 summarizes the conclusions from this study.

## 2 Proposed GGAP-RBF Learning Algorithm

In this section, the main ideas behind the GGAP-RBF algorithm are described.

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

transpose of vectors, is given by

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

where  $\alpha_k$  is the weight connecting the  $k$ -th hidden neuron to the output neuron and  $\phi_k(\mathbf{x})$  is the response of the  $k$ -th hidden neuron for an input vector  $\mathbf{x}$ :

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

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

## 2.1 Calculation of Neuron's Significance

This paper 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 learned.

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  $(\mathbf{x}_i, \mathbf{y}_i)$ ,  $i = 1, 2, \dots$ , 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

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

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 :

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

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:

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

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

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

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

$$E_q(k) = \|(E(k, 1), \dots, E(k, n))^T\|_q \quad (7)$$

This can be further written as

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

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 learned the training observations  $(\mathbf{x}_i, \mathbf{y}_i)$ ,  $i = 1, \dots, n$ , 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 in further. Thus, there must

have some simpler and better way to calculate  $E_q(k)$  as stated in equation (8) without the prior knowledge of each specified observations  $(\mathbf{x}_i, \mathbf{y}_i)$ .

Suppose that the observations  $(\mathbf{x}_i, \mathbf{y}_i)$ ,  $i = 1, 2, \dots$ , are drawn from a sampling range  $X$  with a sampling density function  $p(\mathbf{x})$ . Suppose that at an instant 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  $\Delta_j$ ,  $j = 1, \dots, N$ . The size of  $\Delta_j$  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  $\Delta_j$ , where  $\mathbf{x}_j$  is any point chosen in  $\Delta_j$ . From equation (8), we have

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

When the number of input observations  $n$  is large and  $\Delta_j$  is small, we have

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

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

$$E_{\text{sig}}(k) = \lim_{n \rightarrow +\infty} E_q(k) = \|\alpha_k\|_q \left( \int_X \exp\left(-\frac{q \|\mathbf{x} - \mu_k\|^2}{\sigma_k^2}\right) p(\mathbf{x}) d\mathbf{x} \right)^{1/q} \quad (11)$$

If the significance of neuron  $k$  is less than the required accuracy  $e_{\text{min}}$ , 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  $(\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_l)^T$  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  $i$ -th attribute  $x_i$  of observations. Thus, in this case, “significance” (11) can be re-written as:

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

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

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  $p(\mathbf{x})$  functions like uniform, normal, exponential and Rayleigh functions, etc.

#### A. Uniform Sampling Distribution:

When the input samples are uniformly drawn from a range  $X$ , the sampling density function  $p(\mathbf{x})$  is given by  $p(\mathbf{x}) = \frac{1}{S(X)}$ , 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 equation (11) we get,

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

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

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

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 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

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

Thus, the ‘‘significance’’ (11) can be re-written as,

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

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

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

## C. Rayleigh Sampling Distribution:

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

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

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

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

#### 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,

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

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)) is 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 the combination of some of these distributions according to equation (12). For example, as shown in Section 3, the sampling distribution of the real large scale complex application “California Housing” can be simply approximated by a combination of all these four typical distributions.

## 2.2 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:

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

where  $\mathbf{x}_n$  is the latest input received,  $\mu_{nr}$  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  $\epsilon_n$  is a threshold to be selected appropriately.

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

$$\begin{cases} \alpha_{K+1} = e_n \\ \mu_{K+1} = \mathbf{x}_n \\ \sigma_{K+1} = \kappa\|\mathbf{x}_n - \mu_{nr}\| \end{cases} \quad (23)$$

where  $e_n = \mathbf{y}_n - f(\mathbf{x}_n)$  and  $\kappa$  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 equation (23) in equation (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  $\|e_n\|_q \left( \int_X \exp\left(-\frac{q\|\mathbf{x}-\mathbf{x}_n\|^2}{\kappa^2\|\mathbf{x}_n-\mu_{nr}\|^2}\right) p(\mathbf{x}) d\mathbf{x} \right)^{1/q} > e_{\min}$  based on the significance of newly intentionally added neuron.

### 2.3 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

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

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 section 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.

### 2.4 Nearest Neuron for Parameter Adjustment and Pruning

In order to increase the learning speed further it can be shown that, (instead of adjusting the parameters and conducting pruning checking for all neurons after each observation), 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 instant time, only *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  $\phi(x) = \exp(-\frac{x^2}{\sigma^2})$ , its first and second derivatives will approach zero faster when  $x > \sigma$ . Thus, in the gradient vector  $\mathbf{a}_n$  of EKF[11] all

elements except  $\phi_{nr}(\mathbf{x}_n)$ ,  $\phi_{nr}(\mathbf{x}_n)\frac{2\alpha_{nr}}{\sigma_{nr}^2}(\mathbf{x}_n - \mu_{nr})^T$ ,  $\phi_{nr}(\mathbf{x}_n)\frac{2\alpha_{nr}}{\sigma_{nr}^3}\|\mathbf{x}_n - \mu_{nr}\|^2$  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 presented, RANEKF and MRAN needs to calculate Kalman gain vector and error covariance matrix by using  $3K$  gradient vector, where  $K$  is the number of neurons obtained so far. When  $K$  becomes large in some large-scale complex applications, this would result in 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  $3 \times 3$  matrix no matter how many neurons have been obtained.

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  $n$ -th 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 Yingwei, et al[11] for EKF equation details.)

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

**The Proposed GGAP-RBF (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  $n = 1, 2, \dots$ , do

1. **compute** the overall network output:

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

where  $K$  is the number of hidden neurons.

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

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

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

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

**allocate** a new hidden neuron  $K + 1$  with

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

*Else*

**adjust** the network parameters  $\alpha_{nr}$ ,  $\mu_{nr}$ ,  $\sigma_{nr}$  for the nearest neuron only.

**check** the criterion for pruning the adjusted hidden neuron:

$$\text{If } E_{\text{sig}}(nr) = \|\alpha_{nr}\|_q \left( \int_X \exp\left(-\frac{q\|\mathbf{x}-\mu_{nr}\|^2}{\sigma_{nr}^2}\right) p(\mathbf{x}) d\mathbf{x} \right)^{1/q} < e_{\min}$$

remove the  $nr$ -th hidden neuron

reduce the dimensionality of EKF

*Endif*

*Endif*

### 3 Simulation Results

In this section, the performance comparison of GGAP-RBF with three other popular sequential algorithms (RAN, RANEKF and MRAN) and SVR are presented for three different artificial and real problems in the function approximation area. The sampling data for first problem are uniformly drawn and the sampling density function of the second problem, a real large-scale complex problem ‘‘California Housing’’, can be approximated by a combination of four typical distributions discussed in this paper: uniform, normal, exponential and Rayleigh sampling distributions. All studies on GGAP-RBF and MRAN, and some simulations of RAN and RANEKF are carried out in Matlab 6.5 environment running in

an ordinary PC with 256 Megabits RAM, Windows 2000, and a Pentium 4 1.7GHZ CPU. Some simulations of RAN and RANEKF are conducted in MATLAB running in a SUN E250 with one Gigabits RAN and dual 900MHZ CPUs since large network are obtained by RAN and RANEKF for some applications and large memory are required.

For all the simulations except time series applications, 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, RANEKF and MRAN we set  $Q_0 = 0.00001$  and  $P_0 = 0.9$ .

Two learning accuracy measurements are used in all the simulation experiments. Mean Arithmetic Error (MAE) is defined as:

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

and Root Mean Square Error (RMSE) is defined as:

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

### 3.1 Uniform Sampling Distribution: Multiple-Input Multiple-Output Function Approximation

In this example, GGAP-RBF, RAN, RANEKF, and MRAN are used to approximate the following two input-two output sinusoidal function[19]:

$$\begin{aligned} y_1 &= \frac{1}{6}(5 \sin(x_1) + \cos(x_2)), \\ y_2 &= \frac{1}{5}(3 \cos(x_1) + 2 \sin(x_2)), \quad x_1 \in [0, 2\pi], \quad x_2 \in [-2\pi, 2\pi] \end{aligned} \quad (30)$$

A sample set is created by generating 3100 uniformly distributed random values of  $x_1$  and  $x_2$  and calculating the associated values of  $y_1$  and  $y_2$ . Among them, 3000 data and 100

data are used for training and testing, respectively. The chosen approximation accuracy  $e_{\min}$  is 0.0001. After trial-and-error, an appropriate size of sliding window for growing and pruning in MRAN algorithm is chosen as  $M = 50$ . Since training samples are drawn from the range  $X = [0, 2\pi] \times [-2\pi, 2\pi]$ , the sampling range size  $S(X)$  in the GGAP-RBF algorithm can be simply set as  $S(X) = 8\pi^2$ . Seen from Table 1, RANEKF spent 8605.0s CPU time and obtained a network with 400 neurons when learning around 1800 samples only. RAN spent 476.11s CPU time learning 3000 data and obtaining a network with 1222 neurons. MRAN spent 285.04s CPU time learning 3000 data and obtaining a network with 68 neurons. However, GGAP-RBF with 1-norm growing and pruning approach<sup>1</sup> spent only 181.73s CPU time learning 3000 data and obtaining a very small network with 57 neurons only. It is also seen that GGAP-RBF with 2-norm growing and pruning approach spent 74459.0s CPU time learning 3000 data and obtaining a network with 1020 neurons which is smaller than the one obtained by RAN. In fact, during the simulation it is found that it is not able to learn all the training data using RANEKF and RAN in our PC computing environment because of lower memory configuration in our PC. In fact, when an observation presented, RANEKF needs to calculate Kalman gain vector and error covariance matrix by using  $3K$  gradient vector, where  $K$  is the number of neurons obtained so far. For example, when 400 neurons are obtained  $1200 \times 1200$  matrix needs to be calculated. 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  $3 \times 3$  matrix no matter how many neurons have been obtained. Seen from Table 1, GGAP-RBF with 1-norm growing and pruning approach obtains the smallest network in the fastest way than other algorithms and achieves the best

---

<sup>1</sup>In the context of this paper, GGAP-RBF with 1-norm growing and pruning approach ( $q = 1$  in “significance” (11)) is denoted by (1-norm) GGAP-RBF and GGAP-RBF with 2-norm growing and pruning approach ( $q = 2$  in “significance” (11)) is denoted by (2-norm) GGAP-RBF.

generalization performance as well. Similar to the other experimental results, it is found that for both MAE and RMSE learning accuracy measures GGAP-RBF with 1-norm growing and pruning approach always outperforms GGAP-RBF with 2-norm growing and pruning approach. Figures 1 and 2 respectively compared the learning error (for both MAE and RMSE) adjusted with the number of training observations and the spent CPU times for the three learning algorithms (RAN, MRAN and GGAP-RBF). Figure 3(a) compares the neuron updating patterns of the three algorithms, and Figure 3(b) displays more details on the updating patterns for (1-norm) GGAP-RBF and MRAN networks. Since, obviously the RANEKF spent much larger time on training than others, the performance of RANEKF is not displayed in these figures.

### 3.2 Real-World Large-Scale Benchmark Problem with Complex Sampling Distributions: California Housing

California Housing<sup>2</sup> is a dataset obtained from the StatLib repository. There are 20,640 observations predicting the price of house pricing. Each observation consists of 8 continuous inputs and one continuous output. For simplicity, the 8 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 12,640 testing data are randomly generated from the California Housing database. Performance among GGAP-RBF, RAN, RANEKF, MRAN, and SVR are compared in this real large scale complex problem. 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, set  $C = 500$  after several trial-and-error. For MRAN algorithm,

---

<sup>2</sup><http://www.niaad.liacc.up.pt/ltorgo/Regression/calhousing.html>

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  $M = 60$  after several trial-and-error. After analyzing the distributions of the eight input attributes as shown in Figures 4 and 5, it is found that the distribution of the first two input attributes are bi-normal, 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 approximated by a Rayleigh distribution. In fact, after simple estimation, the sampling density functions  $p_i(x)$  ( $i = 1, \dots, 8$ ) of these 8 attributes can simply be approximated as follows, respectively:

$$\begin{aligned}
p_1(x) &= 0.5 \left( \frac{1}{\sqrt{2\pi} \times 0.08} \exp\left(-\frac{x-0.25}{2 \times 0.08^2}\right) + \frac{1}{\sqrt{2\pi} \times 0.10} \exp\left(-\frac{x-0.65}{2 \times 0.10^2}\right) \right) \\
p_2(x) &= 0.5 \left( \frac{1}{\sqrt{2\pi} \times 0.08} \exp\left(-\frac{x-0.15}{2 \times 0.08^2}\right) + \frac{1}{\sqrt{2\pi} \times 0.10} \exp\left(-\frac{x-0.55}{2 \times 0.10^2}\right) \right) \\
p_3(x) &= 1 \\
p_4(x) &= \frac{1}{0.0675} \exp\left(\frac{x}{0.0675}\right) \\
p_5(x) &= \frac{1}{0.0838} \exp\left(\frac{x}{0.0838}\right) \\
p_6(x) &= \frac{1}{0.0501} \exp\left(\frac{x}{0.0501}\right) \\
p_7(x) &= \frac{1}{0.0825} \exp\left(\frac{x}{0.0825}\right) \\
p_8(x) &= \frac{x}{0.1880^2} \exp\left(-\frac{x^2}{2 \times 0.1880^2}\right)
\end{aligned} \tag{31}$$

During this simulation, we assume that the 8 attributes are independent from each. Thus, according to “significance” (12), the significance of neuron  $k$  defined in the GGAP-RBF algorithm

$$E_{\text{sig}}(k) = \|\alpha_k\|_q \prod_{i=1}^8 \left( \int_0^1 \exp\left(-\frac{q(x-\mu_{k,i})^2}{\sigma_k^2}\right) p_i(x) dx \right)^{1/q} \tag{32}$$

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

respectively.

Table 2 shows that GGAP, MRAN, RANEKF, RAN, and SVR achieve the comparable generalization performance. SVR spent 164.8440s (running in C executable environment faster than MATLAB environment<sup>3</sup>) 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 also. RAN spent 4537.8s obtaining a network with 3822 neurons and MRAN spent 2891.5s obtaining a smaller network with 64 neurons. Since RANEKF needs more complex and large matrix computation whenever a new observation presented, it has spent 14181s for learning 2337 observations only. However, GGAP-RBF for 1-norm and 2-norm growing and pruning approaches respectively spent only 115.34s and 191.23s obtaining much smaller networks with 18 and 24 neurons only. Figure 6 shows the learning speed comparison of different learning algorithms (except SVR) for this large scale complex problem and Figure 7 shows that neuron updating progress. Since SVR is not a sequential algorithm, unlike the rest sequential algorithms, the per-observation learning information (learning time and neuron updating progress) cannot be provided.

### 3.3 Application to Time Series Prediction

One possible application of GGAP-RBF is to predict complex time series, a special function approximation problem. As mentioned by Platt[9], the need to time series arises in such real-world problems as detecting arrhythmias in heartbeats. The chaotic Mackey-Glass differential delay equation is recognized as one of the benchmark time series problems,

---

<sup>3</sup>refer to <http://www.mathworks.com/products/compiler/examples/example2.shtml> for compiler execution speed comparison.

which is generated from the following delay differential equation:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \quad (33)$$

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

$$x(t + \Delta t) = \frac{2 - b\Delta t}{2 + \Delta t}x(t) + \frac{a\Delta t}{2 + b\Delta t} \left[ \frac{x(t + \Delta t - \tau)}{1 + x^{10}(t + \Delta t - \tau)} + \frac{x(t - \tau)}{1 + x^{10}(t - \tau)} \right] \quad (34)$$

Same as Yingwei, et al[11], set  $\Delta t = 1$ , the time series is generated under the condition  $x(t - \tau) = 0.3$  for  $0 \leq t \leq \tau$  ( $\tau = 17$  in our test case). The series is predicted with  $v = 50$  sample steps ahead using four past samples:  $s_{n-v}$ ,  $s_{n-v-6}$ ,  $s_{n-v-12}$ , and  $s_{n-v-18}$ . Hence, the  $n$ -th input-output data for the network to learn are

$$\mathbf{x}_n = [s_{n-v}, s_{n-v-6}, s_{n-v-12}, s_{n-v-18}]^T, \quad (35)$$

$$y_n = s_n$$

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

$$z_n = f(\mathbf{x}_n) \quad (36)$$

The  $v$  step-ahead prediction error is

$$e_n = s_n - z_n \quad (37)$$

Similar to Yingwei, et al[11], the parameter values are selected as follows:  $e_{\min} = 0.01$ ,  $\kappa = 0.87$ ,  $\epsilon_{\max} = 0.1$ ,  $\epsilon_{\min} = 0.01$ . For MRAN, the growing threshold and growing/pruning window size are chosen as 0.04 and 90, respectively.

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 3 shows the learning speed, training accuracy,

prediction performance and the obtained neurons. It can be seen that both GGAP and MRAN can get good prediction performance on the long-term prediction. However, (1-norm) GGAP spent 24.326s and obtained a compact network with 13 neurons only while MRAN spent 140.58s and obtained a network with 20 neurons. Figures 8 9 display the neuron updating progress during the online learning phase and the approximated time series curve. As shown in Yingwei, et al[11], MRAN outperforms RAN and RANEKF in this application, since GGAP-RBF shows its superiority over MRAN, it is reasonable to think that the GGAP also outperforms the other algorithms in this application. For the brevity, detailed comparisons with RAN and RANEKF are not presented in this paper.

## 4 Conclusion

It has been found that the networks obtained by RAN and RANEKF become disastrously large in some simulations, indicating that RAN and RANEKF without pruning approaches may not be able to widely cope with real practical applications. 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 training stage would be possibly large. For example, for the California Housing application, one trial of MRAN would be up to 50 minutes. In order to get proper parameters, at least 4 trials were conducted in our experiment and the actual training time we spent was much more than 200 minutes, compared to 115 seconds training time spent for our new proposed algorithm.

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 disastrously large networks in some applications and its applications may be limited, EKF based parameter adjustment method introduced in Kadirkamanathan and Niranjan[10] still plays the key role in both MRAN and GGAP-RBF. The key difference between the parameters adjustments of RANEKF, MRAN and ours is that GGAP-RBF dramatically reduces the computation complexity by only adjusting the parameters of the nearest neuron everytime instead of all neurons without losing learning performance. Actually, this would dramatically increase the learning speed by removing the “curse of dimensionality” of RANEKF and MRAN in most cases. In fact, when an observation presented, RANEKF and MRAN needs to calculate Kalman gain vector and error covariance matrix by using  $3K$  gradient vector, where  $K$  is the number of neurons obtained so far. Since no pruning strategy is adopted in RANEKF, in most cases, especially in some large-scale complex applications, number of neurons obtained by RANEKF would be large although it is much smaller than those obtained by RAN. When the number of neurons obtained by RANEKF becomes large, this would result in large learning time and possibly make ordinary computers incapable of learning because of lower memory configuration and lower computation capability. 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  $3 \times 3$  matrix no matter how many neurons have been obtained.

Using the idea of significance of neurons, which is rigourously defined and **quantified** 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 new algorithm adds and prunes hidden neurons smoothly and produces much more compact

networks than other popular sequential learning algorithms with much better accuracy and generalization performance, and tremendously increased learning speed. The GGAP-RBF algorithm can be used for any arbitrary input sampling distribution and the simulation results are provided for uniform, normal, Rayleigh, and exponential sampling cases.

There are several open problems raised from this study, which need to be investigated in further: 1) From all the simulations done in our research for various artificial and real problems, it is found that for both MAE and RMSE learning accuracy measures GGAP-RBF with 1-norm growing and pruning approach always outperforms GGAP-RBF with 2-norm growing and pruning approach. It may be interesting to investigate in theory why it happens in that way; 2) The convergence and generalization performance seem also interesting to be addressed in theory.

## References

- [1] S. Chen, C. F. N. Cowan, and P. M. Grant, “Orthogonal least squares learning algorithm for radial basis function networks,” *IEEE Trans. Neural Networks*, vol. 2, no. 2, pp. 302–309, 1991.
- [2] S. Chen, E. S. Chng, and K. Alkadhimi, “Regularized orthogonal least squares algorithm for constructing radial basis function networks,” *International Journal of Control*, vol. 64, no. 5, pp. 329–337, 1996.
- [3] E. S. Chng, S. Chen, and B. Mulgrew, “Gradient radial basis function networks for nonlinear and nonstationary time series prediction,” *IEEE Trans. Neural Networks*, vol. 7, no. 1, pp. 190–194, 1996.
- [4] M. J. L. Orr, “Regularization on the selection of radial basis function centers,” *Neural Computation*, vol. 7, pp. 606–623, 1995.
- [5] A. G. Bors and M. Gabbouj, “Minimal topology for a radial basis functions neural network for pattern classification,” *Digital Signal Processing*, vol. 4, pp. 173–188, 1994.
- [6] 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. Petsche, eds.), (MIT Press), pp. 155–161, 1997.
- [7] S. Vijayakumar and S. Wu, “Sequential support vector classifiers and regression,” in *Proceedings of International Conference on Soft Computing (SOCO’99)*, (Genoa, Italy), pp. 610–619, 1999.
- [8] J. Platt, “Sequential Minimal Optimization: A fast algorithm for training support vector machines,” *Microsoft Research Technical Report MSR-TR-98-14*, 1998.

- [9] J. Platt, “A resource-allocating network for function interpolation,” *Neural Computation*, vol. 3, pp. 213–225, 1991.
- [10] V. Kadiramanathan and M. Niranjan, “A function estimation approach to sequential learning with neural networks,” *Neural Computation*, vol. 5, pp. 954–975, 1993.
- [11] L. Yingwei, N. Sundararajan, and P. Saratchandran, “A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks,” *Neural Computation*, vol. 9, pp. 461–478, 1997.
- [12] L. Yingwei, N. Sundararajan, and P. Saratchandran, “Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm,” *IEEE Trans. Neural Networks*, vol. 9, no. 2, pp. 308–318, 1998.
- [13] N. Sundararajan, P. Saratchandran, and L. Yingwei, “Radial basis function neural networks with sequential learning: MRAN and its applications,” River Edge: Singapore; World Scientific: NJ, 1999.
- [14] M. Salmerón, J. Ortega, C. G. Puntonet, and A. Prieto, “Improved ran sequential prediction using orthogonal techniques,” *Neurocomputing*, vol. 41, pp. 153–172, 2001.
- [15] 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.
- [16] J. N. Franklin, “Determinants,” in *Matrix Theory*, pp. 1–25, Prentice Hall, New Jersey, 1968.
- [17] S. Lee and R. M. Kil, “A gaussian potential function network with hierarchically self-organizing learning,” *Neural Networks*, vol. 4, pp. 207–224, 1991.

- [18] B. Todorović and M. Stanković, “Sequential growing and pruning of radial basis function network,” in *Proceedings of International Joint Conference on Neural Networks (IJCNN'2001)*, (Washington DC, USA), pp. 1954–1959, July 15-19, 2001.
- [19] J. O. Moody and P. J. Antsaklis, “The dependence identification neural network construction algorithm,” *IEEE Trans. Neural Networks*, vol. 7, no. 1, pp. 3–15, 1996.
- [20] C.-C. Chang and C.-J. Lin, “LIBSVM – a library for support vector machines,” in <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>, Department of Computer Science and Information Engineering, National Taiwan University, Taiwan, 2003.

## Captions of the tables and figures

Table 1: Performance comparison of different algorithms for multi-input multi-output function approximation

Table 2: Performance comparison of different algorithms for a real large-scale complex application: California Housing

Table 3: Performance comparison of different algorithms for a time series application: Mackey-Glass

Figure 1: Learning accuracy comparison: (a) MAE (b) RMSE.

Figure 2: Learning speed comparison of different learning algorithms

Figure 3: Neuron updating progress: (a) for different algorithms; (b) details for (1-norm) GGAP-RBF and MRAN algorithms only.

Figure 4: A histogram shows the distribution of California Housing training samples: Attributes 1-4.

Figure 5: A histogram shows the distribution of California Housing training samples: Attributes 5-8.

Figure 6: Learning speed comparison of different learning algorithms for California Housing application.

Figure 7: Neuron updating progress: (a) for different algorithms; (b) details for GGAP-RBF and MRAN algorithms only.

Figure 8: Neuron updating progress for Mackey-Glass time series.

Figure 9: Time series prediction: (a) (1-norm) GGAP-RBF; (b) (2-norm) GGAP-RBF.

Algorithms	CPU Time(s)	Training Error		Testing Error		No. of Neurons
		MAE	RMSE	MAE	RMSE	
GGAP (1-norm) <sup>a</sup>	181.73	0.044274	0.08221	0.045673	0.085897	57
GGAP (2-norm) <sup>b</sup>	74459.0	0.04508	0.10859	0.076178	0.14311	1020
MRAN <sup>a</sup>	285.04	0.11984	0.22559	0.1237	0.23171	68
RANEKF <sup>a,c</sup>	8605.0	0.024339	0.067929	0.056812	0.1146	400
RAN <sup>b</sup>	476.11	0.092084	0.19452	0.14326	0.25662	1222

<sup>a</sup> run in Windows 2000 environment. <sup>b</sup> run in Sun Solaris 8 environment. <sup>c</sup> when 1769 samples were learned.

Table 1: Performance comparison of different algorithms for multi-input multi-output function approximation

Algorithms	CPU Time(s)	Training Error		Testing Error		No. of Neurons/SVs
		MAE	RMSE	MAE	RMSE	
GGAP (1-norm) <sup>a</sup>	115.34	0.10468	0.14169	0.1025	0.13861	18
GGAP (2-norm) <sup>a</sup>	191.23	0.10208	0.13905	0.10361	0.14038	24
MRAN <sup>a</sup>	2891.5	0.11452	0.15984	0.11397	0.15859	64
RANEKF <sup>a,d</sup>	14181	0.029104	0.073576	0.11159	0.14952	200
RAN <sup>b</sup>	4537.8	0.057661	0.1009	0.10276	0.15265	3822
SVR <sup>c</sup>	164.8440	-	0.1252	-	0.1282	2429

<sup>a</sup> run in MATLAB and Windows 2000 environment. <sup>b</sup> run in MATLAB and Sun Solaris 8 environment.

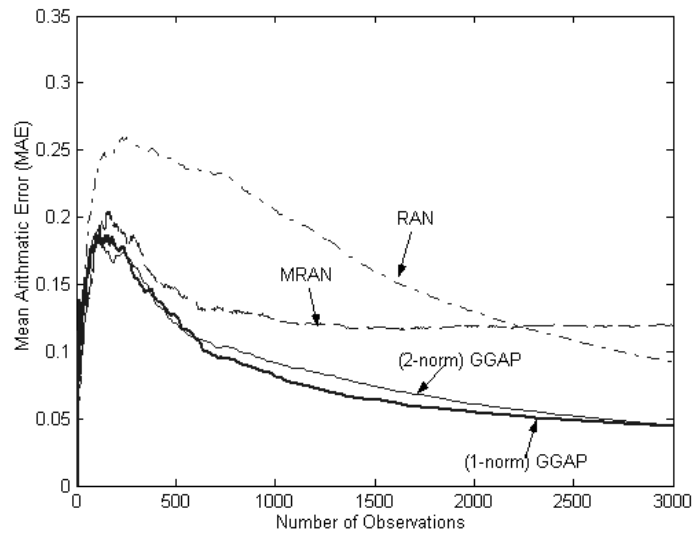
<sup>c</sup> run in C executable and Windows 2000 environment. <sup>d</sup> when 2337 samples were learned.

Table 2: Performance comparison of different algorithms for a real large-scale complex application: California Housing

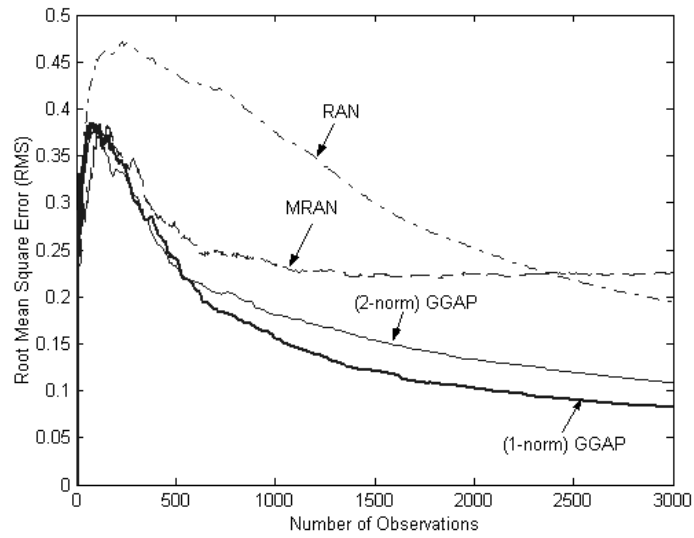
Algorithms	CPU Time(s)	Training Error		Testing Error		No. of Neurons
		MAE	RMSE	MAE	RMSE	
GGAP (1-norm)	24.326	0.034363	0.069952	7.1962e-5	0.036771	13
GGAP (2-norm)	47.079	0.0375	0.074826	1.4656e-4	0.045238	21
MRAN	140.58	0.022219	0.06747	6.6156e-5	0.019686	20

Table 3: Performance comparison of different algorithms for a time series application:

Mackey-Glass



(a)



(b)

Figure 1: Learning accuracy comparison: (a) MAE (b) RMSE.

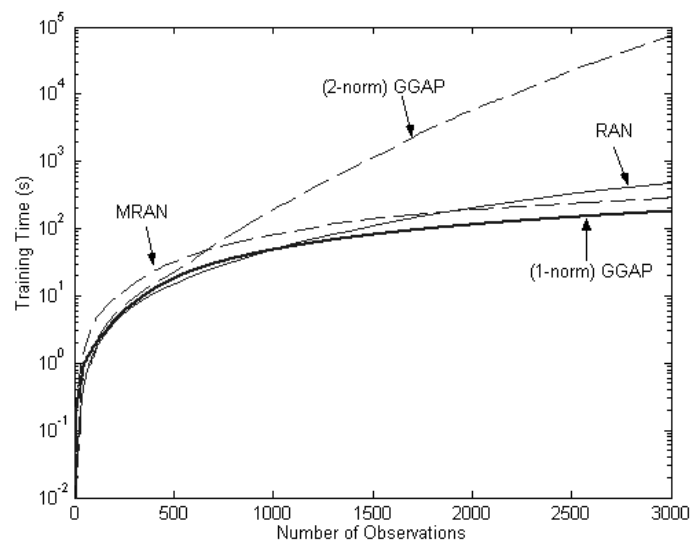
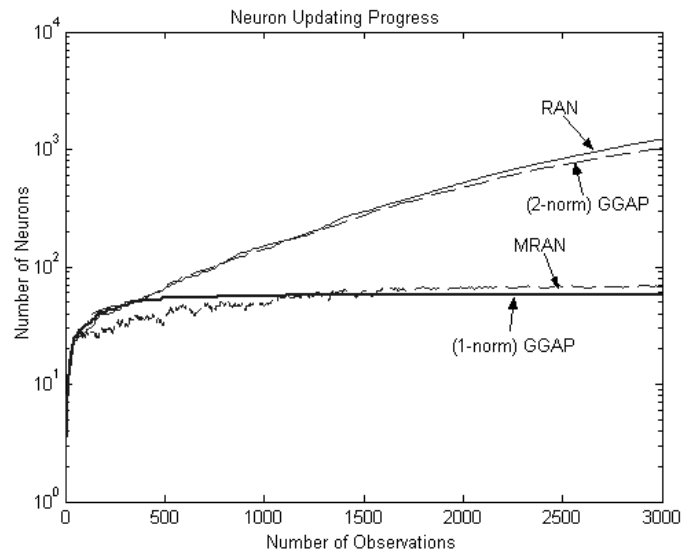
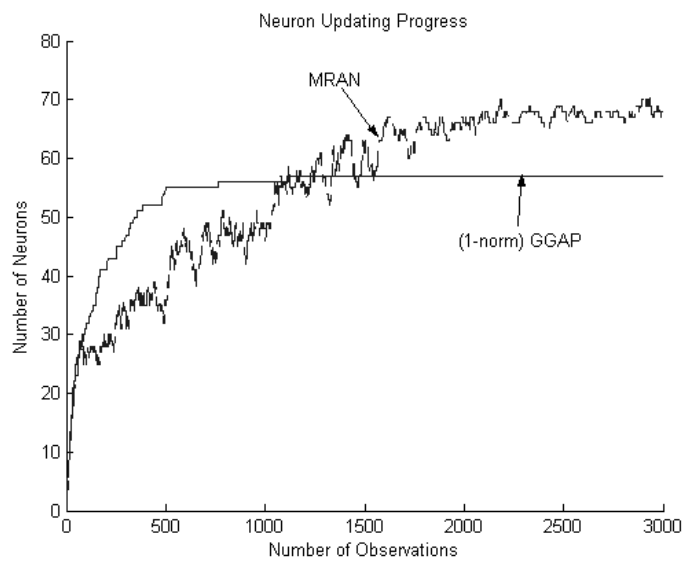


Figure 2: Learning speed comparison of different learning algorithms.



(a)



(b)

Figure 3: Neuron updating progress: (a) for different algorithms; (b) details for (1-norm) GGAP-RBF and MRAN algorithms only.

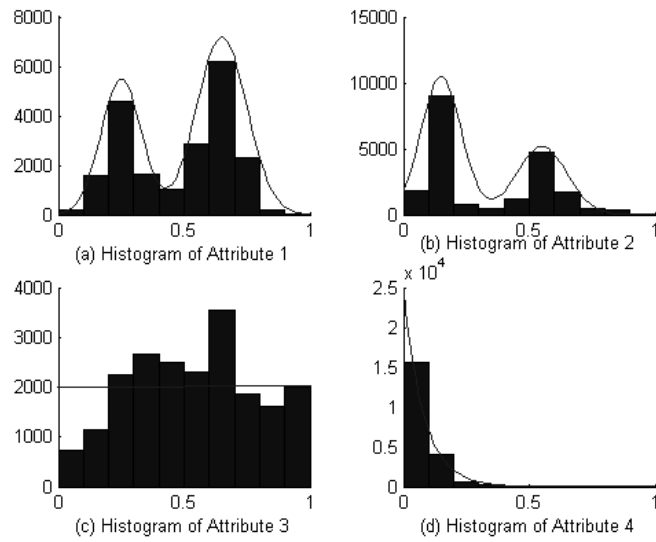


Figure 4: A histogram shows the distribution of California Housing training samples: Attributes 1-4.

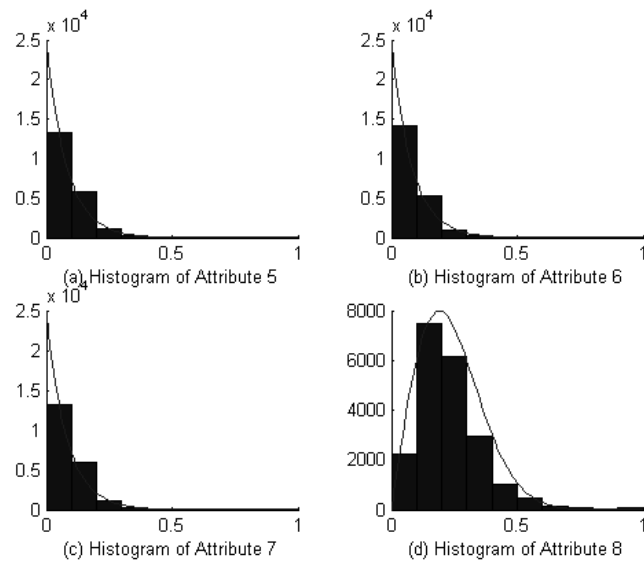


Figure 5: A histogram shows the distribution of California Housing training samples: Attributes 5-8.

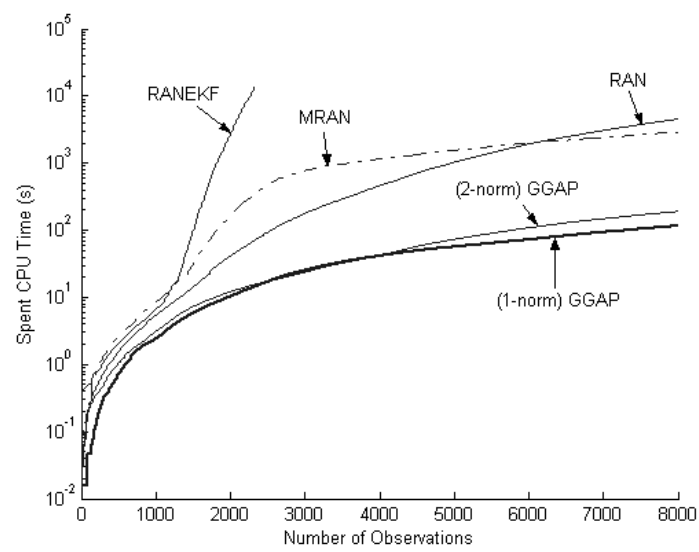
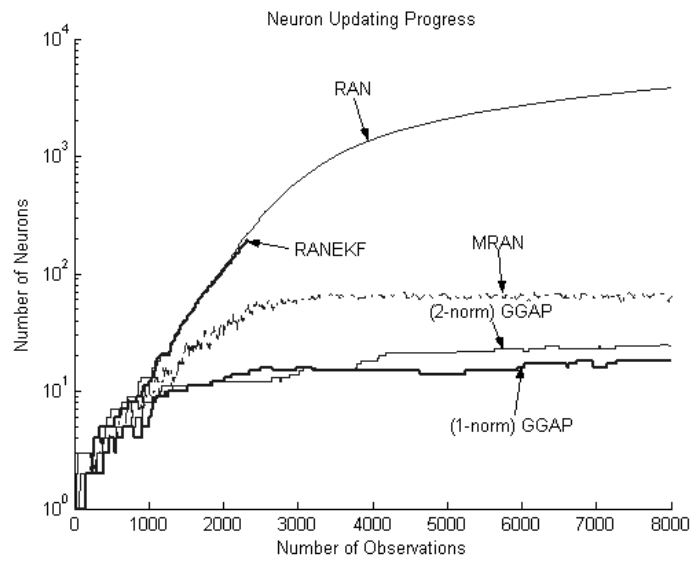
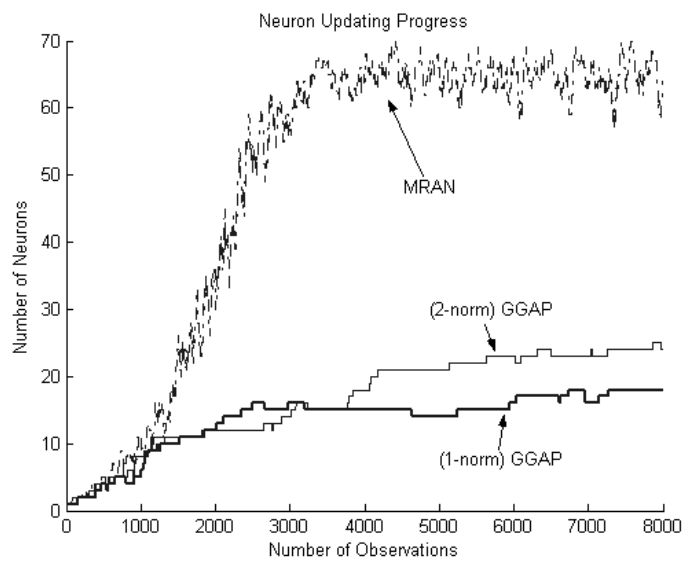


Figure 6: Learning speed comparison of different learning algorithms for California Housing application.



(a)



(b)

Figure 7: Neuron updating progress: (a) for different algorithms; (b) for GGAP-RBF and MRAN algorithms only.

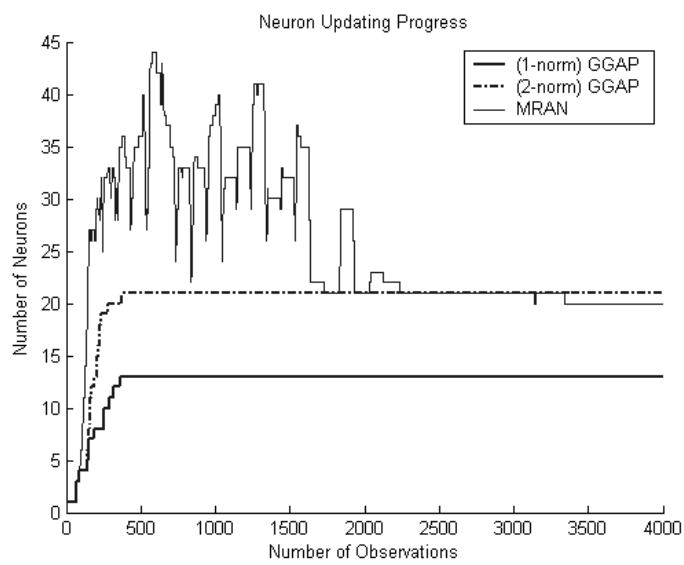
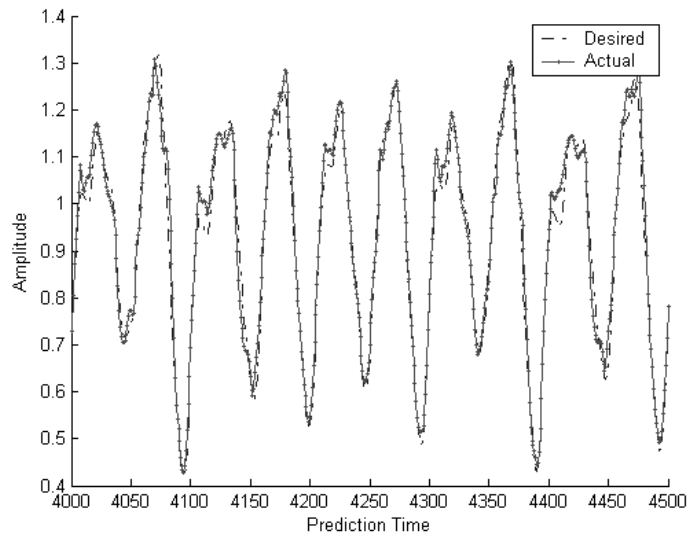
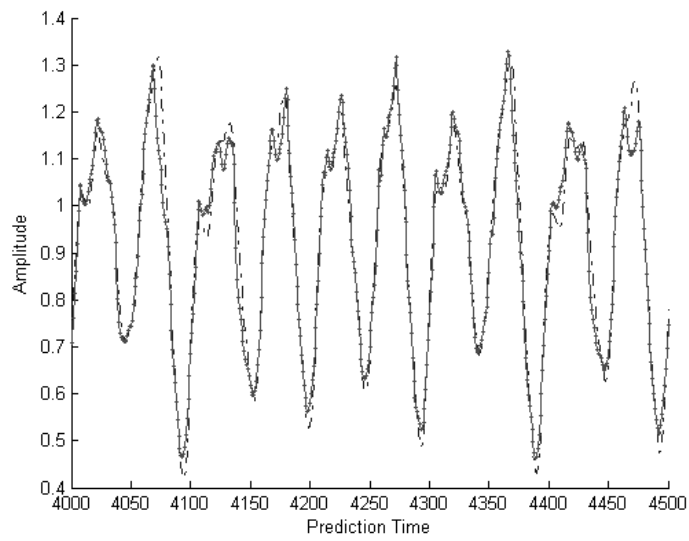


Figure 8: Neuron updating progress for Mackey-Glass time series.



(a)



(b)

Figure 9: Time series prediction: (a) (1-norm) GGAP-RBF; (b) (2-norm) GGAP-RBF.