Growing Radial Basis Neural Networks: Merging Supervised and Unsupervised Learning with Network Growth Techniques

Nicolaos B. Karayiannis, *Member, IEEE,* and Glenn Weiqun Mi

***Abstract*— This paper proposes a framework for constructing and training radial basis function (RBF) neural networks. The proposed growing radial basis function (GRBF) network begins with a small number of prototypes, which determine the locations of radial basis functions. In the process of training, the GRBF network grows by splitting one of the prototypes at each growing cycle. Two splitting criteria are proposed to determine which prototype to split in each growing cycle. The proposed hybrid learning scheme provides a framework for incorporating existing algorithms in the training of GRBF networks. These include unsupervised algorithms for clustering and learning vector quantization, as well as learning algorithms for training single-layer linear neural networks. A supervised learning scheme based on the minimization of the localized class-conditional variance is also proposed and tested. GRBF neural networks are evaluated and tested on a variety of data sets with very satisfactory results.**

***Index Terms*— Class-conditional variance, network growing, radial basis neural network, radial basis function, splitting criterion, stopping criterion, supervised learning, unsupervised learning.**

I. Introduction

RADIAL basis function (RBF) neural networks consist of neurons which are locally tuned. An RBF network can be regarded as a feedforward neural network with a single layer of hidden units, whose responses are the outputs of radial basis functions. The input of each radial basis function of an RBF neural network is the distance between the input vector (activation) and its center (location).

Fig. 1 shows the schematic diagram of an RBF network with n_i inputs, c radial basis functions, and n_o output units. In Fig. 1, $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^{n_i}$ is the input vector and $\boldsymbol{v}_j \in \mathbb{R}^{n_i}$ are prototypes of the input vectors $\boldsymbol{x} \in \mathcal{X}$. Given a set of prototypes $\boldsymbol{v}_j \in \mathbb{R}^{n_i}, 1 \leq j \leq c$, the output of each radial basis function is

$$h_j = h_j(||\boldsymbol{x} - \boldsymbol{v}_j||), \qquad 1 \leq j \leq c \tag{1}$$

where $h_j(\cdot)$ is the *basis function* and $||\cdot||$ is a norm on the input space. The Euclidean norm is frequently used in (1), while the Gaussian function $G(x) = \exp(-x^2/\sigma^2)$ is preferred among possible radial basis functions due to the fact that it is factorizable. In such a case, σ determines the width of the radial basis function. If the RBF network consists of c radial basis functions and the output units are linear, the response of the ith output unit is

$$\hat{y}_i = \boldsymbol{w}_i^T \boldsymbol{h} = \sum_{j=0}^{c} w_{ij} h_j, \qquad 1 \leq i \leq n_o \tag{2}$$

where $\boldsymbol{w}_i = [w_{i0} \;\; w_{i1} \;\; w_{i2} \cdots w_{ic}]^T$ and $\boldsymbol{h} = [1 \;\; h_1 \;\; h_2 \;\; \cdots \;\; h_c]^T$. Thus the RBF network maps the input vector $\boldsymbol{x} \in \mathbb{R}^{n_i}$ into $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{h} \in \mathbb{R}^{n_o}$, where $\hat{\boldsymbol{y}} = [\hat{y}_1 \;\; \hat{y}_2 \;\; \cdots \;\; \hat{y}_{n_o}]^T$, and $\boldsymbol{W}$ is the weight matrix defined as $\boldsymbol{W} = [\boldsymbol{w}_1 \;\; \boldsymbol{w}_2 \;\; \cdots \;\; \boldsymbol{w}_{n_o}]^T$.

An RBF neural network is usually trained to map a vector $\boldsymbol{x}_k \in \mathbb{R}^{n_i}$ into a vector $\boldsymbol{y}_k \in \mathbb{R}^{n_o}$, where the pairs $(\boldsymbol{x}_k, \boldsymbol{y}_k), 1 \leq k \leq M$, form the training set. If this mapping is viewed as a function in the input space $\mathbb{R}^{n_i}$, learning can be seen as a function approximation problem. According to this point of view, learning is equivalent to finding a surface in a multidimensional space that provides the best fit to the training data. Generalization is therefore synonymous with interpolation between the data points along the constrained surface generated by the fitting procedure as the optimum approximation to this mapping.

Broomhead and Lowe [4] were the first to exploit the use of radial basis functions in the design of neural networks and to show how RBF networks model nonlinear relationships and implement interpolation. Micchelli [16] showed that RBF neural networks can produce an interpolating surface which exactly passes through all the pairs of the training set. However, in applications the exact fit is neither useful nor desirable since it may produce anomalous interpolation surfaces. Poggio and Girosi [24] viewed the learning process in an RBF network as an ill-posed problem, in the sense that the information in the training data is not sufficient to reconstruct uniquely the mapping in regions where data are not available. Thus learning is closely related to classical approximation techniques, such as generalized splines and regularization theory. Park and Sandberg [22], [23] proved that RBF networks with one hidden layer are capable of universal approximation. Under certain mild conditions on the radial basis functions, RBF networks are capable of approximating arbitrarily well any function. Similar proofs also exist in the literature for conventional feedforward neural models with sigmoidal nonlinearities [5].

Manuscript received August 11, 1996; revised February 3, 1997 and August 8, 1997.

N. B. Karayiannis is with the Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77204-4793 USA.

G. W. Mi is with CommTech Corporation, Westerville, OH 43081 USA.

Publisher Item Identifier S 1045-9227(97)07968-X.

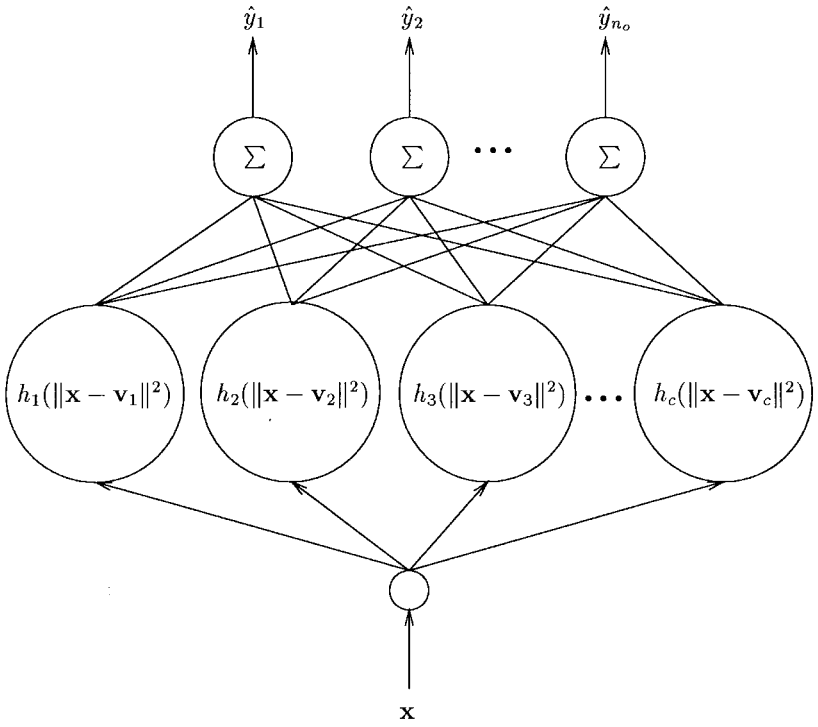

Fig. 1. An RBF neural network.

The performance of an RBF network depends on the number and positions of the radial basis functions, their shape, and the method used for determining the associative weight matrix $\boldsymbol{W}$. The existing learning strategies for RBF neural networks can be classified as follows: 1) RBF networks with a fixed number of radial basis function centers selected randomly from the training data [4]; 2) RBF networks employing unsupervised procedures for selecting a fixed number of radial basis function centers [18]; and 3) RBF networks employing supervised procedures for selecting a fixed number of radial basis function centers [11], [24].

Although some of the RBF neural networks mentioned above are reported to be computationally efficient compared with feedforward neural networks, they have the following important drawback: the number of radial basis functions is determined *a priori*. This leads to similar problems as the determination of the number of hidden units for multilayer neural networks [6], [9], [10], [15]. Several researchers attempted to overcome this problem by determining the number and locations of the radial basis function centers using *constructive* and *pruning* methods.

Fritzke [7] attempted to solve this problem by the *growing cell structure* (GCS), a constructive method that allows RBF neural networks to grow by inserting new prototypes into positions in the feature space where the mapping needs more details. Whitehead and Choate [26] proposed a similar approach which evolves space-filling curves to distribute radial basis functions over the input space. Berthold and Diamond [2] introduced the *dynamic decay adjustment* (DDA) algorithm, which works along with a constructive method to adjust the decay factor (width) of each radial basis function. Musavi *et al*. [19] eliminated the redundant prototypes by merging two prototypes at each adaptation cycle.

This paper presents an alternative approach for constructing and training *growing radial basis function* (GRBF) neural networks. Section II proposes a hybrid scheme for constructing and training GRBF neural networks. Section III presents a variety of unsupervised and supervised algorithms that can be used to determine the locations and widths of Gaussian radial basis functions. Section IV presents learning schemes for updating the synaptic weights of the upper associative network based on recursive least-squares and gradient descent. Section V presents criteria for splitting the prototypes of GRBF neural networks during training and a stopping criterion that can be used to terminate the learning process. Section VI evaluates the performance of a variety of GRBF networks and also compares their performance with that of conventional RBF and feedforward neural networks. Section VII contains concluding remarks.

II. A Hybrid Learning Scheme

Each feature vector presented to an RBF network is mapped into a vector of higher dimension formed by the responses of the radial basis functions. Under certain conditions, such a nonlinear mapping can transform a problem that is linearly nonseparable in the feature space into a linearly separable problem in a space of higher dimension. This transformation depends on the number of radial basis functions, their widths, and their locations in the feature space. The locations of the radial basis functions are determined by the prototypes representing the feature vectors. Because of the role of the

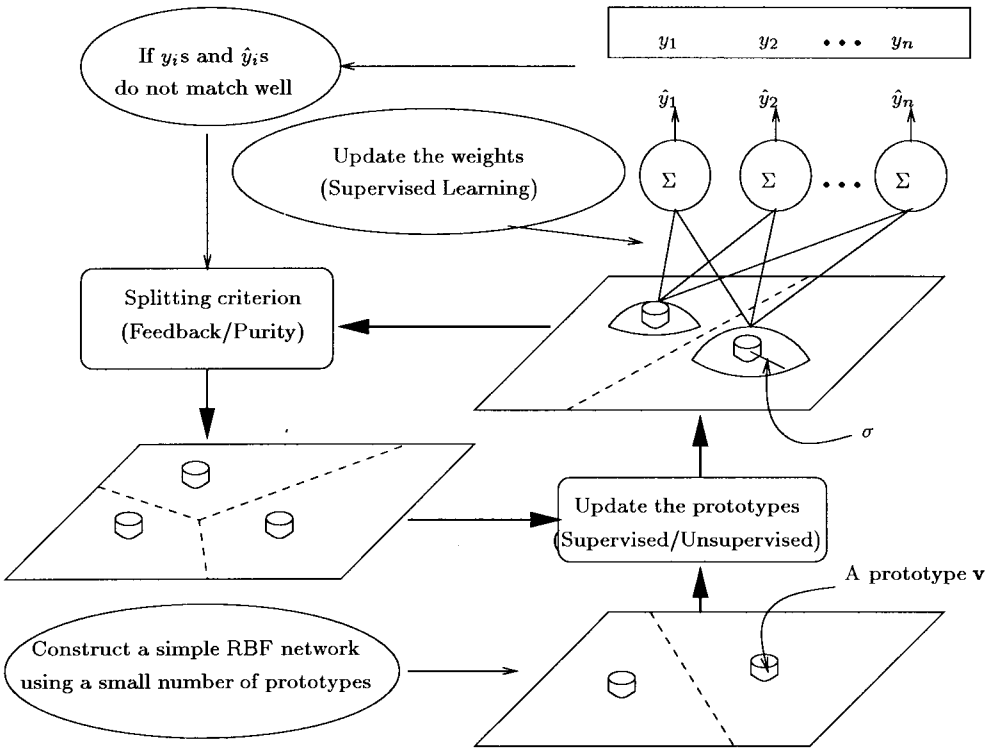

Fig. 2. The proposed construction and learning scheme for GRBF neural networks.

prototypes in the formation of the inputs of the upper network, the partition of the input space produced by the existing set of prototypes is of critical importance in RBF network training. More specifically, the ability of an RBF network to implement the desired mapping and the performance of the trained network are both influenced by the *resolution* of the partition induced upon the feature space by the prototypes. The resolution of the partition is mainly determined by the *a priori* selection of the number of prototypes. A rough partition could make the training of the RBF network impossible. On the other hand, a very fine partition of the feature space could force the network to memorize the examples and, thus, compromise its generalization ability.

The proposed approach allows GRBF networks to grow during training by gradually increasing the number of prototypes that represent the feature vectors in the training set and play the role of the centers of the radial basis functions. The learning process is roughly illustrated in Fig. 2. Learning begins by determining the best two-partition of the feature space. This can be accomplished by employing an unsupervised or a supervised scheme. The two prototypes resulting in this phase define a nontrivial partition of the lowest possible resolution. A supervised learning scheme is subsequently employed to train the upper associative network to give the desired responses when presented with the outputs of the radial basis functions generated by the available prototypes. If the upper associative network is unable to learn the desired mapping, then a finer partition of the feature space is created by splitting one of the existing prototypes as indicated by a splitting criterion. A partition of the feature space of higher resolution is subsequently formed by using the existing set of prototypes as the initial estimate. The prototypes corresponding to the new partition are used to train the upper associative network. This process is repeated until learning is terminated according to a stopping criterion. The stopping criterion is based on the success of the network to implement the desired input–output mapping or the performance of the GRBF network. Splitting and stopping criteria are working together to improve the generalization ability of GRBF networks by preventing the unnecessary splitting of the prototypes.

The proposed learning scheme for GRBF networks can be summarized as follows:

1) Set $c = 2$. Initialize the prototypes $\boldsymbol{v}_j, 1 \leq j \leq c$.
2) Update the prototypes $\boldsymbol{v}_j, 1 \leq j \leq c$.
3) Update the widths of the radial basis functions.
4) Initialize the weights of the upper network.
5) Update the weights of the upper network.
6) If a stopping criterion is satisfied; then stop; else:
7) Split one of the existing prototypes as indicated by a splitting criterion.
8) Set $c = c + 1$ and go to Step 2).

A GRBF network is constructed and trained in a sequence of *growing cycles*. In the above learning scheme, a complete growing cycle is described by Steps 2)–8). If a GRBF network contains c radial basis functions, it has gone through $c - 2$ growing cycles. The proposed construction/learning strategy is compatible with any supervised or unsupervised technique that can be used to map the feature vectors to the prototypes. It is also compatible with a variety of supervised learning schemes

that can be used to train the upper associative network that maps the responses of the radial basis functions to the output units. Thus the proposed learning scheme can be used as a framework for constructing and training a great variety of GRBF neural networks.

III. Learning the Locations and Widths of Radial Basis Functions

This section presents a variety of unsupervised and supervised methods that can be used to determine the locations and widths of radial basis functions.

A. Learning the Locations of Radial Basis Functions: Unsupervised Methods

Unsupervised methods used to determine the locations of the RBF's as the prototypes of the training vectors essentially involve clustering or learning vector quantization (LVQ). The development of clustering algorithms is frequently based on alternating optimization [3]. In contrast, LVQ algorithms map the feature vectors to the prototypes by minimizing a loss function using gradient descent [12], [13].

Clustering is based on the assignment of the feature vectors $\boldsymbol{x}_k \in \mathbb{R}^{n_i}, 1 \le k \le M$, into c clusters, which are represented by the prototypes $\boldsymbol{v}_j \in \mathbb{R}^{n_i}, 1 \le j \le c$. The certainty of the assignment of the feature vector $\boldsymbol{x}_k \in \mathcal{X}$ to the jth cluster is measured by the membership function $u_{kj} = u_j(\boldsymbol{x}_k)$.

1) Hard c-Means Algorithm: The *hard c-means* (HCM) algorithm assigns each of the training vectors to the cluster whose prototype is its closest neighbor. The new prototypes are evaluated from the results of the new partition as the centroids of the training vectors assigned to the corresponding clusters. The HCM algorithm can be summarized as follows [3].

1) Select c and ϵ; set $\nu = 0$; generate the initial set of prototypes: $\mathcal{V}_0 = \{\boldsymbol{v}_{1,0}, \boldsymbol{v}_{2,0}, \cdots, \boldsymbol{v}_{c,0}\}$.
2) Set $\nu = \nu + 1$ (increment iteration counter).
3) Calculate

- $u_{kj,\nu} = \begin{cases} 1 & \text{if } ||\boldsymbol{x}_k - \boldsymbol{v}_{j,\nu}||^2 < ||\boldsymbol{x}_k - \boldsymbol{v}_{\ell,\nu}||^2, \\ & \forall \ell \ne j, \\ 0 & \text{otherwise} \end{cases} \quad \forall k, j.$
- $\boldsymbol{v}_{j,\nu} = \left(\sum_{k=1}^{M} u_{kj,\nu} \boldsymbol{x}_k \right) \Big/ \left(\sum_{k=1}^{M} u_{kj,\nu} \right), \quad \forall j.$

4) Calculate $E_\nu = \Sigma_{j=1}^{c} ||\boldsymbol{v}_{j,\nu} - \boldsymbol{v}_{j,\nu-1}||^2$.
5) If $E_\nu > \epsilon$, then go to Step 2).

Given an initial set of prototypes, the HCM algorithm finds the nearest local minimum in the space of all possible c-partitions. A shortcoming of the algorithm is the creation of degraded clusters, that is, clusters with no input vectors assigned to them. The enhanced version of the HCM algorithm used in this paper eliminates the degraded clusters by assigning to each of them a vector randomly selected from the input vectors.

2) Fuzzy c-Means Algorithm: The *fuzzy c-means* (FCM) algorithm considers each cluster as a fuzzy set, while each feature vector may be assigned to multiple clusters with some degree of certainty measured by the membership function taking values from the interval [0, 1]. The FCM algorithm can be summarized as follows [3].

1) Select c, ϵ, and m; set $\nu = 0$; generate the initial set of prototypes: $\mathcal{V}_0 = \{\boldsymbol{v}_{1,0}, \boldsymbol{v}_{2,0}, \cdots, \boldsymbol{v}_{c,0}\}$.
2) Set $\nu = \nu + 1$ (increment iteration counter).
3) Calculate

- $u_{kj,\nu} = \left(\sum_{\ell=1}^{c} \left(\frac{||\boldsymbol{x}_k - \boldsymbol{v}_{j,\nu}||^2}{||\boldsymbol{x}_k - \boldsymbol{v}_{\ell,\nu}||^2} \right)^{1/(m-1)} \right)^{-1}, \quad \forall k, j.$
- $\boldsymbol{v}_{j,\nu} = \left(\sum_{k=1}^{M} (u_{kj,\nu})^m \boldsymbol{x}_k \right) \Big/ \left(\sum_{k=1}^{M} (u_{kj,\nu})^m \right), \quad \forall j.$

4) Calculate $E_\nu = \Sigma_{j=1}^{c} ||\boldsymbol{v}_{j,\nu} - \boldsymbol{v}_{j,\nu-1}||^2$.
5) If $E_\nu > \epsilon$, then go to Step 2).

The parameter $m \in (1, \infty)$ determines the fuzziness of the partition produced by the FCM algorithm. If $m \to 1_+$, then the resulting partition approaches asymptotically a hard or crisp partition. The algorithm produces a maximally fuzzy partition if $m \to \infty$.

3) Fuzzy Algorithms for LVQ: Consider the set of samples $\boldsymbol{x} \in \mathbb{R}^{n_i}$ and let $f(\boldsymbol{x})$ be the probability density function of $\boldsymbol{x} \in \mathbb{R}^{n_i}$. LVQ is frequently based on the minimization of the functional [21]

$$L(\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_c) = \int \int \cdots \int_{\mathbb{R}^{n_i}} \sum_{r=1}^{c} u_r ||\boldsymbol{x} - \boldsymbol{v}_r||^2 f(\boldsymbol{x})\, d\boldsymbol{x} \tag{3}$$

which represents the expectation of the *loss function*

$$L_{\boldsymbol{x}}(\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_c) = \sum_{r=1}^{c} u_r ||\boldsymbol{x} - \boldsymbol{v}_r||^2. \tag{4}$$

In the above definitions, $u_r = u_r(\boldsymbol{x}), 1 \le r \le c$, are membership functions which regulate the competition between the prototypes $\boldsymbol{v}_r, 1 \le r \le c$, for the input $\boldsymbol{x}$. The specific form of the membership functions determines the strength of attraction between each input and the prototypes during the learning process [13]. Assuming that $\boldsymbol{v}_i$ is the winning prototype corresponding to the input vector $\boldsymbol{x}$, that is, the closest prototype to $\boldsymbol{x}$ in the Euclidean distance sense, the membership $u_r, 1 \le r \le c$, can be of the form

$$u_r(\boldsymbol{x}, \boldsymbol{v}_i) = u_{ir}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } r = i \\ u(||\boldsymbol{x} - \boldsymbol{v}_i||^2 / ||\boldsymbol{x} - \boldsymbol{v}_r||^2) & \text{if } r \ne i. \end{cases} \tag{5}$$

In such a case, the loss function measures the locally weighted error of each input vector with respect to the winning prototype [21].

If the probability density function is not known and u_{ir} is an admissible membership function, a variety of LVQ algorithms can be developed by minimizing the loss function (4) using gradient descent [13]. If $\boldsymbol{x}$ is the input vector, the winning prototype $\boldsymbol{v}_i$ can be updated by

$$\Delta \boldsymbol{v}_i = \eta \left(1 + \sum_{r \ne i}^{c} w_{ir} \right) (\boldsymbol{x} - \boldsymbol{v}_i) \tag{6}$$

where $w_{ir} = w(||\boldsymbol{x}-\boldsymbol{v}_i||^2/||\boldsymbol{x}-\boldsymbol{v}_r||^2)$, with $w(z) = u'(z)$. Each nonwinning prototype $\boldsymbol{v}_j \neq \boldsymbol{v}_i$ can be updated by

$$\Delta \boldsymbol{v}_j = \eta m_{ij}(\boldsymbol{x}-\boldsymbol{v}_j) \tag{7}$$

where $n_{ij} = n(||\boldsymbol{x}-\boldsymbol{v}_i||^2/||\boldsymbol{x}-\boldsymbol{v}_j||^2)$, with $n(z) = u(z) - zw(z)$.

The update of the prototypes depends on the learning rate $\eta \in (0,1)$, which is a monotonically decreasing function of the number of iterations ν. The learning rate can be a linear function of ν defined as $\eta = \eta(\nu) = \eta_0(1-\nu/N)$, where η_0 is its initial value and N is the total number of iterations predetermined for the learning process.

The above formulation provided the basis for the development of the FALVQ 1, FALVQ 2, and FALVQ 3 families of algorithms [12], [13]. The FALVQ 1 algorithm used in this paper corresponds to $u(z) = z(1+z)^{-1}$. If $\boldsymbol{x}$ is the input vector, then the winning prototype $\boldsymbol{v}_i$ is updated by (6) with $w_{ir} = w(||\boldsymbol{x}-\boldsymbol{v}_i||^2/||\boldsymbol{x}-\boldsymbol{v}_r||^2)$ and $w(z) = (1+z)^{-2}$. The nonwinning prototypes $\boldsymbol{v}_j \neq \boldsymbol{v}_i$ can be updated by (7) with $n_{ij} = n(||\boldsymbol{x}-\boldsymbol{v}_i||^2/||\boldsymbol{x}-\boldsymbol{v}_j||^2)$ and $n(z) = z^2(1+z)^{-2}$.

The algorithms described above can be summarized as follows.

1) Select c; fix η_0, N; set $\nu = 0$; generate an initial set of prototypes $\mathcal{V}_0 = \{\boldsymbol{v}_{1,0}, \boldsymbol{v}_{2,0}\cdots, \boldsymbol{v}_{c,0}\}$.
2) Calculate $\eta = \eta_0(1-\nu/N)$.
3) Set $\nu = \nu + 1$ (increment iteration counter).
4) For all $k = 1, 2, \cdots, M$:
   - set $\boldsymbol{x} = \boldsymbol{x}_k$.
   - find i such that $||\boldsymbol{x} - \boldsymbol{v}_{i,\nu-1}||^2 < ||\boldsymbol{x} - \boldsymbol{v}_{j,\nu-1}||^2, \forall j \neq i$.
   - calculate $u_{ir,\nu} = u(||\boldsymbol{x} - \boldsymbol{v}_{i,\nu-1}||^2/||\boldsymbol{x} - \boldsymbol{v}_{r,\nu-1}||^2), \forall r \neq i$.
   - calculate $w_{ir,\nu} = u'(||\boldsymbol{x} - \boldsymbol{v}_{i,\nu-1}||^2/||\boldsymbol{x} - \boldsymbol{v}_{r,\nu-1}||^2), \forall r \neq i$.
   - calculate $n_{ir,\nu} = u_{ir,\nu} - (||\boldsymbol{x} - \boldsymbol{v}_{i,\nu-1}||^2/||\boldsymbol{x} - \boldsymbol{v}_{r,\nu-1}||^2) w_{ir,\nu}, \forall r \neq i$.
   - update $\boldsymbol{v}_i$ by $\boldsymbol{v}_{i,\nu} = \boldsymbol{v}_{i,\nu-1} + \eta(1+\Sigma_{r\neq i}^{c} w_{ir,\nu})(\boldsymbol{x} - \boldsymbol{v}_{i,\nu-1})$.
   - update $\boldsymbol{v}_j \neq \boldsymbol{v}_i$ by $\boldsymbol{v}_{j,\nu} = \boldsymbol{v}_{j,\nu-1} + \eta m_{ij,\nu}(\boldsymbol{x} - \boldsymbol{v}_{j,\nu-1})$.
5) If $\nu < N$, then go to Step 2).

B. Learning the Locations of Radial Basis Functions: The Class-Conditional Variance Method

Consider an RBF network trained to classify the feature vectors $\boldsymbol{x}_k \in \mathcal{X}$ belonging to n_o distinct classes $\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_{n_o}$. In such a case, the desired outputs are usually formed as $y_{i,k} = 1$ if $\boldsymbol{x}_k \in \mathcal{C}_i$ and $y_{i,k} = 0$ if $\boldsymbol{x}_k \notin \mathcal{C}_i$. The labels of the feature vectors play no role in the partition of the feature space if the prototypes are formed by unsupervised methods. Thus unsupervised methods may form clusters of feature vectors that are closely spaced in the feature space but belong to different classes. Such a cluster formation may lead to repeated splitting of prototypes with a negative effect on the generalization ability of the resulting GRBF network.

The information available about the feature space can be directly used for creating a *consistent representation* of the feature space by the prototypes. A set of prototypes constitute a consistent representation of the feature space if the responses of the radial basis functions centered on these prototypes to feature vectors from the same class are as similar as possible. Thus, the consistency of the representation of the feature space by the prototypes can be improved by updating the prototypes so as to minimize the *class-conditional variance*, which measures the similarity of the response of each radial basis function to feature vectors from the same class.

The output variance of the jth radial basis function for the ℓth class $\mathcal{C}_\ell$ is given by

$$S_{j,\ell}^2 = \sum_{k:\boldsymbol{x}_k \in \mathcal{C}_\ell} (\langle h_{j,\mathcal{C}_\ell}\rangle - h_{j,k})^2 \tag{8}$$

where $h_{j,k} = \exp(-||\boldsymbol{x}_k - \boldsymbol{v}_j||^2/\sigma_j^2)$

$$\langle h_{j,\mathcal{C}_\ell}\rangle = \frac{1}{|\mathcal{C}_\ell|} \sum_{k:\boldsymbol{x}_k \in \mathcal{C}_\ell} h_{j,k} \tag{9}$$

and $|\mathcal{C}_\ell|$ denotes the cardinality of $\mathcal{C}_\ell$.

The variance defined in (8) is an effective measure only if the feature vectors belonging to a certain class are represented by a single prototype. However, the feature vectors from the same class may be represented by a number of prototypes if there exists overlapping between the classes or the distribution is nonconvex. Since the variance (8) is defined in terms of all input vectors belonging to class $\mathcal{C}_\ell$, an input vector $\boldsymbol{x}_k \in \mathcal{C}_\ell$ affects all prototypes that represent feature vectors from this class.

A "localized" measure can be obtained by defining the variance for the jth radial basis function in terms of the input vectors selected using as criterion the response of the radial basis functions located at the existing prototypes. According to the proposed scheme, an input vector $\boldsymbol{x}_k \in \mathcal{X}$ is included in the calculation of the variance of the jth radial basis function only if the corresponding response $h_{j,k}$ is the largest in magnitude among the responses of all radial basis functions. Define the set $\mathcal{P}_j = \{\boldsymbol{x}_k \in \mathcal{X} | h_{j,k} > h_{q,k}, \forall q \neq j\}$. The *localized class-conditional variance* of the output of the jth radial basis function for the ℓth class $\mathcal{C}_\ell$ is given by

$$S_{j,\ell}^2 = \sum_{k:\boldsymbol{x}_k \in \mathcal{C}_\ell \cap \mathcal{P}_j} (\langle h_{j,\mathcal{C}_\ell \cap \mathcal{P}_j}\rangle - h_{j,k})^2 \tag{10}$$

where $h_{j,k} = \exp(-||\boldsymbol{x}_k - \boldsymbol{v}_j||^2/\sigma_j^2)$

$$\langle h_{j,\mathcal{C}_\ell \cap \mathcal{P}_j}\rangle = \frac{1}{|\mathcal{C}_\ell \cap \mathcal{P}_j|} \sum_{k:\boldsymbol{x}_k \in \mathcal{C}_\ell \cap \mathcal{P}_j} h_{j,k} \tag{11}$$

and $|\mathcal{C}_\ell \cap \mathcal{P}_j|$ denotes the cardinality of $\mathcal{C}_\ell \cap \mathcal{P}_j$. If the input vectors $\boldsymbol{x}_k \in \mathcal{X}$ belong to n_o distinct classes $\mathcal{C}_1, \mathcal{C}_2, \cdots \mathcal{C}_{n_o}$, the prototypes can be updated by minimizing the objective function G formed by summing $S_{j,\ell}^2$ over all the classes and all the radial basis functions, i.e.,

$$G = \tfrac{1}{2} \sum_{\ell=1}^{n_o} \sum_{j=1}^{c} S_{j,\ell}^2. \tag{12}$$

The update equation for each prototype $\boldsymbol{v}_p$ is derived in the Appendix using gradient descent as

$$\begin{aligned}\Delta \boldsymbol{v}_p &= -\eta \frac{\partial G}{\partial \boldsymbol{v}_p} \\ &= -2\eta \sum_{\ell=1}^{n_o} \sum_{k:\boldsymbol{x}_k \in \mathcal{C}_\ell \cap \mathcal{P}_p} (\langle h_{p,\mathcal{C}_\ell \cap \mathcal{P}_p} \rangle - h_{p,k}) \\ &\quad \cdot (\langle \boldsymbol{e}_{p,\mathcal{C}_\ell \cap \mathcal{P}_p} \rangle - \boldsymbol{e}_{p,k}) \end{aligned} \tag{13}$$

where η is the learning rate

$$\boldsymbol{e}_{p,k} = \frac{h_{p,k}}{\sigma_p^2}(\boldsymbol{x}_k - \boldsymbol{v}_p) \tag{14}$$

and

$$\langle \boldsymbol{e}_{p,\mathcal{C}_\ell \cap \mathcal{P}_p} \rangle = \frac{1}{|\mathcal{C}_\ell \cap \mathcal{P}_p|} \sum_{k:\boldsymbol{x}_k \in \mathcal{C}_\ell \cap \mathcal{P}_p} \boldsymbol{e}_{p,k}. \tag{15}$$

C. Determining the Widths of Radial Basis Functions

The width of each Gaussian in the hidden layer is frequently determined by heuristics. The objective of these schemes is to achieve a certain amount of overlapping between the responses of neighboring units so that the radial basis functions form a smooth and contiguous interpolation over the regions of the input space that they represent.

1) Heuristics: According to the "P nearest-neighbor" heuristic [18], the width of a radial basis function is determined as the average Euclidean distance of its P nearest neighboring functions. The "P nearest-neighbor" heuristic becomes the "nearest neighbor" heuristic when $P = 1$. For a set of prototypes $\mathcal{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_c\}$, the width of the radial basis function centered on the pth prototype is determined as $\sigma_p = \min_{\forall j \neq p} \{||\boldsymbol{v}_p - \boldsymbol{v}_j||\}$.

The "average of close input vectors" heuristic determines the width of a Gaussian radial basis function using the prototypes and the input vectors. Let $\mathcal{I}_j$ be the set of input vectors represented by the prototype $\boldsymbol{v}_j$, that is

$$\mathcal{I}_j = \{\boldsymbol{x}_k \in \mathcal{X} \,|\, ||\boldsymbol{x}_k - \boldsymbol{v}_j||^2 < ||\boldsymbol{x}_k - \boldsymbol{v}_q||^2, \quad \forall q \neq j\}. \tag{16}$$

The width σ_j of the Gaussian function centered on the prototype $\boldsymbol{v}_j$ is calculated according to this scheme as

$$\sigma_j = \frac{1}{|\mathcal{I}_j|} \sum_{k:\boldsymbol{x}_k \in \mathcal{I}_j} ||\boldsymbol{x}_k - \boldsymbol{v}_j||. \tag{17}$$

2) The Class-Conditional Variance Method: The localized class-conditional variance defined above can also be used for updating the widths of the radial basis functions during training. Such an approach is expected to improve the representation of the training set by the receptive fields around the radial basis functions.

The update equation for the width σ_p of the pth radial basis function is derived in the Appendix using gradient descent as

$$\begin{aligned}\Delta \sigma_p &= -\eta \frac{\partial G}{\partial \sigma_p} \\ &= -2\eta \sum_{\ell=1}^{n_o} \sum_{k:\boldsymbol{x}_k \in \mathcal{C}_\ell \cap \mathcal{P}_p} (\langle h_{p,\mathcal{C}_\ell \cap \mathcal{P}_p} \rangle - h_{p,k}) \\ &\quad \cdot (\langle \tilde{h}_{p,\mathcal{C}_\ell \cap \mathcal{P}_p} \rangle - \tilde{h}_{p,k}) \end{aligned} \tag{18}$$

where η is the learning rate

$$\tilde{h}_{p,k} = \frac{||\boldsymbol{x}_k - \boldsymbol{v}_p||^2}{\sigma_p^3} h_{p,k} \tag{19}$$

and

$$\langle \tilde{h}_{p,\mathcal{C}_\ell \cap \mathcal{P}_p} \rangle = \frac{1}{|\mathcal{C}_\ell \cap \mathcal{P}_p|} \sum_{k:\boldsymbol{x}_k \in \mathcal{C}_\ell \cap \mathcal{P}_p} \tilde{h}_{p,k}. \tag{20}$$

In applications, the widths of the radial basis functions can be initialized first by the "P nearest-neighbor" heuristic and then updated using the above equation.

IV. Learning the Input–Output Mapping

The upper network can be trained by minimizing the error

$$E = \tfrac{1}{2} \sum_{k=1}^{M} \sum_{j=1}^{n_o} (y_{i,k} - \hat{y}_{i,k})^2 \tag{21}$$

where $y_{i,k}$ is the desired response of the ith output unit and $\hat{y}_{i,k}$ is the actual response of the ith output unit to $\boldsymbol{h}_k$. If the output units are linear, then

$$\hat{y}_{i,k} = \boldsymbol{w}_i^T \boldsymbol{h}_k = \boldsymbol{h}_k^T \boldsymbol{w}_i = \sum_{j=0}^{c} w_{ij} h_{j,k} \tag{22}$$

where $\boldsymbol{h}_k = [h_{0,k} \quad h_{1,k} \quad \cdots \quad h_{c,k}]^T, h_{0,k} = 1$ and $h_{j,k} = \exp(-||\boldsymbol{x}_k - \boldsymbol{v}_j||^2/\sigma_j^2), 1 \leq j \leq c$. In this case, the solution of this minimization problem is given by ([14], [15])

$$\frac{\partial E}{\partial \boldsymbol{w}_i} = -\sum_{k=1}^{M} \boldsymbol{h}_k (y_{i,k} - \boldsymbol{h}_k^T \boldsymbol{w}_i) = 0. \tag{23}$$

If $\boldsymbol{H}(\boldsymbol{Y})$ is the matrix with column vectors $\boldsymbol{h}_k(\boldsymbol{y}_k), k = 1, 2, \cdots, M$, then (23) results in the normal equation $\boldsymbol{W}\boldsymbol{H}\boldsymbol{H}^T = \boldsymbol{Y}\boldsymbol{H}^T$, where $\boldsymbol{W}$ is the weight matrix. The optimal solution of the normal equation, in the least-squares sense, is given by ([14], [15])

$$\boldsymbol{W} = \boldsymbol{Y}\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{H}^T)^+ \tag{24}$$

where $\boldsymbol{A}^+$ denotes the generalized inverse of any rectangular matrix $\boldsymbol{A}$.

A. Recursive Least-Squares

The evaluation of the weight matrix requires a flexible algorithm that is not based on matrix inversion. This can be accomplished by the ELEANNE 2, a recursive least-squares algorithm proposed for solving the normal equation. This algorithm can be summarized as follows [14], [15].

1) Select η and ϵ; initialize $\boldsymbol{W}$.
2) Set $E = 0$ and $\boldsymbol{P} = (1/\eta)\boldsymbol{I}$.
3) For all $k = 1, 2, \cdots, M$:
   - $\boldsymbol{h} = \boldsymbol{h}_k$.
   - $\boldsymbol{y} = \boldsymbol{y}_k$.
   - $\delta = (1 + \boldsymbol{h}^T \boldsymbol{P} \boldsymbol{h})^{-1}$.
   - $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{W}\boldsymbol{h}$.
   - $\boldsymbol{W} \leftarrow \boldsymbol{W} + \delta \boldsymbol{e} \boldsymbol{h}^T \boldsymbol{P}$.
   - $\boldsymbol{P} \leftarrow \boldsymbol{P} - \delta \boldsymbol{P} \boldsymbol{h} \boldsymbol{h}^T \boldsymbol{P}$.

- $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{W}\boldsymbol{h}$.
- $E \leftarrow E + \frac{1}{2}||\boldsymbol{e}||^2$.

4) if: $E > \epsilon$; then: go to Step 2).

The positive parameter η is used to initialize the recursive evaluation of $\boldsymbol{P} = (\boldsymbol{H}\boldsymbol{H}^T)^+$. If η approaches zero, then the algorithm computes the optimal least-squares estimate for the weight matrix, given by (24), in a single adaptation cycle. The stability of the recursive evaluation of $\boldsymbol{P}$ can be guaranteed in practical situations by selecting $\eta \geq 0.1$ and employing additional adaptation cycles [14], [15].

B. Gradient Descent

The error E can also be minimized by an iterative algorithm based on gradient descent. According to this approach, the weight vectors $\boldsymbol{w}_i, 1 \leq i \leq n_o$, of the upper associative network can be updated by

$$\begin{aligned}\Delta \boldsymbol{w}_i &= -\alpha \frac{\partial E}{\partial \boldsymbol{w}_i} \\ &= \alpha \sum_{k=1}^{M} \boldsymbol{h}_k (y_{ik} - \hat{y}_{ik}) \end{aligned} \tag{25}$$

where α is the learning rate. This algorithm can be summarized as follows.

1) Select α and ϵ; initialize $\boldsymbol{W}$; compute $\hat{y}_{i,k} = \boldsymbol{w}_i^T \boldsymbol{h}_k, \forall i, k$.
2) For all $i = 1, 2, \cdots, n_o$:
   a) $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i + \alpha \Sigma_{k=1}^{M} \boldsymbol{h}_k (y_{i,k} - \hat{y}_{i,k})$.
   b) $\hat{y}_{i,k} = \boldsymbol{w}_i^T \boldsymbol{h}_k, \forall k$.
3) Compute $E = \frac{1}{2} \Sigma_{k=1}^{M} \Sigma_{i=1}^{n_o} (y_{i,k} - \hat{y}_{i,k})^2$.
4) if: $E > \epsilon$; then: go to Step 2).

V. Splitting and Stopping Criteria

There is a tradeoff between the ability of an RBF network to learn a desired input–output mapping and its ability to generalize. An RBF with linear output units can learn a desired input–output mapping if there are enough radial basis functions so that their responses feed the upper associative network with linearly separable inputs. However, an RBF network with an excessive number of prototypes is allowed to learn the peculiarities in the training set, which degrades its ability to generalize. This section presents splitting and stopping criteria that can be used to control the growth of GRBF neural networks during training.

A. Splitting Criteria

According to the approach proposed in this paper, the number of radial basis functions increases gradually during training. This is accomplished by creating new prototypes as indicated by the training set. Based on a splitting criterion, a new prototype is created in a region of the input space by splitting an existing one. Splitting is performed by adding the perturbation vectors $\pm \boldsymbol{d}_j$ to the prototype $\boldsymbol{v}_j$ selected by the splitting criterion. The resulting vectors $\boldsymbol{v}_j + \boldsymbol{d}_j$ and $\boldsymbol{v}_j - \boldsymbol{d}_j$ are used together with the existing prototypes to form the initial set of prototypes used in the next growing cycle. The perturbation vector $\boldsymbol{d}_j$ can be obtained in terms of a deviation measure computed from the input vectors represented by the prototype $\boldsymbol{v}_j$ and the prototype itself. Let $\mathcal{I}_j$ be the set of input vectors represented by the prototype $\boldsymbol{v}_j$, defined at (16), and

$$d_{i,j}^2 = \frac{1}{|\mathcal{I}_j|} \sum_{k: \boldsymbol{x}_k \in \mathcal{I}_j} (x_{i,k} - v_{i,j})^2, \quad 1 \leq i \leq n_i \tag{26}$$

where $|\mathcal{I}_j|$ denotes the cardinality of $\mathcal{I}_j$. The perturbation vector $\boldsymbol{d}_j$ can be obtained as

$$\boldsymbol{d}_j = d_0 [d_{1,j} \quad d_{2,j} \quad \cdots \quad d_{n_i,j}]^T \tag{27}$$

where d_0 is a scaling constant which guarantees that $||\boldsymbol{d}_j|| \ll ||\boldsymbol{v}_j||$.

1) Feedback Splitting Criterion: This criterion uses the response of the network to decide which of the existing prototypes should be split after a certain growing cycle. Each input vector $\boldsymbol{x}_k \in \mathcal{X}$ presented to the network is represented by its the nearest prototype in the Euclidean distance sense. In other words, $\boldsymbol{x}_k \in \mathcal{X}$ is represented by $\boldsymbol{v}_{t_k}$ if $||\boldsymbol{x}_k - \boldsymbol{v}_{t_k}|| < ||\boldsymbol{x}_k - \boldsymbol{v}_j||, \forall j \neq t_k$. The response of the network to each input vector $\boldsymbol{x}_k \in \mathcal{X}$ is also determined. The counter m_{t_k} corresponding to $\boldsymbol{v}_{t_k}$ is incremented every time an input vector represented by $\boldsymbol{v}_{t_k}$ is assigned to a wrong class by the network. In other words, m_{t_k} counts the number of input vectors represented by the prototype $\boldsymbol{v}_{t_k}$ that are incorrectly classified by the network. After all inputs $\boldsymbol{x}_k \in \mathcal{X}$ are presented to the network, the counters corresponding to all existing prototypes are compared. The prototype responsible for most of the incorrect classifications is split. The feedback splitting criterion is described in detail by the following:

Set $m_j = 0, 1 \leq j \leq c$.
For all $k = 1, 2, \cdots, M$:
  Determine $t_k \in \{1, 2, \cdots, c\}$ such that
    $||\boldsymbol{x}_k - \boldsymbol{v}_{t_k}|| < ||\boldsymbol{x}_k - \boldsymbol{v}_j||, \forall j \neq t_k$.
  If $\boldsymbol{x}_k$ is assigned to a wrong class
    then $m_{t_k} = m_{t_k} + 1$.
If $m_p > m_j, \forall j \neq p$, then split the pth prototype.

2) Purity Splitting Criterion: A cluster of input vectors is called *pure* if it contains only input vectors belonging to a single class. Since the input vectors may be clustered by an unsupervised learning process, the resulting clusters may contain input vectors belonging to different classes. Suppose the network assigns one or more input vectors to a wrong class. This is an indication that the upper associative network is unable to classify correctly the input vectors from clusters which are not pure. The purity splitting criterion selects the prototype to be split on the basis of the purity of the clusters of input vectors represented by the existing prototypes. According to this criterion, the prototype that represents the least pure cluster is selected for splitting after every growing cycle. A counter vector $\boldsymbol{s}_j = [s_{j,1}, s_{j,2}, \cdots, s_{j,n_o}]^T$ is assigned to each prototype $\boldsymbol{v}_j, 1 \leq j \leq c$. The ith component of $\boldsymbol{s}_j$ is incremented if the input vector $\boldsymbol{x}_k \in \mathcal{X}$ belongs to the ith class. After all input vectors are presented to the network, the components of $\boldsymbol{s}_{t_k}$ describe how the input vectors $\boldsymbol{x}_k \in \mathcal{X}$

represented by $\boldsymbol{v}_{t_k}$ are distributed over various classes. Each prototype $\boldsymbol{v}_j$ is also assigned an additional counter m_j, defined as $m_j = \max_{1 \le i \le n_o} \{s_{j,i}\}, 1 \le j \le c$. This counter gives the maximum number of input vectors represented by the prototype $\boldsymbol{v}_j$ that belongs to a certain class. The least pure prototype $\boldsymbol{v}_p$ is selected for splitting according to

$$\sum_{i=1}^{n_o} s_{p,i} - m_p > \sum_{i=1}^{n_o} s_{j,i} - m_j, \quad \forall j \neq p. \tag{28}$$

The purity splitting criterion is described in detail by the following:

Set $\boldsymbol{s}_j = [s_{j1}, s_{j2}, \cdots, s_{jn_o}]^T = [0, 0, \cdots, 0]^T, 1 \le j \le c$.
For all $k = 1, 2, \cdots, M$:
  Determine $t_k \in \{1, 2, \cdots, c\}$ such that
    $||\boldsymbol{x}_k - \boldsymbol{v}_{t_k}|| < ||\boldsymbol{x}_k - \boldsymbol{v}_j||, \forall j \neq t_k$.
  If $\boldsymbol{x}_k$ is assigned to the ith class
    then $s_{t_k,i} = s_{t_k,i} + 1$.
Determine $m_j = \max_{1 \le i \le n_o} \{s_{j,i}\}, 1 \le j \le c$.
If $\Sigma_{i=1}^{n_o} s_{p,i} - m_p > \Sigma_{i=1}^{n_o} s_{j,i} - m_j, \forall j \neq p$
  then split the pth prototype.

B. Stopping Criterion

If the training set contains no overlapping data, then a GRBF network can be trained until the error evaluated on the training set becomes sufficiently small. In pattern classification problems, for example, a GRBF can be trained until the number of classification errors reduces to zero. In most pattern classification applications, however, some training examples from one class appear in a region of the feature space that is mostly occupied by examples from another class. By increasing the number of prototypes, each prototype would represent very few training examples. This makes it possible for the network to map correctly onto its output training examples in overlapping regions. However, such a strategy would have a negative impact on the ability of the trained network to generalize. This phenomenon is often referred to in the literature as *overtraining*.

In order to avoid overtraining, the stopping criterion used in this approach employs a *testing* or *cross-validation* set in addition to the training set [8]. Both training and testing sets are formed from the same data set. The errors on both training and testing sets are recorded after each growing cycle. Both these errors reduce at the early stages of learning. At a certain point in the training process, the error on the testing set begins increasing even though the error on the training set reduces. This is an indication of overtraining and can be used to terminate the training process. In practice, the point in time where overtraining begins is not clear because of the fluctuation of both errors and the dependency on the objective function. The following strategy is used in practice to terminate training: Both errors are recorded after each growing cycle. The network keeps growing even if there is a temporary increase in the error evaluated on the testing set. The status of the network (i.e., the weights of the upper network, the prototypes, and the widths of the Gaussian functions) is recorded if the error on the testing set decreases after a growing cycle. If the error on the testing set does not decrease after a sufficient number of growing cycles, then the training is terminated while the network that resulted in the smallest error on the testing set is the final product of the learning process.

VI. Experimental Results

A. Two-Dimensional Vowel Data

In this set of experiments the performance of GRBF neural networks was evaluated using a set of two-dimensional (2-D) vowel data formed by computing the first two formants F1 and F2 from samples of ten vowels spoken by 67 speakers [17], [20]. This data set has been extensively used to compare different pattern classification approaches because there is significant overlapping between the vectors corresponding to different vowels in the F1-F2 plane [17], [20]. The available 671 feature vectors were divided into a training set, containing 338 vectors, and a testing set, containing 333 vectors.

This training set is ideal for testing the stopping criterion employed in this approach because of the overlapping between the classes. The operation of the stopping criterion is illustrated in Fig. 3, which shows the number of classification errors on the training and testing sets recorded during the training of a GRBF network as a function of the growing cycles. The classification errors on the training set decreased with some fluctuations during the training of the network. In contrast, the classification errors on the testing set decreased during the initial growing cycles but began increasing again after 33 growing cycles. Fig. 3 indicates that the GRBF network creates a more detailed partition of the input space by creating additional prototypes in order to implement the mapping required by the training set. However, this degrades the ability of the network to generalize as indicated by the number of classification errors on the testing set. The arrow in Fig. 3 indicates the point where training is terminated according to the stopping criterion, which prevents the degradation of the performance of the trained GRBF on the testing set.

The training and testing sets formed from the 2-D vowel data were classified by several GRBF networks trained using various combinations of learning rules and splitting criteria. The prototypes were formed in these experiments using the FALVQ 1, HCM and FCM algorithms. The weights of the upper associative network were updated using the ELEANNE 2 and gradient descent. The widths of the Gaussian radial basis function were computed using the "average of close inputs" heuristic. Splitting of the prototypes during training was based on the feedback and purity stopping criteria. Table I summarizes the number and percentage of feature vectors from the training and testing sets classified incorrectly by each GRBF network. Table I also shows the number of radial basis functions created by each network during training, which varied from 20–35. Although the number of classification errors fluctuates for the networks tested in these experiments, Table I offers no basis for drawing conclusions about the efficiency of different learning rules and splitting criteria.

The same experiments were repeated by updating the prototypes and the widths of the radial basis functions so as to min-

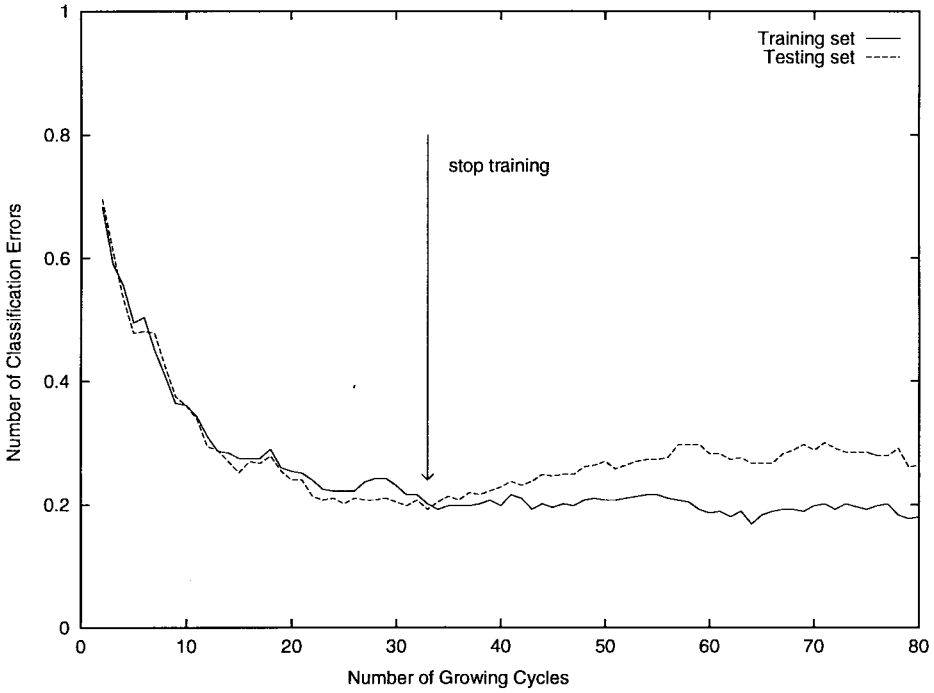

Fig. 3. Number of feature vectors from the training and testing sets formed from the 2-D vowel data classified incorrectly by the GRBF network as a function of the number of growing cycles. The arrow indicates termination of training according to the stopping criterion employed.

TABLE I
THE PERFORMANCE OF VARIOUS GRBF NETWORKS EVALUATED ON THE 2-D VOWEL DATA SET: NUMBER/PERCENTAGE OF INCORRECTLY CLASSIFIED FEATURE VECTORS FROM THE TRAINING SET ($E_{\rm train}/\%E_{\rm train}$) AND THE TESTING SET ($E_{\rm test}/\%E_{\rm test}$) AND THE FINAL NUMBER c OF RADIAL BASIS FUNCTIONS. PARAMETER SETTING: $N = 30, \eta_0 = 0.001$ FOR FALVQ 1; $\epsilon = 0.0001$ FOR HCM; $\epsilon = 0.0001, m = 1.1$ FOR FCM; $\eta = 0.5, \epsilon = 0.001$ FOR ELEANNE 2; $\eta = 0.001, \epsilon = 0.001$ FOR GRADIENT DESCENT

Lower net	Upper net	Splitting	E_{train}	$\%E_{train}$	E_{test}	$\%E_{test}$	c
FALVQ 1	ELEANNE 2	Feedback	80	23.67%	72	21.62%	20
FALVQ 1	ELEANNE 2	Purity	78	23.07%	71	21.32%	27
FALVQ 1	Gradient descent	Feedback	75	22.19%	69	20.72%	28
FALVQ 1	Gradient descent	Purity	78	23.08%	70	21.02%	23
HCM	ELEANNE 2	Feedback	85	25.15%	76	22.82%	25
HCM	ELEANNE 2	Purity	77	22.78%	76	22.82%	35
HCM	Gradient descent	Feedback	77	22.78%	70	21.02%	33
HCM	Gradient descent	Purity	86	25.44%	82	24.62%	28
FCM	ELEANNE 2	Feedback	81	23.96%	73	21.92%	33
FCM	ELEANNE 2	Purity	95	28.11%	80	24.02%	21
FCM	Gradient descent	Feedback	80	23.67%	77	23.12%	33
FCM	Gradient descent	Purity	92	27.22%	76	22.82%	28
	Average		82	24.26%	74.3	22.31%	27.8

imize the localized class-conditional variance. The ELEANNE 2 and gradient descent were used to update the weights of the upper associative network while splitting of the prototypes was based on the feedback and purity splitting criteria. The results of these experiments are summarized in Table II. Comparison of the average number of classification errors summarized in Tables I and II indicates that updating the prototypes and the widths of the radial basis functions by minimizing the localized class-conditional variance improved the performance of the trained GRBF networks on both training and testing sets. On the average, these learning rules resulted in trained GRBF networks with fewer radial basis functions.

TABLE II
THE PERFORMANCE OF VARIOUS GRBF NETWORKS EVALUATED ON THE 2-D VOWEL DATA SET: NUMBER/PERCENTAGE OF FEATURE VECTORS FROM THE TRAINING SET ($E_{\text{train}}/\%E_{\text{train}}$) AND THE TESTING SET ($E_{\text{test}}/\%E_{\text{test}}$) CLASSIFIED INCORRECTLY AND THE FINAL NUMBER c OF RADIAL BASIS FUNCTIONS. THE PROTOTYPES AND WIDTHS OF THE RADIAL BASIS FUNCTIONS WERE UPDATED BY MINIMIZING THE LOCALIZED CLASS-CONDITIONAL VARIANCE. PARAMETER SETTING: $\eta = 0.5, \epsilon = 0.001$ FOR ELEANNE 2; $\eta = 0.001, \epsilon = 0.001$ FOR GRADIENT DESCENT; $\eta = 0.001$ FOR CLASS-CONDITIONAL VARIANCE MINIMIZATION

Upper net	Splitting	E_{train}	$\%E_{train}$	E_{test}	$\%E_{test}$	c
ELEANNE 2	Feedback	73	21.60%	68	20.42%	25
ELEANNE 2	Purity	78	23.08%	72	21.62%	22
Gradient descent	Feedback	77	22.78%	70	21.02%	21
Gradient descent	Purity	82	24.26%	70	21.02%	25
Average		77.5	22.93%	70	21.02%	23.3

Figs. 4 and 5 show the partition of the feature space produced by two GRBF networks trained by employing the feedback and purity splitting criteria, respectively. In both experiments, the prototypes were updated to minimize the localized class-conditional variance while the weights of the upper associative network were updated using the ELEANNE 2 algorithm. It can be seen from Figs. 4 and 5 that the GRBF networks attempt to find the best possible compromise in regions of the input space with extensive overlapping between the classes. The two splitting criteria used to train the network produced a similar partition of the feature space although some of the regions formed differ in terms of both shape and size. For example, there are some remarkable differences in the shape and size of the regions that represent the phonemes "had," "hod," and "who'd."

The last set of experiments on the 2-D vowel data compared the performance and training time requirements of GRBF and conventional feedforward neural networks. The GRBF network was trained by employing minimization of the localized class-conditional variance for updating the prototypes, the ELEANNE 2 for updating the weights of the upper associative network and the feedback splitting criterion. The trained GRBF network consisted of $c = 25$ radial basis functions, which is close to the average number of radial basis functions created by the proposed learning scheme for the GRBF networks trained in these experiments. Since the number of growing cycles depends only on the final number of radial basis functions in the trained GRBF network, the time required for constructing and training this particular network is close to the average time required for training GRBF networks in these experiments. Two feedforward neural networks with five and ten hidden units were also trained to perform the same classification task by the error backpropagation algorithm [25]. Table III shows the number and percentage of classification errors produced by the trained networks tested and the central processing unit (CPU) time required for their training. All networks tested in these experiments classified incorrectly almost the same number of feature vectors from both training and testing sets. The GRBF network was trained slightly faster than both feedforward neural networks and especially the one consisting of five hidden units. Fig. 6 shows the partition of the input vector space produced by a conventional feedforward neural network with one layer of five hidden units. This network partitioned the input vector space into different regions by line segments. This is consistent with the role of the hidden units in this network. Because of their structure, GRBF networks partitioned the input space by drawing more circular segments.

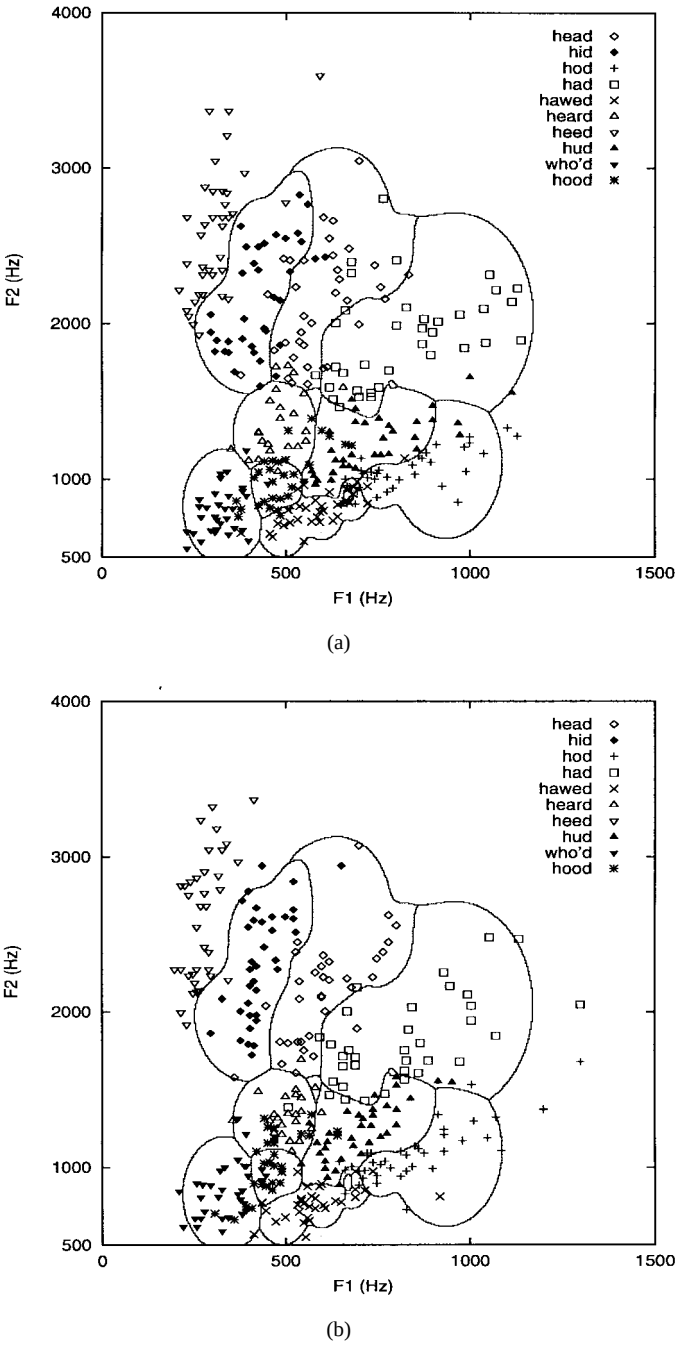

Fig. 4. Classification of the 2-D vowel data produced by the GRBF network: (a) the training data set and (b) the testing data set. The prototypes were updated to minimize the localized class-conditional variance, training of the upper network was based on ELEANNE 2, and splitting was based on the feedback splitting criterion.

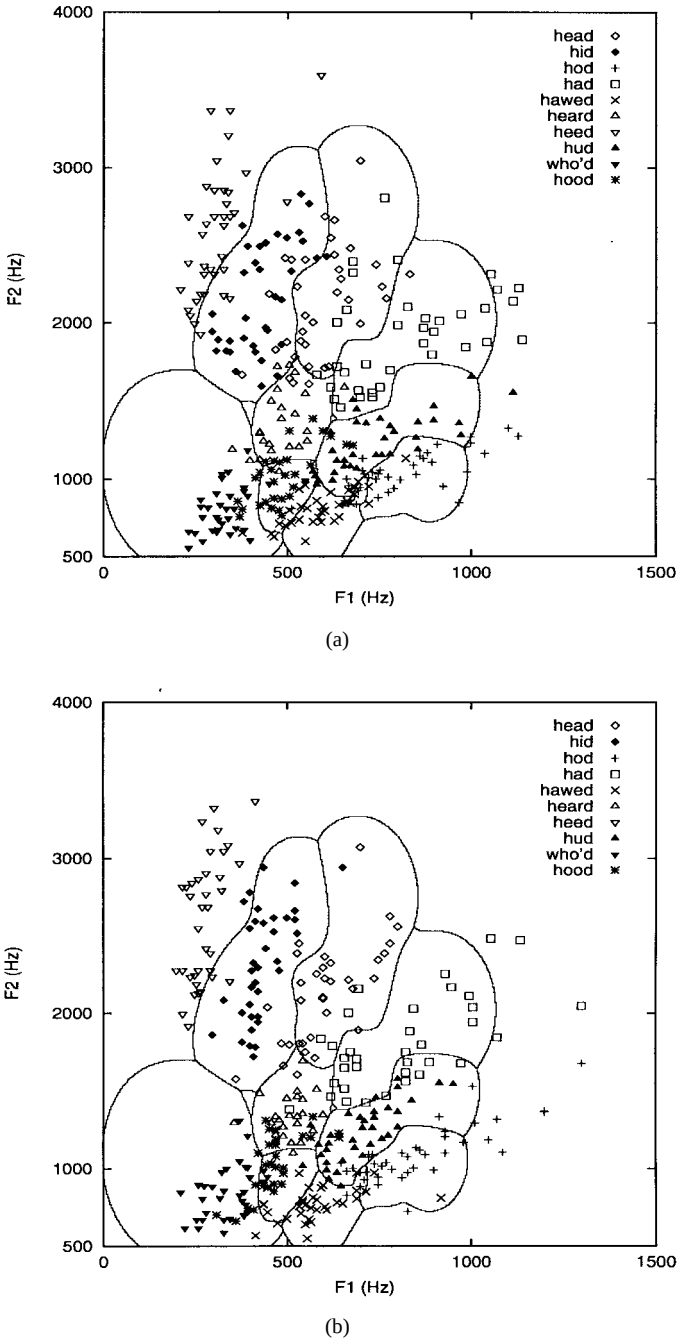

Fig. 5. Classification of the 2-D vowel data produced by the GRBF network: (a) the training data set and (b) the testing data set. The prototypes were updated to minimize the localized class-conditional variance, training of the upper network was based on ELEANNE 2, and splitting was based on the purity splitting criterion.

B. The IRIS Data

The proposed GRBF networks were also tested using Anderson's IRIS data set [1]. This data set contains 150 feature vectors of dimension 4, which belong to three classes representing different IRIS subspecies. Each class contains 50 feature vectors. One of the three classes is well separated from the other two, which are not easily separable due to the overlapping of their convex hulls. The 150 examples contained in the IRIS data set were randomly split to form the training and testing sets, each containing 75 examples.

TABLE III
PERFORMANCE AND TRAINING TIME REQUIREMENTS OF A GRBF NEURAL NETWORK AND TWO FEEDFORWARD NEURAL NETWORKS TRAINED USING THE ERROR BACKPROPAGATION ALGORITHM: NUMBER/PERCENTAGE OF INCORRECTLY CLASSIFIED FEATURE VECTORS FROM THE TRAINING SET ($E_{\text{train}}/\%E_{\text{train}}$) AND TESTING SET ($E_{\text{test}}/\%E_{\text{test}}$) AND CPU TIME REQUIRED FOR TRAINING ON A DEC ALPHA WORKSTATION (166 MHz)

Network	E_{train}	$\%E_{train}$	E_{test}	$\%E_{test}$	CPU time (sec)
GRBF	73	21.60%	68	20.42%	73.54
FFNN (5 hidden units)	69	20.41%	78	23.42%	100.63
FFNN (10 hidden units)	68	20.12%	66	19.82%	89.50

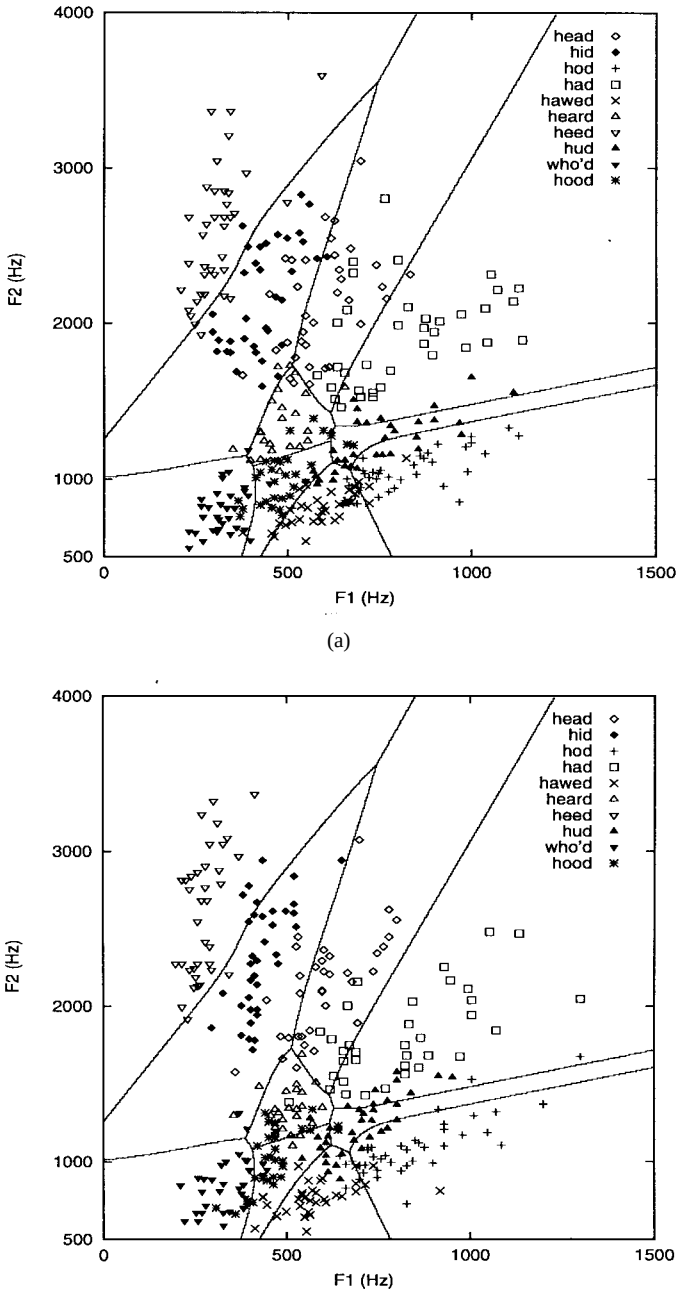

Fig. 6. Classification of the 2-D vowel data produced by a feedforward neural network with a hidden layer of five units: (a) the training data set and (b) the testing data set.

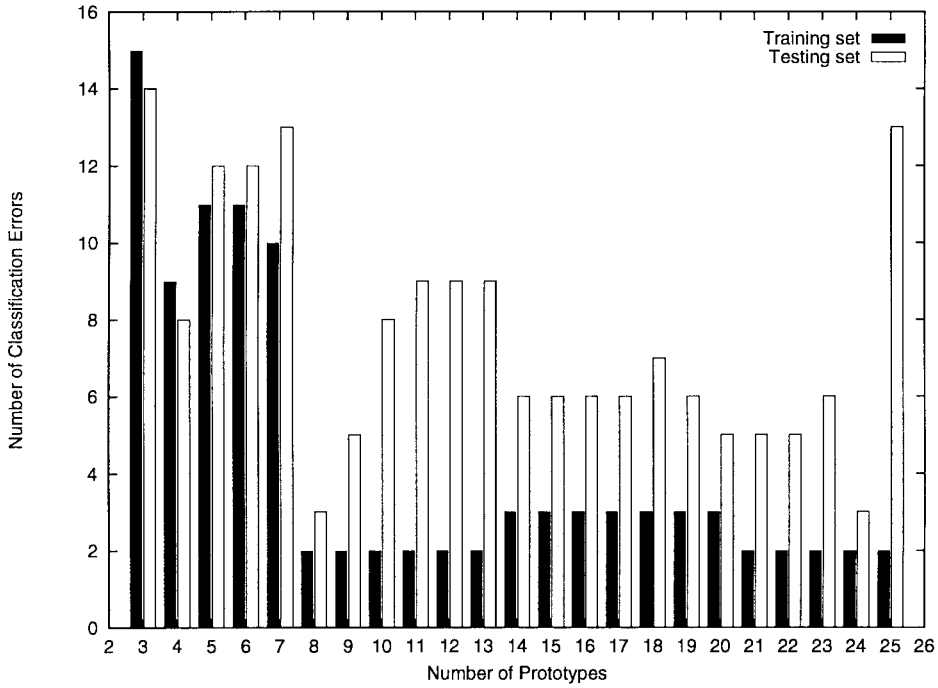

Fig. 7. Feature vectors from the training and testing sets formed from the IRIS data classified incorrectly by conventional RBF networks. The number of radial basis functions was fixed during training and varied from three to 25.

The IRIS data were used to evaluate the performance of conventional RBF networks with a predetermined number of radial basis functions. The prototypes representing the feature vectors were formed using the HCM algorithm while the weights of the upper associative network were updated using gradient descent. The number c of radial basis functions varied in these experiments from 3 to 25. Fig. 7 shows the number of feature vectors from the training and testing sets formed from the IRIS data classified incorrectly by the trained RBF networks. The trained RBF networks consisting of three–seven radial basis functions resulted in a very high number of classification errors on both training and testing sets. For comparison, note that unsupervised clustering algorithms tested on the IRIS data set typically result in 12–17 errors [12], [13], [21]. The performance of the trained RBF network on the training set improved as the number of radial basis functions increased above seven. For $c > 7$, the RBF network classified incorrectly two or three feature vectors from the training set. However, the performance of the trained RBF networks on the testing set was erratic. The RBF networks classified incorrectly 3–13 feature vectors from the testing set as the number of radial basis functions increased. This experimental outcome reveals an important disadvantage of conventional RBF networks and also justifies the growing strategy proposed in this paper for GRBF networks.

The training and testing sets formed from the IRIS data were classified by several GRBF networks trained using various combinations of learning rules and splitting criteria. The prototypes were formed in these experiments using the FALVQ 1, HCM and FCM algorithms. The weights of the upper associative network were updated using the ELEANNE 2 and gradient descent. The widths of the Gaussian radial basis function were computed using the "average of close inputs" heuristic. Splitting of the prototypes during training was based on the feedback and purity stopping criteria. Table IV summarizes the number and percentage of feature vectors from the training and testing sets classified incorrectly by each GRBF network. For each network, Table IV also shows the number of radial basis functions created during training. Although the number of classification errors on the training set varied from zero to six, most of the trained GRBF networks classified incorrectly three of the vectors included in the testing set. On the average, the GRBF networks employing the FCM classified incorrectly more training vectors compared with the GRBF networks employing the FALVQ 1 and HCM algorithms. On the other hand, both FALVQ 1 and HCM algorithms created more prototypes than the FCM algorithm.

The same experiments were repeated by updating the prototypes and the widths of the radial basis functions so as to minimize the localized class-conditional variance. The ELEANNE 2 and gradient descent were used to update the weights of the upper associative network while splitting of the prototypes was based on the feedback and purity splitting criteria. The results of these experiments are summarized in Table V. The learning rules used for updating the prototypes and the widths of the radial basis functions improved the performance of the trained GRBF networks on both training and testing sets. The GRBF networks employing the ELEANNE 2 classified correctly all vectors included in the training set. The ELEANNE 2 resulted in trained GRBF networks with more prototypes compared with those employing the gradient descent. In fact, the gradient descent resulted in trained GRBF networks with very few radial basis functions. The generalization ability of all trained GRBF networks was very satisfactory regardless

TABLE IV
THE PERFORMANCE OF VARIOUS GRBF NETWORKS EVALUATED ON THE IRIS DATA SET: NUMBER/PERCENTAGE OF INCORRECTLY CLASSIFIED FEATURE VECTORS FROM THE TRAINING SET ($E_{\text{train}}/\%E_{\text{train}}$) AND THE TESTING SET ($E_{\text{test}}/\%E_{\text{test}}$) AND THE FINAL NUMBER c OF RADIAL BASIS FUNCTIONS. PARAMETER SETTING: $N = 30, \eta_0 = 0.01$ FOR FALVQ 1; $\epsilon = 0.0001$ FOR HCM; $\epsilon = 0.0001, m = 1.1$ FOR FCM; $\eta = 1.0, \epsilon = 0.0005$ FOR ELEANNE 2; $\eta = 0.01, \epsilon = 0.0005$ FOR GRADIENT DESCENT

Lower net	Upper net	Splitting	E_{train}	$\%E_{train}$	E_{test}	$\%E_{test}$	c
FALVQ 1	ELEANNE 2	Feedback	2	2.67%	2	2.67%	11
FALVQ 1	ELEANNE 2	Purity	3	4.00%	3	4.00%	10
FALVQ 1	Gradient descent	Feedback	3	4.00%	3	4.00%	10
FALVQ 1	Gradient descent	Purity	1	1.33%	3	4.00%	11
HCM	ELEANNE 2	Feedback	0	0.00%	3	4.00%	15
HCM	ELEANNE 2	Purity	2	2.67%	3	4.00%	8
HCM	Gradient descent	Feedback	1	1.33%	3	4.00%	9
HCM	Gradient descent	Purity	6	8.00%	3	4.00%	7
FCM	ELEANNE 2	Feedback	4	5.33%	2	2.67%	6
FCM	ELEANNE 2	Purity	4	5.33%	2	2.67%	6
FCM	Gradient descent	Feedback	5	6.67%	3	4.00%	6
FCM	Gradient descent	Purity	5	6.67%	3	4.00%	6
Average			3	4.00%	2.8	3.73%	8.6

TABLE V
THE PERFORMANCE OF VARIOUS GRBF NETWORKS EVALUATED ON THE IRIS DATA SET: NUMBER/PERCENTAGE OF FEATURE VECTORS FROM THE TRAINING SET ($E_{\text{train}}/\%E_{\text{train}}$) AND THE TESTING SET ($E_{\text{test}}/\%E_{\text{test}}$) CLASSIFIED INCORRECTLY AND THE FINAL NUMBER c OF RADIAL BASIS FUNCTIONS. THE PROTOTYPES AND WIDTHS OF THE RADIAL BASIS FUNCTIONS WERE UPDATED BY MINIMIZING THE LOCALIZED CLASS-CONDITIONAL VARIANCE. PARAMETER SETTING: $\eta = 1.0, \epsilon = 0.0005$ FOR ELEANNE 2; $\eta = 0.01, \epsilon = 0.0005$ FOR GRADIENT DESCENT; $\eta = 0.005$ FOR CLASS-CONDITIONAL VARIANCE MINIMIZATION

Upper net	Splitting	E_{train}	$\%E_{train}$	E_{test}	$\%E_{test}$	c
ELEANNE 2	Feedback	0	0.00%	1	1.33%	11
ELEANNE 2	Purity	0	0.00%	2	2.67%	15
Gradient descent	Feedback	1	1.33%	1	1.33%	7
Gradient descent	Purity	1	1.33%	1	1.33%	7
Average		0.5	0.67%	1.25	1.67%	10

of the learning scheme employed for updating the weights of the upper associative network. With only one exception, the GRBF networks trained in these experiments classified incorrectly only one vector from the testing set.

VII. CONCLUSIONS

This paper presented the development, testing, and evaluation of a framework for constructing and training growing RBF neural networks. The hybrid learning scheme proposed in this paper allows the participation of the training set in the creation of the prototypes, even if the prototypes are created using an unsupervised algorithm. This is accomplished through the proposed splitting criteria, which relate directly to the training set and create prototypes in certain locations of the input space which are crucial for the implementation of the input–output mapping. This strategy allows the representation of the input space of different resolution levels by creating few prototypes to represent smooth regions in the input space and utilizing the available resources where they are actually necessary. The stopping criterion employed in this paper prevents the creation of degenerate prototypes that could compromise the generalization ability of trained GRBF networks by stopping their growth as indicated by their performance on a testing set. The proposed hybrid scheme is compatible with a variety of unsupervised clustering and LVQ algorithms that can be used to determine the locations and widths of the radial basis functions. This paper also proposed a new supervised scheme for updating the locations and widths of the radial basis functions during learning. This scheme is based on the minimization of a localized class-conditional variance measure, which is computed from the outputs of the radial basis functions and relates to the structure of the feature space.

A variety of learning algorithms were employed in this paper for constructing and training GRBF neural networks. The weights of the upper associative network were updated in the experiments using the ELEANNE 2, a recursive least-squares algorithm, and gradient descent. The prototypes were updated by an enhanced version of the c-means algorithm, the fuzzy c-means algorithm, and the FALVQ 1 algorithm. The localized class-conditional variance criterion was also used in the experiments to update the prototypes as well as

the widths of the radial basis functions. The performance of the trained GRBF networks was compared with that of conventional RBF and feedforward neural networks. The experiments revealed a significant limitation of RBF neural networks with a fixed number of hidden units: Increasing the number of radial basis functions did not significantly improve their performance. In fact, trained GRBF neural networks performed better than conventional RBF networks containing a larger number of radial basis functions. If the prototypes are created by an unsupervised process, the inferior performance of conventional RBF networks can be attributed to the following reasons: 1) The training set does not participate in the creation of the prototypes which is based on the criterion used for developing the clustering and LVQ algorithm. 2) The unsupervised algorithm can waste the resources of the network, namely the radial basis functions, by creating prototypes in insignificant regions while ignoring regions which are important for implementing the input–output mapping. The experiments indicated that the performance of trained GRBF networks is mostly affected by the algorithm used for updating the prototypes after splitting. The GRBF networks employing minimization of the localized class-conditional variance achieved the most robust performance in pattern classification tasks among those tested in the experiments. What distinguishes this learning scheme from those employing unsupervised algorithms to update the prototypes is that the input vectors are treated as labeled feature vectors during training. Thus, the training set has a direct impact on the formation of the prototypes in addition to its role in the splitting of the prototypes.

APPENDIX

Updating the Locations of Radial Basis Functions

The gradient of G with respect to the prototype $\boldsymbol{v}_p$ that determines the location of the pth radial basis function is

$$\frac{\partial G}{\partial \boldsymbol{v}_p} = \frac{1}{2}\frac{\partial}{\partial \boldsymbol{v}_p}\sum_{\ell=1}^{n_o}\sum_{j=1}^{c} S_{j,\ell}^2 = \frac{1}{2}\sum_{\ell=1}^{n_o}\frac{\partial}{\partial \boldsymbol{v}_p} S_{p,\ell}^2. \tag{A1}$$

From the definition of $S_{j,\ell}^2$ in (8)

$$\frac{\partial}{\partial \boldsymbol{v}_p} S_{p,\ell}^2 = 2\sum_{k:\boldsymbol{x}_k\in\mathcal{C}_\ell\cap\mathcal{P}_p} (\langle h_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle - h_{p,k}) \cdot \frac{\partial}{\partial \boldsymbol{v}_p}(\langle h_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle - h_{p,k}). \tag{A2}$$

Since $h_{j,k} = \exp(-||\boldsymbol{x}_k - \boldsymbol{v}_j||^2/\sigma_j^2)$

$$\frac{\partial}{\partial \boldsymbol{v}_p} h_{p,k} = 2\boldsymbol{e}_{p,k} \tag{A3}$$

where

$$\boldsymbol{e}_{p,k} = \frac{h_{p,k}}{\sigma_p^2}(\boldsymbol{x}_k - \boldsymbol{v}_p). \tag{A4}$$

The definition of $\langle h_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle$ in (11) gives

$$\frac{\partial}{\partial \boldsymbol{v}_p}\langle h_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle = \frac{1}{|\mathcal{C}_\ell\cap\mathcal{P}_p|}\sum_{k:\boldsymbol{x}_k\in\mathcal{C}_\ell\cap\mathcal{P}_p}\frac{\partial}{\partial \boldsymbol{v}_p} h_{p,k} = 2\langle \boldsymbol{e}_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle \tag{A5}$$

where

$$\langle \boldsymbol{e}_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle = \frac{1}{|\mathcal{C}_\ell\cap\mathcal{P}_p|}\sum_{k:\boldsymbol{x}_k\in\mathcal{C}_\ell\cap\mathcal{P}_p}\boldsymbol{e}_{p,k}. \tag{A6}$$

The gradient of G with respect to $\boldsymbol{v}_p$ can be obtained from (A1) after combining (A2) with (A3) and (A5) as

$$\frac{\partial G}{\partial \boldsymbol{v}_p} = 2\sum_{\ell=1}^{n_o}\sum_{k:\boldsymbol{x}_k\in\mathcal{C}_\ell\cap\mathcal{P}_p}(\langle h_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle - h_{p,k}) \cdot (\langle \boldsymbol{e}_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle - \boldsymbol{e}_{p,k}). \tag{A7}$$

Updating the Widths of Radial Basis Functions

The gradient of G with respect to the width of the pth radial basis function is

$$\frac{\partial G}{\partial \sigma_p} = \frac{1}{2}\frac{\partial}{\partial \sigma_p}\sum_{\ell=1}^{n_o}\sum_{j=1}^{c} S_{j,\ell}^2 = \frac{1}{2}\sum_{\ell=1}^{n_o}\frac{\partial}{\partial \sigma_p} S_{p,\ell}^2. \tag{A8}$$

From the definition of $S_{p,\ell}^2$ in (8)

$$\frac{\partial}{\partial \sigma_p} S_{p,\ell}^2 = 2\sum_{k:\boldsymbol{x}_k\in\mathcal{C}_\ell\cap\mathcal{P}_p}(\langle h_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle - h_{p,k}) \cdot \frac{\partial}{\partial \sigma_p}(\langle h_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle - h_{p,k}). \tag{A9}$$

Since $h_{j,k} = \exp(-||\boldsymbol{x}_k - \boldsymbol{v}_j||^2/\sigma_j^2)$

$$\frac{\partial}{\partial \sigma_p} h_{p,k} = 2\tilde{h}_{p,k} \tag{A10}$$

where

$$\tilde{h}_{p,k} = \frac{||\boldsymbol{x}_k - \boldsymbol{v}_p||^2}{\sigma_p^3} h_{p,k}. \tag{A11}$$

The definition of $\langle h_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle$ in (11) gives

$$\frac{\partial}{\partial \sigma_p}\langle h_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle = \frac{1}{|\mathcal{C}_\ell\cap\mathcal{P}_p|}\sum_{k:\boldsymbol{x}_k\in\mathcal{C}_\ell\cap\mathcal{P}_p}\frac{\partial}{\partial \sigma_p} h_{p,k} = 2\langle \tilde{h}_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle \tag{A12}$$

where

$$\langle \tilde{h}_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle = \frac{1}{|\mathcal{C}_\ell\cap\mathcal{P}_p|}\sum_{k:\boldsymbol{x}_k\in\mathcal{C}_\ell\cap\mathcal{P}_p}\tilde{h}_{p,k}. \tag{A13}$$

The gradient of G with respect to σ_p can be obtained from (A8) after combining (A9) with (A10) and (A12) as

$$\frac{\partial G}{\partial \sigma_p} = 2\sum_{\ell=1}^{n_o}\sum_{k:\boldsymbol{x}_k\in\mathcal{C}_\ell\cap\mathcal{P}_p}(\langle h_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle - h_{p,k}) \cdot (\langle \tilde{h}_{p,\mathcal{C}_\ell\cap\mathcal{P}_p}\rangle - \tilde{h}_{p,k}). \tag{A14}$$

REFERENCES

[1] E. Anderson, "The IRISes of the Gaspe peninsula," *Bull. Amer. IRIS Soc.*, vol. 59, pp. 2–5, 1939.

[2] M. R. Berthold and J. Diamond, "Boosting the performance of RBF networks with dynamic decay adjustment," in *Advances in Neural Inform. Processing Syst. 3*, R. P. Lippmann *et al.*, Eds. San Mateo, CA: Morgan Kaufmann, 1991.

[3] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.

[4] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Syst.*, vol. 2, pp. 321–355, 1988.

[5] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Contr., Signals, Syst.*, vol. 2, pp. 303–314, 1989.

[6] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.

[7] B. Fritzke, "Growing cell structures—A self-organizing network for unsupervised and supervised learning," Int. Computer Sci. Inst., Univ. California at Berkeley, Tech. Rep. TR–93–026, May 1993.

[8] R. Hecht–Nielsen, *Neurocomputing*. Reading, MA: Addison-Wesley, 1990.

[9] J.–N. Hwang, "The cascade-correlation learning: A projection pursuit learning perspective," *IEEE Trans. Neural Networks*, vol. 7, pp. 278–289, 1996.

[10] N. B. Karayiannis, "ALADIN: Algorithms for learning and architecture determination," *IEEE Trans. Circuits Syst.*, vol. 41, pp. 752–759, 1994.

[11] ———, "Gradient descent learning of radial basis neural networks," in *Proc. 1997 Int. Conf. Neural Networks (ICNN '97)*, Houston, TX, June 9–12, 1997, pp. 1815–1820.

[12] ———, "A methodology for constructing fuzzy algorithms for learning vector quantization," *IEEE Trans. Neural Networks*, vol. 8, pp. 505–518, 1997.

[13] N. B. Karayiannis and P.-I Pai, "Fuzzy algorithms for learning vector quantization," *IEEE Trans. Neural Networks*, vol. 7, pp. 1196–1211, 1996.

[14] N. B. Karayiannis and A. N. Venetsanopoulos, "Efficient learning algorithms for neural networks (ELEANNE)," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 1372–1383, 1993.

[15] ———, *Artificial Neural Networks: Learning Algorithms, Performance Evaluation, and Applications*. Boston, MA: Kluwer, 1993.

[16] C. A. Micchelli, "Interpolation of scattered data: Distance matrices and conditionally positive definite functions," *Constructive Approximation*, vol. 2, pp. 11–22, 1986.

[17] R. P. Lippmann, "Pattern classification using neural networks," *IEEE Commun. Mag.*, vol. 27, pp. 47–54, 1989.

[18] J. E. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computa.*, vol. 1, pp. 281–294, 1989.

[19] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels, "On the training of radial basis function classifiers," *Neural Networks*, vol. 5, pp. 595–603, 1992.

[20] K. Ng and R. P. Lippmann, "Practical characteristics of neural network and conventional pattern classifiers," in *Advances in Neural Inform. Processing Syst. 3*, R. P. Lippmann *et al.*, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 970–976.

[21] N. R. Pal, J. C. Bezdek, and E. C.-K. Tsao, "Generalized clustering networks and Kohonen's self-organizing scheme," *IEEE Trans. Neural Networks*, vol. 4, pp. 549–557, 1993.

[22] J. Park and I. W. Sandberg, "Universal approximation using radial basis function networks," *Neural Computa.*, vol. 3, pp. 246–257, 1991.

[23] ———, "Approximation and radial basis function networks," *Neural Computa.*, vol. 5, pp. 305–316, 1993.

[24] T. Poggio and F. Girosi, "Regularization algorithms for learning that are equivalent to multilayer networks," *Science*, vol. 247, pp. 978–982, 1990.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, D. E. Rumelhart and J. L. McClelland, Eds., vol. I. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[26] B. A. Whitehead and T. D. Choate, "Evolving space-filling curves to distribute radial basis functions over an input space," *IEEE Trans. Neural Networks*, vol. 5, pp. 15–23, 1994.

Nicolaos B. Karayiannis (S'85–M'91), for a photograph and biography, see the May 1997 issue of this TRANSACTIONS, p. 518.

Glenn Weiqun Mi was born in Shanghai, China, in 1968. He received the B.S. degree from the Department of Electronic Engineering, Fudan University, Shanghai, China, in 1990, and the M.S. degree from the Department of Electrical and Computer Engineering, University of Houston, in 1996.

He is currently a software engineer in CommTech Corporation, Columbus, OH. His research interests include radial basis neural networks, supervised and unsupervised learning, and vector quantization.