Pergamon

Neural Networks, Vol. 10, No. 8, pp. 1495–1503, 1997

Printed in Great Britain
0893–6080/97 $17.00+.00

PII: S0893-6080(97)00002-6

CONTRIBUTED ARTICLE

An Efficient Method to Construct a Radial Basis Function Neural Network Classifier

YOUNG-SUP HWANG AND SUNG-YANG BANG

Pohang University of Science and Technology

(*Received* 17 *June* 1996; *accepted* 11 *December* 1996)

Abstract—*Radial basis function neural network (RBFN) has the power of the universal function approximation. But how to construct an RBFN to solve a given problem is usually not straightforward. This paper describes a method to construct an RBFN classifier efficiently and effectively. The method determines the middle layer neurons by a fast clustering algorithm and computes the optimal weights between the middle and the output layers statistically. We applied the proposed method to construct an RBFN classifier for an unconstrained handwritten digit recognition. The experiment showed that the method could construct an RBFN classifier quickly and the performance of the classifier was better than the best result previously reported. © 1997 Elsevier Science Ltd.*

Keywords—Radial basis function, Linear discriminant function, Classification, APC-III, Clustering, GRBF, LMS, Handwritten digit recognition.

1. INTRODUCTION

Radial basis function neural network (RBFN) (Moody & Darken, 1989; Poggio & Girosi, 1990; Musavi, Ahmed, Chan, Faris, & Hummels, 1992; Broomhead & Lowe, 1988; S. Chen & Grant, 1991; Asim Roy & Miranda, 1995; Chang & Lippmann, 1993; Haykin, 1994) can be used for a wide range of application primarily because it can approximate any regular function (Park & Sandberg, 1993) and its training is faster than that of a multilayer perceptron. This faster learning speed comes from the fact that RBFN has just two layers of weights and each layer can be determined sequentially.

Despite these advantages RBFNs are not as widely used as they should be. The main reason for this seems to be that it is not straightforward to design an optimal RBFN to solve the given problem. The current paper makes an effort to improve this situation.

An RBFN consists of three layers, i.e., an input, a middle and an output layer. The input layer corresponds to the input vector space and the output layer to the pattern classes. So the whole architecture is fixed only by determining the middle layer and the weights between the middle and the output layers. The weights between the input and the middle layer are fixed when the middle layer is determined.

The typical traditional methods to determine the middle layer are either those selecting n samples randomly from the training patterns or those using clustering algorithms such as K-means or Self-Organizing Feature Map (SOFM). We will propose a method to construct the middle layer faster than those clustering algorithms and better than the random selection method.

There are two common ways to determine the weights between the middle and the output layers. One is the regularization method (Haykin, 1994) and the other is the usual gradient descent method (Moody & Darken, 1989).

The regularization method determines the weights by the matrix computation. If the regularization parameter is zero, the weights converge to the pseudo inverse solution. However, when the input dimension and the number of training patterns are large, not only is it difficult to implement the regularization method but some numerical errors may occur during the computation.

With the gradient descent method it is generally much faster to calculate the weights for an RBFN than those for a regular multilayer perceptron since the weights to be calculated for an RBFN are only of one layer. But the calculation of the weights still takes time since the

Acknowledgements: This work was supported in part by the Korea Science and Engineering Foundation, Korea Telecom, and Systems Engineering Research Institute.

Requests for reprints should be sent to Sung-Yang Bang, Department of Computer Science & Engineering, Pohang University of Science and Technology, Pohang 790-784, Korea; tel.: +82-562-279-2917; fax: +82-562-279-2299; e-mail: sybang@vision.postech.ac.kr.

gradient descent method has to go through the entire set of training patterns repeatedly. And it will take much longer when the number of the training patterns is extremely large and the dimension of the input is big. Furthermore the gradient descent method tends to settle down to a local minimum and sometimes does not even converge if the patterns of the outputs of the middle layer are not linearly separable (Monica Bianchini & Gori, 1995). A recent study (Luo, 1991) presented the conditions under which the training of a network with weights of just one layer always converges by the gradient descent method. This means that we can make the training of an RBFN always converge by the gradient descent method. However, even when it does converge, it usually takes a lot of time and effort to find the best values for parameters like the learning rate.

In order to alleviate aforementioned problems we propose a statistical approach to calculate the optimal weights efficiently. This idea is based on the observation that there are some correspondences between an RBFN classifier and a traditional statistical pattern classification method. First, an output layer neuron of an RBFN classifier can be seen as a discriminant function of the statistical classification. Second, an output of an RBFN is obtained by a linear weighted summation, as is a linear discriminant function in the statistical classification. Third, a middle layer neuron of an RBFN represents or is a prototype of some training patterns. There are various methods in statistical classification such as K-means clustering algorithms to find prototypes for a given set of patterns.

The purpose of this paper is to propose a method which combines these two ideas to construct an RBFN efficiently and to verify its effectiveness. The paper is organized as follows. Section 2 reviews the basic principle of an RBFN. Section 3 presents a method to construct the middle layer and two statistical approaches to compute the weights of the output layer. An experiment to recognize unconstrained handwritten digits using an RBFN with its result is given in Section 4. Section 5 analyzes the time complexity of the proposed method and explains how the method differs from Generalized Radial Basis Function Network (GRBF) (Haykin, 1994). Section 6 concludes the paper.

2. RADIAL BASIS FUNCTION NEURAL NETWORK

An RBFN is a three layer feed-forward network that consists of one input layer, one middle layer and one output layer as shown in Figure 1. Each input neuron corresponds to a component of an input vector **x**. The middle layer consists of n neurons and one bias neuron. Each input neuron is fully connected to the middle layer neurons except the bias one. Each middle layer neuron computes a kernel function (activation function) which is usually the following Gaussian function:

$$y_i = \begin{cases} \exp\left(-\dfrac{\|\mathbf{x}-\mathbf{c}_i\|}{2\sigma_i^2}\right) & i = 1, 2, \cdots, n \\ 1 & i = 0 \text{(bias neuron)} \end{cases} \tag{1}$$

where we call $\mathbf{c}_i$ and σ_i the center and the width of the ith neuron in the middle layer, respectively. $\|.\|$ denotes the Euclidean distance. The weight vector between the input layer and the ith middle layer neuron corresponds to the center $\mathbf{c}_i$ in eqn (1). And in an RBFN the net input to the ith middle layer neuron is $\|\mathbf{x} - \mathbf{c}_i\|$ rather than $\mathbf{x}\cdot\mathbf{c}_i$. The kernel function decreases rapidly if the width σ_i is small, and slowly if it is large. The output layer consists of m neurons which correspond to the possible classes of the problem and it is fully connected to the middle layer.

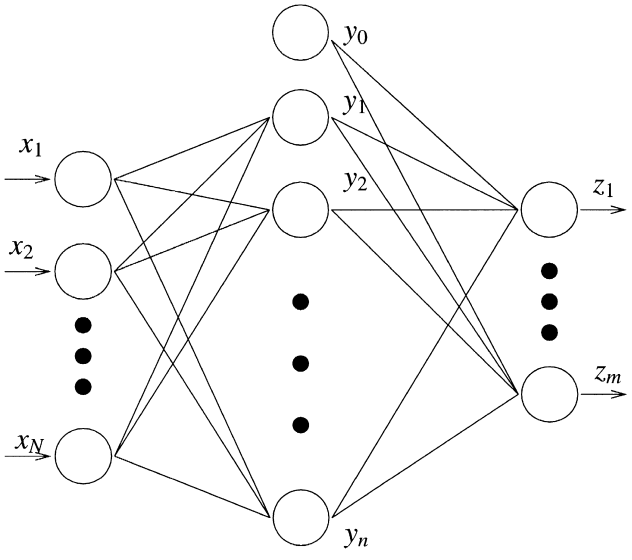

FIGURE 1. The architecture of an RBFN.

Each output layer neuron computes a linear weighted sum of the outputs of the middle layer as follows:

$$z_j = \sum_{i=0}^{n} y_i w_{ij}, j = 1, 2, \cdots, m \quad (2)$$

where w_{ij} is the weight between the ith middle layer neuron and the jth output layer neuron.

3. NEW METHOD TO CONSTRUCT AN RBFN CLASSIFIER

3.1. Determination of the Middle Layer

In order to specify the middle layer of an RBFN we have to decide the number of neurons of the layer and their kernel functions which are usually Gaussian functions. In this paper we use a Gaussian function as a kernel function. A Gaussian function is specified by its center and width. The simplest and most general method to decide the middle layer neurons is to create a neuron for each training pattern. However, the method is usually not practical since in most applications there are a large number of training patterns and the dimension of the input space is fairly large. Therefore, it is usual and practical to first cluster the training patterns to a reasonable number of groups by using a clustering algorithm such as K-means or SOFM and then to assign a neuron to each cluster. A simple way, though not always effective, is to choose a relatively small number of patterns randomly among the training patterns and create only that many neurons.

A clustering algorithm is a kind of an unsupervised learning algorithm and is used when the class of each training pattern is not known. But an RBFN is a supervised learning network. And we know at least the class of each training pattern, so we'd better take advantage of the information of these class memberships when we cluster the training patterns. Namely, we cluster the training patterns class by class instead of the entire patterns at the same time (Moody & Darken, 1989; Musavi et al., 1992). In this way we can reduce at least the total computation time required to cluster the entire training patterns since the number of patterns of each class is usually far less than that of the entire patterns.

We use a one-pass clustering algorithm called APC-III (Hwang & Bang, 1994). APC-III is similar to RCE (Reilly, Cooper, & Elbaum, 1982) but different in that APC-III has a constant radius while RCE has a variable radius. First, we decide the radius R_0 of clusters. Therefore, APC-III creates many clusters if the radius is small and few clusters if it is large. We set R_0 to the mean minimum distance between the training patterns multiplied by α:

$$R_0 = \alpha \frac{1}{P} \sum_{i=1}^{P} \min_{i \neq j} \left(\|\mathbf{x}_i - \mathbf{x}_j\| \right) \quad (3)$$

where P is the number of the training patterns. If the number of the training patterns is too large, we may well use a subset of them to obtain an approximate R_0 instead of the exact R_0. This will speed up the calculation of R_0.

Next, the following procedure is repeated to find clusters. If a given training pattern falls in the region of R_0 of any existing cluster, we include it in the cluster by adjusting the center of the cluster as described in the algorithm below. By keeping only the number of the training patterns included in the cluster, we can readily calculate the new center of the cluster. If it falls in none of the existing clusters, we create a new cluster whose center is set to the given training pattern. The outline of APC-III algorithm can be stated as follows.

3.1.1. *APC-III Algorithm.*

Input: training patterns $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_P\}$
Output: centers of clusters
var
C: number of clusters
$\mathbf{c}_j$: center of the jth cluster
n_j: number of patterns in the jth cluster
d_{ij}: distance between $\mathbf{x}_i$ and the jth cluster

begin
  $C = 1$; $\mathbf{c}_1 \leftarrow \mathbf{x}_1$; $n_1 := 1$; /* for each pattern */
  for $i := 2$ **to** P **do** /* for each cluster */
    for $j := 1$ **to** C **do**
      compute d_{ij};
      if $d_{ij} \leq R_0$ **then**
        /* include $\mathbf{x}_i$ into the jth cluster */
        $\mathbf{c}_j \leftarrow (\mathbf{c}_j n_j + \mathbf{x}_i)/(n_i + 1)$;
        $n_i := n_i + 1$;
        exit from the loop;
      end if
    end for
    if $\mathbf{x}_i$ is not included in any clusters **then**
      /* create a new cluster */
      $C := C + 1$;
      $\mathbf{c}_C \leftarrow \mathbf{x}_i$;
      $n_C := 1$;
    end if
  end for
end

APC-III is quite efficient to construct the middle layer of an RBFN since we can finish clustering by going through the entire training patterns only once. This is not true with K-means and SOFM clustering algorithms. Furthermore, APC-III tends to create an appropriate number of clusters since it determines the radius of a cluster based on the distribution of the training patterns. This fact allows APC-III to perform as good as the regular multi-pass clustering algorithms.

It should be noted that the clustering result of APC-III depends on the sequence of the pattern presentation and

the result may not be optimal. But the optimality of the clustering in case of RBFN is not clear at present and should be defined in relation to the overall performance of the network. Therefore, it seems more appropriate to have a fast clustering algorithm like APC-III than an elaborate but time-consuming one.

So far we have determined the number of the neurons and their center locations. Another parameter to be decided for each neuron in the middle layer is the width of its kernel function. The width of a kernel function should be determined so as to cover the input space as uniformly as possible. If the distances between the centers are not uniform, it will be better to assign a different width to each neuron. For example, it looks reasonable to assign a larger width where the centers are widely separated from each other and a smaller width where the centers are closer. Although one paper (Musavi et al., 1992) gave a fairly complex way to provide an optimal width, the following simple heuristic performs reasonably well in practice (Moody & Darken, 1989).

> Find the distance to the center of the nearest cluster which belongs to a different class and assign this value multiplied by β to the width.

In this paper we used this heuristic with $\beta = 5$.

3.2. Calculation of the Weights

Next we give two methods based on the standard statistical approaches to calculate the weights between the middle layer and the output layer.

3.2.1. *By Least Mean Square Error Procedure (Method I).* Let the linear discriminant functions be expressed by the following form:

$$d_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{y}, i = 1, 2, \cdots, m \tag{4}$$

where each component of $\mathbf{y}$ is the function of input $\mathbf{x}$ and $\mathbf{w}_i$ is the weight vector. Namely,

$$\mathbf{y} = \begin{pmatrix} \Phi_1(\mathbf{x}) \\ \Phi_2(\mathbf{x}) \\ \vdots \\ \Phi_n(\mathbf{x}) \\ 1 \end{pmatrix}. \tag{5}$$

Then the condition for the class ω_i to be chosen is

$$d_i(\mathbf{x}) > d_j(\mathbf{x}), \text{for all } j, j \neq i. \tag{6}$$

The least mean square (LMS) error procedure (Devijver & Kittler, 1982) tries to find the discriminant functions which best satisfy the conditions:

$$d_i(\mathbf{x}) = \begin{cases} 1 & \text{for } \mathbf{x} \in \omega_i, \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

For an m-class problem, let $\mathbf{V}_i$ designate the ith column vector of an $m \times m$ identity matrix and $\mathbf{W}$ be an $(n + 1) \times m$ matrix of weights:

$$\mathbf{W} = \left[\mathbf{w}_1, \cdots, \mathbf{w}_m\right].$$

Then the criterion function to be minimized is:

$$J(\mathbf{W}) = \sum_{i=1}^{m} P_i E_i \left\{ \|\mathbf{W}^T \mathbf{y} - \mathbf{V}_i\|^2 \right\} \tag{8}$$

where P_i and $E_i\{\}$ are the a priori probability and the expected value of class ω_i, respectively.

To find the $\mathbf{W}$ which minimizes J, we set the gradient of $J(\mathbf{W})$ to zero:

$$\nabla_{\mathbf{w}} J(\mathbf{W}) = 2 \sum_{i=1}^{m} P_i E_i \{\mathbf{y}\mathbf{y}^T\} \mathbf{W} - 2 \sum_{i=1}^{m} P_i E_i \{\mathbf{y}\} \mathbf{V}_i^T = [0], \tag{9}$$

where [0] is an $(n + 1) \times m$ null matrix. Let μ_i be the class-conditional mean vector of $\mathbf{y}$, and then we see that $E_i\{\mathbf{y}\}\mathbf{V}_i^T$ is an $(n + 1) \times m$ matrix whose column vectors are all null vectors but for the ith one which is μ_i.

Let $\mathbf{K}_i$ denote the class-conditional matrix of the second-order moments of $\mathbf{y}$, i.e.

$$\mathbf{K}_i = E_i \{\mathbf{y}\mathbf{y}^T\}. \tag{10}$$

If $\mathbf{K}$ denotes the matrix of the second-order moments under the mixture distribution, we have

$$\mathbf{K} = \sum_{i=1}^{m} P_i \mathbf{K}_i. \tag{11}$$

Then eqn (9) becomes

$$\mathbf{K}\mathbf{W} = \mathbf{M}, \tag{12}$$

where

$$\mathbf{M} = \left[\mathbf{m}_1, \cdots, \mathbf{m}_m\right], \tag{13}$$

$$\mathbf{M}_i = P_i \mu_i. \tag{14}$$

Under the assumption that $\mathbf{K}$ is nonsingular, the minimizing $\mathbf{W}$ is found to be

$$\mathbf{W}^* = \mathbf{K}^{-1}\mathbf{M}. \tag{15}$$

Thus, the linear discriminant function is

$$d_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{y}, i = 1, 2, \cdots, m \tag{16}$$

$$\mathbf{w}_i = \mathbf{K}^{-1}\mathbf{m}_i. \tag{17}$$

Each component of $\mathbf{y}$ of the LMS error procedure above is a function of input $\mathbf{x}$ as shown in eqn (5). This means that we want to represent $\mathbf{x}$ by a linear combination of nonlinear functions if the problem is not linearly separable. For example, let a discriminant function be

$$d(x) = (x - a)(x - b).$$

Then this can be expanded as

$$d(x) = \nu_1 x^2 + \nu_2 x + \nu_3, \nu_1 = 1, \nu_2 = -(a + b), \nu_3 = ab.$$

Therefore, the above nonlinear discriminant function now can be represented by the linear discriminant function:

$$d(x) = \nu_1\Phi_1 + \nu_2\Phi_2 + \nu_3\Phi_3,$$

$$\Phi_1(x) = x^2, \Phi_2(x) = x, \Phi_3(x) = 1.$$

$$d(x) = \mathbf{v}^T\Phi = \mathbf{w}^T\mathbf{y}.$$

In general, however, it is not possible to know the discriminant functions in advance and to tell what nonlinear functions make it linearly separable. If the input patterns are linearly separable, the simplest way to set $\mathbf{y}$ is

$$\mathbf{y} = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}. \tag{18}$$

Let us apply this idea of the linear discriminant function to an RBFN classifier. Then $\mathbf{y}$ is the vector of outputs of the middle layer and the kernel function of each middle layer neuron is supposed to make the input patterns linearly separable through its nonlinear transformation [eqn (5)]. And we obtain the weights between the middle and output layers by calculating the weights of the discriminant functions [eqn (17)]. In our interpretation each output neuron corresponds to a discriminant function. Therefore, the output neuron whose value (i.e., the value of the discriminant function) is the largest will become the result of the classification.

3.2.2. *By Bayesian decision theory (Method II).* Assume that the outputs of the middle layer form a normal distribution like the following:

$$p(\mathbf{x}|\omega_i) = \frac{1}{(2\pi)^{n/2}|\mathbf{C}_i|^{1/2}}\exp\left[-\frac{1}{2}(\mathbf{x}-\mathbf{m}_i)^T\mathbf{C}_i^{-1}(\mathbf{x}-\mathbf{m}_i)\right] \tag{19}$$

where

$$\mathbf{m}_i = E_i\{\mathbf{x}\}, \mathbf{C}_i = E_i\{(\mathbf{x}-\mathbf{m}_i)^T(\mathbf{x}-\mathbf{m}_i)\}. \tag{20}$$

Then the discriminant function for the class ω_i becomes

$$d_i(\mathbf{x}) = \ln[p(\mathbf{x}|\omega_i)P_i] = \ln[p(\mathbf{x}|\omega_i)] + \ln P_i.$$

In the above equation, P_i can be ignored if it is the same for all classes. Furthermore, if we assume that the covariance of each class is all the same ($\mathbf{C} = \mathbf{C}_i$), the discriminant function can be reduced to the linear one as follows:

$$d_i(\mathbf{x}) = \ln p(\mathbf{x}|\omega_i)$$

$$= -\tfrac{1}{2}(\mathbf{x}-\mathbf{m}_i)^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m}_i) - \ln(2\pi)^{n/2}|\mathbf{C}|^{1/2}$$

$$= \mathbf{x}^T\mathbf{C}^{-1}\mathbf{m}_i - \tfrac{1}{2}\mathbf{m}_i^T\mathbf{C}^{-1}\mathbf{m}_i - \tfrac{1}{2}\mathbf{x}^T\mathbf{C}^{-1}\mathbf{x} - \ln(2\pi)^{n/2}|\mathbf{C}|^{1/2}.$$

The constant terms $1/2\ \mathbf{x}^T\mathbf{C}^{-1}\mathbf{x}$ and $\ln(2\pi)^{n/2}|\mathbf{C}|^{1/2}$ can be removed since they are the same for all classes and hence do not affect the decision. Then the linear discriminant function becomes

$$d_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + w_{0i} \tag{21}$$

where

$$\mathbf{w}_i = \mathbf{C}^{-1}\mathbf{m}_i = \left(w_{1i}, w_{2i}, \cdots, w_{ni}\right)^T \tag{22}$$

$$w_{0i} = -\frac{1}{2}\mathbf{m}_i^T\mathbf{C}^{-1}\mathbf{m}_i. \tag{23}$$

Now we can calculate the weights $\mathbf{w}_i$ between the middle layer and the ith output neuron by eqns (22) and (23). The assumption that the covariances are all the same can be relaxed by the mixture normalization:

$$\mathbf{C} = \sum_{i=1}^{m} P_i\mathbf{C}_i \tag{24}$$

where P_i is the a priori probability of the class ω_i and it is usually set to $1/m$.

This Bayesian approach has a drawback that it is valid only under the assumption that the outputs of the middle layer form a normal distribution. However, this approach works well in practice if the number of the training patterns is reasonably large. This statement is supported by the next experiment.

4. EXPERIMENT

4.1. Unconstrained Handwritten Digit Recognition

In order to evaluate the efficiency of the proposed method and the performance of the obtained network we performed an experiment by using an unconstrained handwritten digit recognition. We first describe the performance of the obtained network in this subsection and then evaluate the effectiveness of the ideas proposed in this paper in the next subsection.

The database we used in our experiment consisted of digit images extracted from zip codes handwritten on U.S. mail envelopes (Suen, Nadal, Mai, Legault, & Lam, 1992). The database provides samples of totally unconstrained handwritten digits from a large number of writers. Each digit has 600 samples. Among them we used 400 samples for the training and 200 for the testing. Since the size of a sample image varies, we first normalized the size of the image and then extracted a feature called PDC (Peripheral Directional Contributivity) (Hagita, Naito, & Masuda, 1983) from the image. The dimension of the PDC feature extracted is 256. This means the number of the input neurons is 256 while that of the output neurons is 10.

We developed an RBFN classifier for this recognition problem by using APC-III described in Section 3.1 to determine the middle layer and using either Method I in Section 3.2.1 or Method II in Section 3.2.2 to calculate

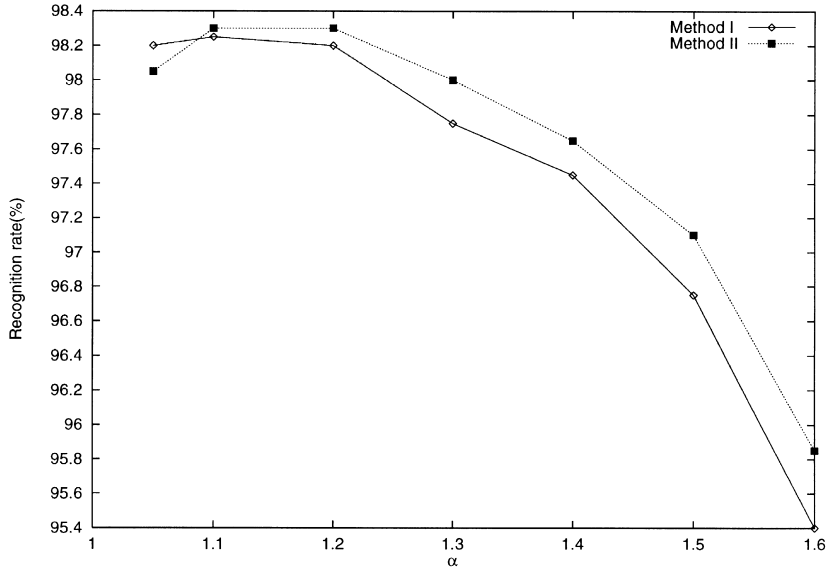

FIGURE 2. Recognition rates for testing patterns obtained by the two proposed methods when α changes.

the weights between the middle and the output layers. We recorded the recognition rates by changing the parameter α from $\alpha = 1.05$ since too many clusters would be generated for $\alpha < 1.05$. The experimental result shows that the recognition rates by both methods were almost the same. Method II performs better than Method I when α is larger than 1.1 (i.e., when there are a smaller number of clusters) although the difference is actually not significant. We obtained the best recognition rate 98.25% for the testing patterns when $\alpha = 1.1$ by the Method I and 98.30% when $\alpha = 1.2$ by the Method II (Figure 2). But which method should be used will depend on the problem to be handled.

The recognition rate for the testing patterns at $\alpha = 1.05$ is not better than that at $\alpha = 1.1$ in the case of Method II. This implies that the overtraining arises at $\alpha = 1.05$ since we generated too many clusters. What looks more interesting is that Method II, which requires that the outputs of the middle layer form a normal distribution, performs no worse than Method I. This indicates that the outputs of the middle layer form or almost form a normal distribution.

For comparison, Table 1 gives along with our results a list of the recognition rates obtained by the other research teams who used the same database as ours. These recognition rates are all for the testing patterns. To our understanding Lam and Legault in Table 1 used structural approaches and Mai used a statistical approach, while Kryzak and Lee used modified neural networks with the error back propagation algorithm (Suen et al., 1992; Lee, 1995).

It may be questioned whether the high recognition rates may be due to the feature we used. In order to clarify this point we performed another experiment in which we instead used a simple template matching algorithm with the same input feature. The experiment gave the recognition rates of 89.1% for the training patterns and 87.9% for the testing. This implies that the proposed method, not the feature used, primarily contributed to the good results.

4.2. Comparison with Other Methods to Determine the Middle Layer

α modifies the radius in APC-III algorithm and in turn the radius controls the number of clusters to be created. As the number of clusters increases, the performance increases to a certain level but drops thereafter. Figure 3 shows the number of clusters created as α changes. When we set $\alpha = 1.5$ in APC-III algorithm; for example, we had 147 clusters in total and on average 15 clusters for each digit. Therefore, in this case we created 147 neurons in the middle layer. Since we clustered the training patterns of each class separately, it created a different number of clusters for each digit as shown in Table 2.

TABLE 1
Comparison of our result with the recognition rates by others who used the same database

Method (Year)	Recognition Rate (%)
Nadal (1988)*	86.05
Lam (1988)*	93.10
Legault (1989)*	93.90
Mai (1990)*	92.95
Kryzak (1990)*	94.85
Lee (1995)	97.80
Method I	98.25
Method II	98.30

*According to Suen et al. (1992).

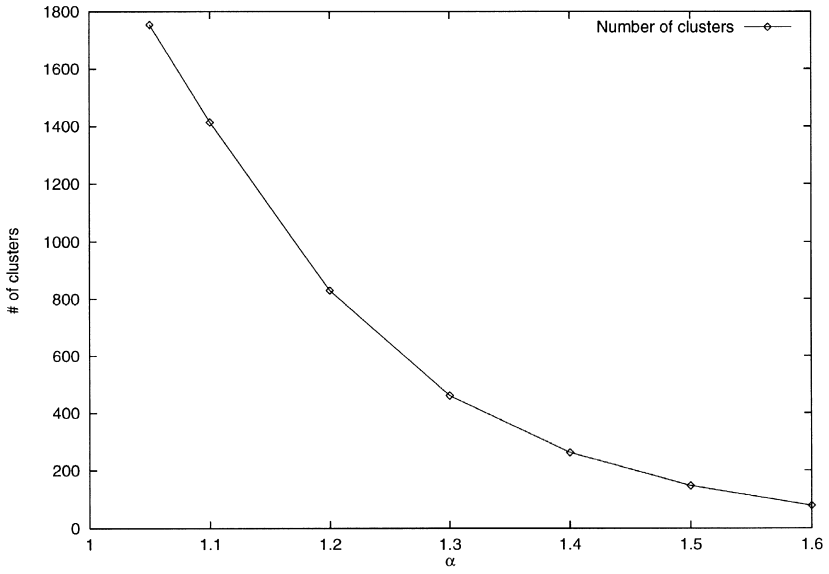

FIGURE 3. Number of clusters obtained by changing α in APC-III.

In order to evaluate the effectiveness of the proposed APC-III with regard to constructing the middle layer we compared the following three methods:

(A) APC-III ($\alpha = 1.5$; the average number of clusters for each digit is 15);
(B) random selection (randomly selects 15 patterns for each digit); and
(C) K-means clustering algorithm ($K = 15$; clusters the training patterns for each class separately).

We obtained the recognition rates given in Table 3 by using the three RBFN classifiers which were obtained by first deciding the middle layer using the above three methods and then calculating the weights using the Method I.

As seen in Table 3, the differences among the recognition rates using the three methods are relatively small, although the recognition rate by Method (A) is better than that by Method (B) but worse than that by Method (C). On the other hand, let us consider the time required to construct the middle layer. Method (B) requires no time to cluster since it just selects patterns randomly. Method (A) needs to go through the training patterns only once. Method (C) needs to go through the training patterns several times until it converges. As seen in the Table 3, APC-III actually obtained a better recognition rate than the random selection with a small additional time. We recorded the actual computation time when we experimented. We implemented the methods in C language and compiled them with the optimization level 3(+O3). Then we executed them on an HP 735 workstation and recorded the time spent in the system by the Unix command ''time''.

It should be noted that the times in the table are those spent just for clustering by those methods and that in a real situation all three methods need additional time for the trial and error efforts to find the optimal number of clusters.

TABLE 2
The number of clusters of each digit created by APC-III when $\alpha = 1.5$

Digit	Clusters	Digit	Clusters
0	10	5	16
1	5	6	12
2	21	7	10
3	13	8	29
4	21	9	10

4.3. Comparison with Other Methods to Determine the Weights

In order to evaluate the effectiveness of the proposed method to calculate the weights between the middle layer and the output layer, we compared the following four methods:

(A) Method I described in Section 3.2.1;
(B) Method II described in Section 3.2.2;
(C) gradient descent method that uses a fixed learning rate ($\eta = 0.02$); and
(D) gradient descent method that uses a varying learning rate ($\eta = 0.05/\sqrt{n}$ where n is the iteration number).

TABLE 3
The comparison of the recognition rates and the computation time by three methods to determine the middle layer when $\alpha = 1.5$

Method	Recognition Rate (%)	Time (Seconds)
APC-III	96.80	0.18
Random selection	95.70	0.00
K-means	97.05	7.02

In this comparison we used the same middle layer that was obtained by using APC-III ($\alpha = 1.5$). Methods (C) and (D) were implemented based on the algorithms described in (Luo, 1991).

As seen in Table 4, the recognition rate by Method (B) was the highest in spite of the fact that it requires the normal distribution of the outputs of the middle layer and the same covariance. The result by Method (A) was not as good as that by Method (B), even though it does not need any assumption. The results of (C) and (D) were obtained after 400 iterations with the best learning rates given above which were obtained through many experiments. It is one of the advantages of the proposed method that we don't need these time-consuming trial and error efforts to find the best learning rate. Although we do not claim that these are the best possible results that can be obtained by the gradient descent method, we are sure that the results represent most cases.

5. ANALYSIS

5.1. Time Complexity Analysis

Let us analyze the time complexity needed to compute the weights when we use either of the two methods described above. The basic computation requirements by both methods are the same. First, we need to go through the training patterns twice in order to obtain the means and the covariances. Namely, we need $O(P(Nn + nm))$ where P is the number of training patterns, N the number of the input layer neurons, n the number of the middle layer neurons, and m the number of the output layer neurons. $O(Nn)$ is the time to calculate the kernel functions of the middle layer and $O(nm)$ the time to calculate the output from the output layer. Further, it takes $O(n^3)$ to calculate the inverse of the covariance matrix and $O(mn^3)$ to multiply matrices to obtain the weights. In total the method needs the time of $O(P(Nn + nm) + mn^3)$ to calculate the weights.

On the other hand, the usual gradient descent method needs the time of $O(RP(Nn + nm))$, where R is the number of iterations. If P is large, the computation time increases as R becomes large.

If the number of the middle layer neurons (n) is relatively small compared with the number of the training patterns(P), we can ignore mn^3 in $O(P(Nn + nm) + mn^3)$.

TABLE 4
The comparison of the recognition rates by four methods to determine the weights

Method	Recognition Rate (%)
A	96.80
B	97.10
C	94.65
D	95.65

Then the only difference between them is R. Therefore, the proposed method will be more efficient in most cases.

We recorded the time executed by Methods (A) and (C) in Section 4.3 by the same way in Section 4.2. According to the times spent in the system there was no competition: the proposed method took 0.5 s while the gradient descent method 900 s for the first 400 iterations.

5.2. Comparison with Generalized Radial Basis Function Network

The proposed method is similar to GRBF (Haykin, 1994) in that both methods calculate the weights by matrix computation. GRBF computes the weights by the following equation.

$$\mathbf{w} = \left(\mathbf{G}^T\mathbf{G} + \lambda\mathbf{G}_0\right)^{-1}\mathbf{G}^T\mathbf{d} \tag{25}$$

where

$$\mathbf{G} = \begin{bmatrix} G(\mathbf{x}_1;\mathbf{c}_1) & G(\mathbf{x}_1;\mathbf{c}_2) & \cdots & G(\mathbf{x}_1;\mathbf{c}_n) \\ G(\mathbf{x}_2;\mathbf{c}_1) & G(\mathbf{x}_2;\mathbf{c}_2) & \cdots & G(\mathbf{x}_2;\mathbf{c}_n) \\ \vdots & \vdots & & \vdots \\ G(\mathbf{x}_p;\mathbf{c}_1) & G(\mathbf{x}_p;\mathbf{c}_2) & \cdots & G(\mathbf{x}_p;\mathbf{c}_n) \end{bmatrix} \tag{26}$$

$$\mathbf{G}_0 = \begin{bmatrix} G(\mathbf{c}_1;\mathbf{c}_1) & G(\mathbf{c}_1;\mathbf{c}_2) & \cdots & G(\mathbf{c}_1;\mathbf{c}_n) \\ G(\mathbf{c}_2;\mathbf{c}_1) & G(\mathbf{c}_2;\mathbf{c}_2) & \cdots & G(\mathbf{c}_2;\mathbf{c}_n) \\ \vdots & \vdots & & \vdots \\ G(\mathbf{c}_n;\mathbf{c}_1) & G(\mathbf{c}_n;\mathbf{c}_2) & \cdots & G(\mathbf{c}_n;\mathbf{c}_n) \end{bmatrix} \tag{27}$$

$$\mathbf{w} = \left[w_1, w_2, \cdots, w_n\right]^T \tag{28}$$

$$\mathbf{d} = \left[d_1, d_2, \cdots, d_p\right]^T. \tag{29}$$

In the above $G()$ is a Green's function and corresponds to the kernel function in eqn (1), $\mathbf{d}$ is the desired response vector, $\mathbf{w}$ is the weight vector and λ is the regularization parameter. When λ approaches zero, the weight vector $\mathbf{w}$ converges to the pseudo inverse (minimum norm) solution to the overdetermined least-squares data-fitting problem, as shown by

$$\mathbf{w} = \mathbf{G}^+\mathbf{d}, \lambda = 0 \tag{30}$$

where $\mathbf{G}^+$ is the pseudo inverse of matrix $\mathbf{G}$, that is,

$$\mathbf{G}^+ = (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T. \tag{31}$$

Although, in theory, the quantity $(\mathbf{G}^T\mathbf{G})^{-1}$ exists, significant numerical difficulties may occur during computing this inverse when $\mathbf{G}^T\mathbf{G}$ is nearly singular. In addition, the process of forming $\mathbf{G}^T\mathbf{G}$, for large $\mathbf{G}$ (note $\mathbf{G}$ is $P \times n$), is computationally expensive. The numerical sensitives can be avoided using the singular value decomposition. But the singular value decomposition produces a $P \times P$ matrix which is larger than $P \times n$.

The proposed method uses a matrix of the size smaller than $(n + 1) \times (n + 1)$ which can be easily handled in any computer.

6. CONCLUSION

We presented an efficient method to construct an RBFN classifier whose performance was shown to be better than the existing classification methods at least in the application described in this paper. The method consists of two parts: one determines the middle layer and the other calculates the weights between the middle and the output layers of an RBFN.

The first part of our method, called APC-III, clusters the training patterns class by class. In APC-III the radius of the clusters is determined based on the distribution of the training patterns before the clustering. And the clustering procedure needs to go through the training patterns only once. This makes APC-III algorithm faster than the usual K-means or SOFM clustering algorithm which usually has to go through the entire training patterns many times. The proposed method creates a middle layer neuron for each of the clusters so generated. The width of the kernel function of each neuron is also determined based on the distribution of the training patterns.

The usual gradient descent method which is widely used to calculate the weights has well-known drawbacks such as slow learning, local minima and sometimes being unable to converge. There is a method called GRBF which is similar to ours, but GRBF requires large matrix computation and the computation may not be feasible in a computer with small memory. On the other hand, the proposed method uses a smaller matrix to compute the weights, which makes our method more practical.

The proposed method uses a statistical approach called linear discriminant function to determine the weights. It is possible by using our approach to obtain the optimal weights analytically and efficiently. We presented two specific methods to obtain the linear discriminant functions. One uses LMS error minimization procedure and the other derives the linear discriminant functions under the assumption that the training patterns form a normal distribution and all the covariance matrices are the same.

We performed an experiment using unconstrained handwritten digit recognition in order to evaluate the effectiveness of our method. As a result, we first confirmed that the recognition rate obtained by the RBFN classifier which was developed by our method was better than the best result of the previous experiments which had used the same database. Second, the performance of APC-III algorithm is better than the random selection algorithm and K-means clustering algorithm. On the other hand, the time required by APC-III is between that of the random selection and by K-means. Third, it turned out that our method using linear discriminant function to determine the weights performs better and extremely faster than the standard gradient descent method.

In summary, we presented an efficient and effective method to construct an RBFN classifier by applying a statistical approach to train the RBFN classifier and confirmed through an experiment using well-known unconstrained handwritten digit recognition problems that the proposed method is efficient and the resulted network performs well. The proposed method will make it possible for an RBFN classifier to be more widely and easily used.

REFERENCES

Asim Roy, S. G., & Miranda, R. (1995). An algorithm to generate radial basis function (RBF)-like nets for classification problems. *Neural Networks*, *8*(2), 179–201.

Broomhead, D. S., & Lowe, D. (1988). Multivariable function interpolation and adaptive networks. *Complex Systems*, *2*, 321–355.

Chang, E. I., & Lippmann, R. P. (1993). A boundary hunting radial basis function classifier which allocates centers constructively. *Advances in Neural Information Processing Systems*, *5*, 139–146.

Devijver, P., & Kittler, J. (1982). *Pattern recognition: a statistical approach*. Englewood Cliffs, N.J.: Prentice-Hall.

Hagita, N., Naito, S., & Masuda, I. (1983). Handprinted Chinese characters recognition by peripheral direction contributivity feature. *Transactions of the IEICE D*, *66*(10), 1185–1192. (in Japanese).

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. New York: MacMillan.

Hwang, Y.-S., & Bang, S.-Y. (1994). A neural network model APC-III and its application to unconstrained handwritten digit recognition. In *Proceedings of International Conference on Neural Information Processing* (pp. 1500–1505). Seoul: Korean Association for Intelligent Information Systems.

Lee, S. W. (1995). Multilayer cluster neural network for totally unconstrained handwritten numeral recognition. *Neural Networks*, *8*(5), 783–792.

Luo, Z.-Q. (1991). On the convergence of the lms algorithm with adaptive learning rate for linear feedforward networks. *Neural Computation*, *2*(3), 226–245.

Monica Bianchini, P. F., & Gori, M. (1995). Learning without local minima in radial basis function networks. *IEEE Transactions on Neural Networks*, *6*(3), 749–756.

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

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

Park, J., & Sandberg, I. W. (1993). Approximation and radial-basis-function networks. *Neural Computation*, *5*, 305–316.

Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, *78*, 1481–1497.

Reilly, D., Cooper, L., & Elbaum, C. (1982). A neural model for category learning. *Biological Cybernetics*, *45*, 35–41.

S. Chen, C. F. C., & Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, *2*(2), 302–309.

Suen, C., Nadal, C., Mai, T., Legault, R., & Lam, L. (1992). Computer recognition of unconstrained handwritten numerals. *Proceedings of the IEEE*, *80*(7), 1162–1189.