A Novel Training Scheme for Multilayered Perceptrons to Realize Proper Generalization and Incremental Learning

Debrup Chakraborty and Nikhil R. Pal, *Senior Member, IEEE*

Abstract—The response of a multilayered perceptron (MLP) network on points which are far away from the boundary of its training data is generally never reliable. Ideally a network should not respond to data points which lie far away from the boundary of its training data. We propose a new training scheme for MLPs as classifiers, which ensures this. Our training scheme involves training subnets for each class present in the training data. Each subnet can decide whether a data point belongs to a certain class or not. Training each subnet requires data from the class which the subnet represents along with some points outside the boundary of that class. For this purpose we propose an easy but approximate method to generate points outside the boundary of a pattern class. The trained subnets are then merged to solve the multiclass classification problem. We show through simulations that an MLP trained by our method does not respond to points which lies outside the boundary of its training sample. Also, our network can deal with overlapped classes in a better manner. In addition, this scheme enables incremental training of an MLP, i.e., the MLP can learn new knowledge without forgetting the old knowledge.

Index Terms—Class boundary, generalization, incremental learning, minimal spanning tree, multilayered perceptrons.

I. INTRODUCTION

IN THE PAST FEW years multilayered perceptrons have been used to solve numerous problems in a variety of domains. It has been proved that MLPs can act as universal approximators for a large class of nonlinear functions [4], further the learning and generalization properties of these networks have found many diverse applications. But MLPs are not interpretable, i.e., one cannot retrieve the meanings of the learned parameters of an MLP by any easy means. The unreadability of this kind of network presents certain limitations on them which are often undermined. The generalization ability of MLPs has also been over estimated. People generally rely on the output of an MLP for any data point without paying due consideration on the position of the data point with respect to the training data. Modern methods of training MLPs involve strict phases of training, validation and testing. But one generally depends on the data set at hand for these phases and good performance can be guaranteed only on the available data. The response of an MLP for a data point which lies well outside the “boundary”

of the training data is usually erratic, and hence, not reliable. For example, even when a test data point is far away from the convex hull of the training data, an MLP will produce some output, sometimes this response could be very high and can be totally useless and dangerous too. As MLPs are being used in critical areas like medical diagnosis, non destructive testing (NDT) [15], this may lead to serious problems. We will illustrate this issue later with compelling examples.

MacKay [7]–[9] addresses more or less the same problem of designing classifiers which are intended to give low response in areas with sparse training data. In [9] he elaborated that a network (classifier) designed with the most probable parameter (weight) vector $\mathbf{w}_{MP}$ (obtained using a training data D) will typically give more extreme, unrepresentative and overconfident output in areas with sparse training data—and this is *not* desirable. Hence, he has worked out a strategy to moderate the output of a network with parameters $\mathbf{w}_{MP}$. His results show that the network with moderated output performs in a better manner. The moderated output is similar to the most probable output in the regions where the training data are dense. On the other hand, where the training data are sparse, the moderated output becomes significantly less certain than the most probable output. This is certainly an interesting approach. But, since MacKay’s training scheme [9] does not consider the input space having no data, the moderated output could be high even in areas far away from the training data. The results in [9] reveals that this is indeed the case for a simple two class problem. Moreover, when a crisp decision is taken, moderated output and most probable output are the same.

Another very important issue concerned with MLP is how to learn new knowledge without forgetting old knowledge. An MLP cannot retain its old knowledge when it is retrained by a new data set. Also, there exists no easy means to augment an MLP to incorporate new knowledge. This issue of incremental learning has been addressed by a few researchers in [2], [3], [12].

In this paper, we would like to address both of the above issues. A network should learn only as much as dictated by the training data. When an MLP is used as a classifier, this means that for test data points which are away from the training data points, the class response for every class should be very low. We call such a generalization as “strict generalization.” The other issue deals with incremental learning. Given a trained network, if some new training data points come for which the current class responses for all classes are very low, then we should be able to augment the network so that it can learn the new data without

Manuscript received September 12, 2001; revised May 20, 2002 and September 30, 2002.

The authors are with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Calcutta 700108, India (e-mail: debrup_r@isical.ac.in; nikhil@isical.ac.in).

Digital Object Identifier 10.1109/TNN.2002.806953

Fig. 1. (a) Scatterplot of Scattered1. (b) Generalization by an MLP on Scattered1.

forgetting the old ones. Note that, the old training data are not available any more.

In this paper, we begin with some discussions on the boundary of a pattern class. Then we show by examples that the behavior of an MLP outside the boundary of the training data is erratic when it is used as a classifier. We also present a new training scheme which does generalization in a stricter sense. The proposed training scheme is also capable of incremental learning, i.e., a trained network can easily be augmented to incorporate new knowledge. Although in this paper we concentrate on MLP as a classifier, but with some changes the methodology may be made applicable for MLPs used for other applications like function approximation, system identification, etc., which we do not consider here.

II. BOUNDARY OF A PATTERN CLASS

The data points from a pattern class usually will be generated following some distribution. The distribution function can give information about the spatial boundary of the pattern class. But in most real life cases such distribution functions are neither known *a priori* nor are easy to estimate from the data points of the different classes. One way to determine the class boundary may be to estimate the sampling window from which the points of that class are generated. The sampling window is generally defined as the convex compact support set of the distribution. The convex hull of the points in a class is considered to be a good estimate of its sampling window [11], [14]. If the points of a class are generated from a convex support set and form a cluster in the input space, then the convex hull of the points in the training data may be a good estimate of its boundary. But, in real life data sets, points from a class can take any shape—it may even form a number of separate clusters in the input space. And in these cases finding boundary of a pattern class poses a serious problem. In a recent work [13], Schölkopf *et al.* attempted to find the support of a multidimensional distribution using the support vector machine framework. This method can be suitably used to determine the boundary of a class, but here we take a simpler and intuitive definition of boundary.

We would intuitively introduce the concept of “boundary” in terms of neighborhood. Suppose C is a set of points which

belongs to a class. Any point x is considered to be a point outside the boundary of the class if the distance of x from its nearest neighbor in C is greater than a threshold th . The threshold th will depend on the shape and density of the points in the class represented by C . We continue with this naive definition of a boundary and in Section IV we shall devise an algorithm to find points outside the boundary of a pattern class.

III. IMPROPER BEHAVIOR OF MLP OUTSIDE THE BOUNDARY OF THE TRAINING SAMPLE

With the naive definition of “boundary” in mind, we show by examples that the behavior of an MLP for a point which lies outside the “boundary” of its training sample is not predictable. For this purpose here we use three representative data sets, all in R^2 . The data sets are named **Scattered1**, **Two-Shell** and **Dish-Shell**. Scattered1 has three reasonably well separated classes with 100 points in each class. The scatterplot of Scattered1 is shown in Fig. 1(a) where *, + and o represent the points from the three different classes. Both Two-Shell and Dish-Shell have two classes with 500 points in each class. Scatterplots of Two-Shell and Dish-Shell are shown in Figs. 2(a) and 3(a), respectively.

An MLP with 20 nodes in the hidden layer with sigmoidal activation functions was used to classify all three data sets. There was no misclassifications on the training data for each of these data sets. This appears very encouraging. Let us now see, in each case how the network generalizes to points outside the “boundary” of the classes. For this, we computed the smallest rectangle containing the data, with its sides parallel to the axes of coordinates. Then we increased each edge by 5% on each side. Next we generated an array of 256×256 points uniformly covering the entire rectangle. And used these 65 536 points as our test data. We consider a point to be classified to class k if the output of the k th output node is *more than 0.8* and the output of all other nodes is *less than 0.2*. For Scattered1 the generalization by a trained MLP is shown in Fig. 1(b). The areas marked by the two shades of gray and black represent the three classes and white area represents the points for which the MLP could not make any decision. We draw the attention of the reader to the U-shaped black patch at the bottom right corner suggesting the class represented by * in Fig. 1(a). This

Fig. 2. (a) Scatterplot of Two-Shell. (b) Generalization by a trained MLP.

Fig. 3. (a) Scatterplot of Dish-Shell. (b) Generalization by a trained MLP.

is surely a very poor generalization, although the training set resulted in zero misclassification.

In Figs. 2(b) and 3(b), the generalization by a trained MLP on Two-Shell and Dish-Shell are shown. Here, black and white represent the two classes and gray represents the points for which no decision was made by the MLP. In these cases too the generalizations are not desirable. Note that, here we use a conservative approach to decide the class. Usually, a data point $\mathbf{x}$ is classified to class k if the k th output node have the maximum response (ignoring how strong or weak the maximum response is or what the response to the other classes are). So, every point gets classified to one of the classes.

The above results clearly demonstrate that an MLP performs quite well on the training data and also on the points which lie inside the “boundary” of each pattern class. But its response on points outside the “boundary” of the training data may not follow any specific behavior—often it behaves in a strange manner!

IV. NEW TRAINING SCHEME

Here we discuss a new training scheme which takes care of the problems discussed above. Let us consider a classification problem of a data set $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \subset R^p$. Suppose X consists of k classes and $X = \cup_{i=1}^k S_i$, $S_i \cap S_l = \phi$, $i \neq l$,

such that $\mathbf{x}_j \in S_i$ is from class i . We assume that we have a mechanism to define the boundary of each S_i .

Let Ω_{small} be the smallest hypercube, which bounds X . By increasing each edge of Ω_{small} by $l\%$ on all sides, we inflate Ω_{small} to Ω , we call Ω as the inflated hyperbox of X . Our scheme will guarantee strict generalization on all test data points which lie within Ω . Thus, Ω is the space from which the data points are expected to come. We may have prior knowledge of Ω or we can compute Ω by inflating Ω_{small} to some extent. For our simulations we have used a specific inflation rate ($l = 5\%$). Now for each pattern class S_i , let Y_i denote a set of points generated uniformly within Ω , but outside the boundary of points in S_i . For the data set X we construct k training sets T_i , $i = 1, 2, \dots, k$; where T_i includes points in S_i with label 1 and points in Y_i with label 0.

We train k MLPs with these T_i s, $i = 1, 2, \dots, k$; we call these networks M_i , $i = 1, \dots, k$. The restriction on the architecture of each M_i is that its input layer consists of p nodes and the output layer consists of only one node. Thus, each M_i learns a 2-class problem and can detect whether a data point is inside the boundary of class i or not.

These k MLPs can be merged together, as shown in Fig. 4, to form a single network $\mathcal{M}$ which solves the required k class problem. The merging of two subnets to identify two different classes is called simple merging. The p dimensional input is fed

Fig. 4. Simple merging of k trained MLPs.TABLE I
ALGORITHM TRAIN**Algorithm TRAIN**

Let $X = \cup_{i=1}^k S_i$, S_i be the set of points in class i .
 for $i = 1$ to k
 Generate Y_i (the points outside the boundary of S_i)
 $T_i = S_i \cup Y_i$ (S_i with class label 1 and Y_i with 0)
 Train subnet M_i (architecture $p : n_i : 1$) with T_i .
 end for
 Simple merge $M_1, M_2, \dots, M_k$ to get M
 end

to each M_i , if it belongs to class q , $q \in \{1, 2, \dots, k\}$, then the response of M_q would be high and those of the rest would be low (Fig. 4).

If the training data from different classes do not overlap, the network can make unambiguous decision. But if the training data from more than one class overlap the network will be able to signal that. We shall discuss this later. We provide a schematic description of the entire training process in algorithm TRAIN in Table I. We use $(m:n:l)$ to indicate an MLP with m input nodes, n hidden and l output nodes.

After the training is over we get a composite network, which, given a test input $\mathbf{x} \in R^p$ with unknown class label, will produce a k -dimensional output vector. The next issue is how to interpret the output of the composite network M . To understand this, we need to consider the structure of the training data. Suppose there is no overlap between the training data from various classes. Thus, for each training and test data points *at most* one of the subnets will produce high response. And consequently either the class label of $\mathbf{x}$ will be unambiguous or no decision can be made on it. If the training data from classes i and j are overlapped, and if the test data point $\mathbf{x} \in R^p$ is from the overlapped region, then output of nodes corresponding to i and j of the composite network will be high. In such a case we should not assign $\mathbf{x}$ to one of i and j , but should make a decision that $\mathbf{x}$ can be in either

of the two classes. This gives an additional information about $\mathbf{x}$, that it probably comes from an overlapped region.

Thus, by properly interpreting the output of our network we can get more knowledge about the points in an overlapped region, and also about the points on which our network may give wrong decisions. We say that a point belongs to class l if the l th output o_l of the composite network is greater than 0.8 and all other outputs are less than 0.2. If all output units give a response less than 0.2, (i.e., $o_j < 0.2, \forall j$) then we do not make any decision about the point. If $o_j > 0.8$ for $j = i_1, i_2, \dots, i_l$ and $o_j < 0.2$ for all other classes, then $\mathbf{x}$ is probably in an overlapped area of the classes $i_1, i_2, \dots, i_l$. In all other cases, responses of one or more classes are high, but none of them is high enough. In such cases we make a soft decision. We take the class label corresponding to the output units giving the maximum response. But in this case our network warns that this decision is a soft decision, and it may be wrong. The test procedure is summarized in the algorithm TEST in Table II.

In conventional training of an MLP, whenever there are overlapped classes, the MLP will surely produce misclassifications. If an MLP learns overlapped classes without any misclassification, then it is a sort of overfitting on the data, which will produce bad generalization. The usual training of MLP is not designed in such a way that the outputs can be interpreted to detect whether a test point belongs to an area of overlap. We provide some simulation results in the results section in support of this. Whereas in case of our network we can obtain multiple class labels for a data point which lies in an area of overlap. Also when the class response for any class is not significantly high, our network gives a soft decision. This is an additional advantage of our method over the MLP. Next we discuss how our method can be used for incremental learning.

A. Incremental Learning

The proposed training scheme can be easily used to augment a trained network with a new set of training data. We develop the method for incremental learning under the assumption that we have prior knowledge about the input space Ω , from which new training data may come. This is really not a strong assump-

TABLE II
ALGORITHM TEST*Algorithm* **TEST**($\mathbf{x}, M$)

```

Let  $M$  contain  $k$  output nodes;
Input  $\mathbf{x}$  to  $M$ ;
Let  $U = \{1, 2, \dots, k\}$  (The set of classes);
Let  $o_i, i \in U$  be the output of the  $i^{\text{th}}$  output node;
Let  $S = \{i : o_i > 0.8, \forall i \in U\}$ ;
If  $o_j \leq 0.2, \forall j \in U - S$  Then
  if  $|S| = 0$  Then
    class for  $\mathbf{x}$  cannot be decided;
  else if  $|S| = 1$  (let  $S = \{c_1\}$ ) Then
    unambiguous decision,  $\mathbf{x}$  belongs to class  $c_1$ ;
  else (let  $S = \{c_1, c_2, \dots, c_l\}, 1 < l \leq k$ )
    ambiguous decision,  $\mathbf{x}$  can belong to any of the
    classes  $c_1, c_2, \dots, c_l$ ;
  end if
else
   $j = \text{Argmax}_{k \in U} \{o_k\}$ ;
  Soft decision,  $\mathbf{x}$  belongs to class  $j$ ;
endif
end

```

tion as one can start with a big enough Ω to account for future augmentation of the network.

Suppose, initially we have a data set $X_{\text{init}} = \cup_{i=1}^k S_i$, which represents k classes. Let us call the k classes as $1, 2, \dots, k$, and S_i contains the points in the i^{th} class. We train k subnets, as discussed in the previous section to solve the k class problem. We name the subnets as $M_{\text{init}}^i, i = 1, 2, \dots, k$. Then we merge these subnets by a simple merge to form a new network $\mathcal{M}_{\text{init}}$. $\mathcal{M}_{\text{init}}$ incorporates the knowledge in X_{init} .

Later suppose we obtain a new data set X_{new} which represents $m = k + n$ classes, where the first k classes are the same as the k classes in X_{init} and the other n classes are new ones. Let X_{new}^O denote the set of points in X_{new} which belongs to the first k classes and X_{new}^N denote the set of points which belongs to the new n classes, so $X_{\text{new}} = X_{\text{new}}^O \cup X_{\text{new}}^N$ and $X_{\text{new}}^O \cap X_{\text{new}}^N = \phi$. Usually the points in X_{new}^N will lie outside the boundary of the k classes in X_{init} . If so, then because of our specialized training, $\mathcal{M}_{\text{init}}$ will not produce high response for the points in X_{new}^N . But, if $\mathcal{M}_{\text{init}}$ produces high response for some points in X_{new}^N , then there is some overlap between the previous and the new classes. Irrespective of whether $\mathcal{M}_{\text{init}}$ produces high response for points in X_{new}^N or not, for each of these n new classes we construct n separate MLPs $M_{\text{additional}}^i, i = k+1, k+2, \dots, k+n$, as discussed before and merge them by simple merge with $\mathcal{M}_{\text{init}}$ to produce a new network $\mathcal{M}_1$. Hence, $\mathcal{M}_1$ should be able to classify points in $X_{\text{init}} \cup X_{\text{new}}^N$ into $m = k + n$ classes.

Fig. 5. Compound merge of two trained MLPs.

Although the points in X_{new}^O represent the existing k classes, $\mathcal{M}_1$ may not correctly classify all points in X_{new}^O . If it can, then we are done and $\mathcal{M}_1$ is the desired network. Otherwise, we need to augment $\mathcal{M}_1$.

Let $X_{\text{new}}^O = X_{\text{new}}^C \cup X_{\text{new}}^R$, where X_{new}^C is the set of points which are correctly classified by $\mathcal{M}_1$, and X_{new}^R contains the points which are not classified correctly by $\mathcal{M}_1$. Note that, here by correct classification of points in X_{new}^C we mean that the network $\mathcal{M}_{\text{init}}$ produces high response for all points in X_{new}^C in one or more classes which includes the correct class. The points in X_{new}^C need not be considered for further training. The points in X_{new}^R are either outside the boundary of all the classes present in X_{init} or there may be points representing class i but lying within the boundary of $S_j, j \neq i$. The latter case is a case of overlapped classes. In both cases points in X_{new}^R contain new information which is to be augmented with the initial network.

Suppose the set X_{new}^R represents $q \leq k$ classes. So we train q subnets, $M_{\text{new}}^i (i = i_1, i_2, \dots, i_q), i_j$ denotes one of the q classes. $M_{\text{new}}^{i_j}$ is trained with the points which belong to class i_j in X_{new}^R along with points generated outside the “boundary” of the respective class. Note that, each of these q subnets detects (represents) one of the classes already represented by some subnet in $\mathcal{M}_{\text{init}}$ (and also in $\mathcal{M}_1$). Let us denote the initial subnets of $\mathcal{M}_{\text{init}}$ by $M_{\text{init}}^i (i = 1, 2, \dots, k)$. Subnet M_{init}^l is merged with $M_{\text{new}}^{i_j}$, if $l = i_j$, by a compound merge (Fig. 5). In a compound merge the outputs of M_{init}^l and $M_{\text{new}}^{i_j} (i_j = l)$ are combined by an OR operator to get the final output. For our case we use max as the OR operator. The network obtained by the compound merging of the subnets of $\mathcal{M}_1$ and $M_{\text{new}}^{i_j}, j = 1, 2, \dots, q$, is denoted by $\mathcal{M}_{\text{final}}$. $\mathcal{M}_{\text{final}}$ should be able to classify all points in $X_{\text{init}} \cup X_{\text{new}}$.

For better clarity, we summarize the above method of augmenting a trained network by the pseudocode AUGMENT in Table III. The algorithm AUGMENT takes as input a pretrained network M_{init} which represents k classes and have k subnets $M_{\text{init}}^i, i = 1, \dots, k$, and a new data set X_{new} . $X_{\text{new}} = X_{\text{new}}^O \cup$

TABLE III
ALGORITHM AUGMENT

Algorithm AUGMENT(M_{init}, X_{new})

$X_{new}^N = S_{k+1} \cup S_{k+2} \dots \cup S_{k+m}$;
(where S_{k+i} , contains points from class $k+i$)
for $i = k+1$ to $k+m$
 train subnet $M_{additional}^i$ with S_i and
 some points outside the boundary of S_i ;
end for
simple merge $M_{init}^1, M_{init}^2, \dots, M_{init}^k$ and
 $M_{additional}^{k+1}, \dots, M_{additional}^{k+m}$ to get $\mathcal{M}_1$;

$l(\mathbf{x}_i)$ be the class label of $\mathbf{x}_i$;
 $U = \{1, 2, \dots, k\}$ (the set of initial classes)
 $I = \phi$;
 $S_i = \phi, \forall i \in U$;
for all $\mathbf{x}_i \in X_{new}^O$;
 if **TEST**($\mathcal{M}_1, \mathbf{x}_i$) cannot decide¹ $\mathbf{x}_i$
 put $\mathbf{x}_i$ in $S_{l(\mathbf{x}_i)}$;
 if $l(\mathbf{x}_i) \notin I$
 put $l(\mathbf{x}_i)$ in I ;
 end if;
 end if;
end for;
if $I = \phi$ call $\mathcal{M}_1$ as M_{final} and stop;
for all $i \in I$
 train subnet M_{new}^i with S_i and points
 outside the boundary of S_i ;
end for;
 $\forall i \in I$ Compound merge M_{new}^i with subnet
 M_{init}^i of $\mathcal{M}_1$ to get M_{final}^i ;
end

¹cannot decide here includes the possibilities of “no decision”, “soft decision”,
“incorrect unambiguous decision”
or “ambiguous decision which does not contain the correct class of $\mathbf{x}_i$ ”.

X_{new}^N , as discussed above. The algorithm AUGMENT creates the augmented network M_{final} .

The crucial step in the methodology described is generation of the training set for each class which contains the given training points of that class along with points which lie outside the boundary of that class. Thus, the success of the scheme will depend on the generation of points outside the boundary of a given pattern class. In Section IV-B we define what we mean by the boundary of a class and a method to generate points outside the boundary of a pattern class.

B. Generating Points Outside the Boundary of a Pattern Class

Here we present a simple and approximate scheme to generate points outside the boundary of a pattern class. Let Ω to be the inflated hyperbox which bounds the data points in X (as discussed earlier). We define δ_{avg}^i as the average edge length of

a minimal spanning tree (MST) which spans the points present in S_i , i.e., the training points for class i .

Definition: Boundary of a pattern class.

A point $\mathbf{x}$ is outside the boundary of the pattern class C if the nearest neighbor of $\mathbf{x}$ in S_C is at a distance greater than $\alpha\delta_{avg}^C$, where $\alpha \geq 1$ is a predefined constant. Otherwise, it is inside the boundary of C .

The average edge length of an MST generally gives us some information about the sparseness of the data points present in the pattern class. If S_i forms a nice cluster then clearly δ_{avg}^i will give a good idea about the interpoint distances. Even if S_i forms a few well separated clusters, and if the density of points in each cluster is almost the same then also δ_{avg}^i will give information about the sparseness of data points in each cluster. Thus, δ_{avg}^i will not be much affected by the number of clusters and the “distance” between the clusters present in S_i , for large $|S_i|$. We state this more clearly in the next two lemmas.

Let S_i consist of k clusters, $S_i^l, l = 1, 2, \dots, k$. Let $\mathcal{T}_i$ be an MST on S_i and let $\mathcal{T}_i^l, l = 1, \dots, k$, be the subsets of the edges in $\mathcal{T}_i$ that spans the points in cluster S_i^l . We assume that clusters are well separated, $|S_i|$ is large and number of clusters are much smaller than $|S_i|$. Under these assumptions we can safely say that $\mathcal{T}_i$ has only $k-1$ edges such that the two end points are in two different clusters. The remaining $|\mathcal{T}_i| - (k-1)$ edges, i.e., $|S_i| - k$ edges have both end points belonging to the same cluster. Then we can have the following lemma.

Lemma 1: $\mathcal{T}_i^l$ is the MST of points in S_i^l .

Proof: If not, let τ_i^l be the MST on S_i^l . From the MST $\mathcal{T}_i$ we remove the subtree $\mathcal{T}_i^l$ and add τ_i^l to get $\mathcal{T}_i^{new}$. Thus

$$\mathcal{T}_i^{new} = (\mathcal{T}_i - \mathcal{T}_i^l) \cup \tau_i^l.$$

Now, $\mathcal{T}_i^{new}$ is a spanning tree of S_i . Also, if $W(T)$ denote the total weight of a tree T . Then by construction

$$W(\mathcal{T}_i^{new}) < W(\mathcal{T}_i)$$

as $W(\tau_i^l) < W(\mathcal{T}_i^l)$. Hence, $\mathcal{T}_i$ is not an MST of S_i . Which is a contradiction.

Thus each $\mathcal{T}_i^l$ is a MST of the points in S_i^l . Let δ_{avg}^i be the average edge length of the MST $\mathcal{T}_i$ then we can prove the following lemma.

Lemma 2: For large $|S_i|$, δ_{avg}^i is independent of the length of the MST edges connecting two different clusters.

Proof: Let the average edge length of $\mathcal{T}_i^l$ be δ_l , and let the lengths of the $k-1$ edges which connect points from two different clusters be $e_j, j = 1, \dots, k-1$. Let n_l be the number of points present in cluster $l, l = 1, 2, \dots, k-1$. Hence, the average edge length of $\mathcal{T}_i$ would be

$$\delta_{avg}^i = \frac{1}{|S_i| - 1} \left\{ \sum_{l=1}^k (n_l - 1) \delta_l + \sum_{j=1}^{k-1} e_j \right\}. \quad (1)$$

Let δ^* be the average edge length of all the edges in $\mathcal{T}_i$ which connects points belonging to the same cluster, i.e., δ^* is the average edge length of all the MSTs $\mathcal{T}_i^l, l = 1, 2, \dots, k$. Then

$$\delta^* = \frac{\sum_{l=1}^k (n_l - 1) \delta_l}{\sum_{l=1}^k (n_l - 1)} \quad (2)$$

$$= \frac{\sum_{l=1}^k (n_l - 1) \delta_l}{|S_i| - k}. \quad (3)$$

Hence, we get

$$\delta_{\text{avg}}^i = \frac{1}{|S_i| - 1} \left\{ (|S_i| - k) \delta^* + \sum_{j=1}^{k-1} e_j \right\} \quad (4)$$

$$= \frac{\delta^* \left(1 - \frac{k}{|S_i|}\right)}{\left(1 - \frac{1}{|S_i|}\right)} + \frac{\sum_{j=1}^{k-1} e_j}{|S_i| - 1}. \quad (5)$$

Here, $k \ll |S_i|$ and $\sum_{j=1}^{k-1} e_j$ is finite. Thus,

$$\lim_{|S_i| \rightarrow \infty} \delta_{\text{avg}}^i = \delta^*. \quad (6)$$

Hence, for large $|S_i|$ the average edge length of $\mathcal{T}_i$ depends on the average length of the edges in each MST of the clusters present in S_i .

Again, if we assume that the density of the points in each of the k clusters is same and hence the average edge lengths of the MSTs of the points in each cluster is the same (say, δ). Then we have

$$\delta^* = \delta$$

thus, δ_{avg}^i is equal to the average edge length of each MST. Hence, for large $|S_i|$, the average edge length of an MST is independent of the number of clusters present in the set. So δ_{avg}^i can be used as a good measure of sparseness of the data even if S_i contains clusters in it.

The method of generation of points outside the boundary of a pattern class is summarized in procedure GENERATE in Table IV. GENERATE takes as input the inflated hyperbox (Ω) of the total data and a set of points S_C in the pattern class C .

The multiplier α controls the tightness of the boundary, i.e., a smaller value of α will yield a tighter boundary than a larger value of α . To demonstrate the effect of α on the boundary we use a new data set called **Square**, which contains 500 points in R^2 , generated randomly over a square. The scatterplot of Square is shown in Fig. 6. Fig. 7 shows the points generated outside the boundary of Square for different values of α . Fig. 8 shows the points generated outside the boundary of Dish-Shell, assuming that the entire data are from one class (scatterplot in Fig. 3). Figs. 7 and 8 clearly exhibit that the algorithm can capture the concept of the boundary and its performance is quite good for $\alpha \in [1.5, 2.5]$. So in our simulations we use $\alpha = 2$.

V. RESULTS

We present here results on five data sets. The data sets are named **Dish-Shell**, **3-D-Elongated**, **Cone-Torus**, **Sat-Image** and **Scattered**. The first two data sets have well separated classes and we use them to demonstrate the generalization ability of our scheme. Cone-Torus and Sat-Image have overlapped classes. These two data sets have been previously used by many researchers to evaluate different classifiers [5]. Hence, using Cone-Torus and Sat-Image we compare the performance of our method with that of a normal MLP. These results are presented in Section V-A. In Section V-B we use the data set

TABLE IV
ALGORITHM GENERATE

Algorithm GENERATE (Ω , S_C)

```

begin
  Select  $\alpha$ ;
  Compute MST of the points in  $S_C$ ;
  Compute  $\delta_{\text{avg}}^C$ ;
  while(count < nCount)
    Randomly generate  $\mathbf{x}$  s.t.  $\mathbf{x} \in \Omega$ ;
    Find  $\mathbf{y} \in S_C$  s.t.
       $\|\mathbf{x} - \mathbf{y}\| = \min_{\mathbf{z} \in S_C} \|\mathbf{x} - \mathbf{z}\|$ ;
    if ( $\|\mathbf{x} - \mathbf{y}\| < \alpha \delta_{\text{avg}}^C$ )
      discard  $\mathbf{x}$ ;
    else
      accept  $\mathbf{x}$  as a point outside the
      boundary of  $C$ ;
      count = count + 1;
    endif
  end while
end
```

Fig. 6. Scatterplot of square.

Scattered to demonstrate the incremental learning ability of our network.

In all the simulations we use networks with sigmoidal node functions.

A. Demonstration of Good Generalization

1) **Dish-Shell**: This data set, as already stated consists of 1000 points in R^2 equally distributed in two classes. The scatterplot of the data set is shown in Fig. 3(a), and for convenience it is again reproduced as Fig. 9(a), the points from the two classes are represented by $+$ and $*$.

As Dish-Shell contains two classes we need to train two subnets. For each of the classes we generated 2500 points outside the boundary and trained two MLPs each with 20 nodes in the hidden layer and then they were merged by a simple merge. The generalization performed by the network is shown in Fig. 9(b). Comparison of Fig. 9(b) with Fig. 3(b) reveals that the proposed method can do an excellent generalization.

Fig. 7. Points generated outside the boundary of Square for different values of α . (a) $\alpha = 1.0$. (b) $\alpha = 1.5$. (c) $\alpha = 2.0$. (d) $\alpha = 2.5$.

Fig. 8. Points generated outside the boundary of Dish-Shell for various values of α . (a) $\alpha = 1.0$. (b) $\alpha = 1.5$. (c) $\alpha = 2.0$. (d) $\alpha = 2.5$.

2) *3-D-Elongated*: 3-D-Elongated is a data set in R^3 having two classes. Each class has 500 points (this data is similar to a data set used in [10]). Fig. 10 shows the projection of the data on various planes.

We generated 3500 points outside the boundary of each class and trained two MLPs with the two data sets. Then they were merged by a simple merge to obtain the final network. For testing the network we randomly generated 100 000 data points in the hyperbox bounding the data. We tested the

trained network with these data points. For 8848 data points the response of the network was high and was significantly low for other points. The plot of those 8848 points are shown in Fig. 11. Comparing Fig. 10 with Fig. 11 we see that the proposed scheme results in an excellent generalization.

3) *Cone-Torus*: Cone-Torus has 400 points in R^2 in both the training and test sets [5], [16]. There are three classes each representing three shapes, namely a cone, a half torus, and a Gaussian. We trained three subnets each with five hidden nodes

Fig. 9. (a) Scatterplot of Dish-Shell. (b) Generalization on Dish-Shell.

Fig. 10. Scatterplot of 3-D-Elongated. (a) Projected on 1-2. (b) Projected on 2-3. (c) Projected on 1-3.

using the data in each class along with 400 points generated in the boundary. The scatterplot of this data (Fig. 12) shows that the classes have considerable overlap. Table V shows the results of our network with this data set as interpreted by our procedure TEST. In Table V, column 2 lists the number of cases for which the network unambiguously detects the correct class, column 3 shows the number of instances for which the network suggests two classes including the correct class. In other words, column 3 represents the data points that are suspected to be in the overlapped area. The column labeled “soft correct” depicts the number of data points that are correctly classified but the decisions are soft. Column 6 gives the number of points on which the network could not make any decision and column 7 gives the total number of points on which soft decisions were made. The last column gives the total number of misclassifications when the decisions are made like a conventional MLP, i.e., we take the maximum output of the network to decide the class label of a test point. Comparing column 8 of Table V with Table VII

(which gives the performance of ordinary MLP on Cone-Torus as reported in [5]) we find that our network can perform as good as ordinary MLP, in addition it can provide a deeper insight into the data. For example, of 59 misclassifications on the test data, 19 are declared as undecided indicating that probably these 19 points are not in the vicinity of the training data. If we take the maximum response of the output units to decide the class we find that out of these 19 undecided points six are only classified correctly. Thus, of the 59 misclassifications 13 (19–6) points were declared as undecided by our network. Of the remaining 46 (59–13) points 8 (39–31) are wrongly classified by the soft decision and the other 38 points probably fall in the overlapped region where the network suggested two possible classes as output.

Table VI shows the results obtained by a conventional MLP with 15 nodes in the hidden layer, but the network outputs are interpreted by our TEST procedure. Comparing Tables V and VI we see that our method gives high single class response to more

Fig. 11. Generalization on 3-D-Elongated. (a) 1-2. (b) 2-3 (c) 1-3.

Fig. 12. Scatterplot of cone-torus. (a) Training data. (b) Test data.

TABLE V
RESULTS ON CONE-TORUS

	1 class	2 class	3 Class	Soft Correct	Undecided	Tot. Soft	Miss
Training	258	104	0	35	2	36	50 (12.5 %)
Test	241	101	0	31	19	39	59 (14.75 %)

TABLE VI
RESULTS ON CONE-TORUS BY CONVENTIONAL MLP

	1 class	2 class	3 Class	Soft Correct	Undecided	Tot. Soft	Mis
Training	161	0	0	173	1	238	66 (16.5%)
Test	228	0	0	109	5	167	63 (15.75 %)

points than the MLP for both training and test data. Table VI fails to point out the overlapped cases and as expected many of the soft decisions are incorrect. For example, of the 238 soft decisions 65 are incorrect for the training data; similarly out of 167 soft decisions for the test data, 58 are wrong. The number of undecided cases are also lower than that by our network. This may be attributed to the fact that when the network is confronted

TABLE VII
RESULTS ON CONE-TORUS USING CONVENTIONAL MLP (REPORTED IN [5])

Architecture	Trg error	Test error
2:10:3	15.25 %	14.25%
2:15:3	13.50 %	12.0%

with similar data points but from different classes, it may learn one of the two classes (it cannot learn both classes). Thus, NOT for a SINGLE data point we find high response for more than one class.

4) *Sat-Image*: The Sat-image data set is generated from Landsat Multispectral Scanner image data [5], [6], [17]. It has four components containing the gray value of a pixel captured by four sensors operating in different spectral regions. The data set has six classes representing different kinds of ground covers. The training set has 500 points and the test set has 5935 points. In the literature there are other studies also which use only these four features [6].

Fig. 13. Scatterplot of sat image along the two most significant components.

TABLE VIII
RESULTS ON SAT-IMAGE USING CONVENTIONAL MLP (REPORTED IN [5])

Architecture	Trg error	Test error
4:20:6	79.2 %	75.92%
4:65:6	24.4 %	23.08%

TABLE IX
RESULTS ON SAT-IMAGE USING CONVENTIONAL MLP OBTAINED BY US

Architecture	Trg error	Test error
4:65:6	73 (14.6 %)	1012 (17.0%)
4:50:6	155 (31.0 %)	1917 (32.3%)
4:30:6	171 (34.0 %)	2160 (36.39%)
4:15:6	308 (61.6 %)	3718 (62.64%)

TABLE X
RESULTS ON SAT-IMAGE USING CONVENTIONAL MLP USING TEST

Arch.		1-class	2-class	3-class	4-class	Undecided	Soft correct	Mis
4:65:6	Trng	320	0	0	0	8	107	73
	Test	3750	0	0	0	198	1168	1012
4:50:6	Trng	207	0	0	0	90	136	155
	Test	2390	0	0	0	1054	1540	1917
4:30:6	Trng	57	0	0	0	14	272	171
	Test	537	0	0	0	167	3231	2160
4:15:6	Trng	0	0	0	0	0	192	308
	Test	537	0	0	0	167	2217	3718

In this data set the classes have significant overlap as evident from Fig. 13, which shows the scatterplot of the data along the two most significant principal components (the various shades of gray shows the various classes). Some results on this data using conventional MLP can be obtained in [5], we summarize these results in Table VIII. We also ran a conventional MLP with this data with various architectures. We got better results than what was reported in [5]. Table IX shows the results obtained by us with conventional MLPs on this data set. Table X shows the results of these MLPs as interpreted by our procedure TEST.

TABLE XI
 $|\bar{C}_i|$ FOR RUN I AND RUN II

	$ \bar{C}_1 $	$ \bar{C}_2 $	$ \bar{C}_3 $	$ \bar{C}_4 $	$ \bar{C}_5 $	$ \bar{C}_6 $
Run I	379	342	342	288	212	316
Run II	394	417	373	381	403	365

We shall describe two different Runs with our network which we call Run-I and Run-II, respectively. In Run-I, we generated 5000 points in R^4 in the boundary of each class C_i and additionally we took the points from the classes $C_j, j \neq i$ which lies *outside* the boundary of C_i . Notice that, as the dimensionality of the input space increases, more and more data points are to be generated outside the boundary to properly represent the geometric structure of a class and consequently training of the subnets becomes more expensive computationally. An easy way to bypass this is to consider the points of the other classes which lie outside the boundary of the class in question. These data points are very important to determine the structure of the class. In Run-I we used $\alpha = 2$, for generating points outside the boundary and also for considering the points of other classes. The results of our method for Run-I are shown in Table XII. Table XII reveals that there is significant overlap between the classes, as we find many points with multiple class labels. To obtain a more specific result, i.e., to get more points classified with single class label, we describe another experiment called Run-II. In Run-II, we generated 5000 points outside the boundary with $\alpha = 2$, and considered the data points from other classes with $\alpha = 1$. The results of Run-II are shown in Table XIII.

We will analyze the results on sat image in two parts. First we will explain why Table XII is so different from Table XIII. Then we will compare Table XIII with Table X.

Let S_i be the set of training data points from class i and $\bar{C}_i$ be the set of training points from the remaining classes which lie outside the boundary of S_i . If the classes are not overlapped then $\bar{C}_i$ will be equal to $\cup_{j \neq i} S_j$. Table XI compares Run I and Run II in terms of the number of points for $\bar{C}_i$ considered by our algorithm for two different values of α . Low value of α defines a tight boundary for each class. For example, in case of class 5, we find that for $\alpha = 2$, only 212 points in $\bar{C}_5$ are considered to be outside the boundary of class 5 and this is increased to 402 (it becomes almost double) for $\alpha = 1$. This tells us that either class 5 has some overlap with the remaining classes or the boundary of class 5 is probably “touching” the boundary of other classes. Consequently for this data set with $\alpha = 1$ we expect to get a better result. Next we shall see that, this is indeed the case. But before that it is worth mentioning that using two different values of α one can get some idea about whether different classes are well separated or not. Comparing Tables XII and XIII, we find that use of a tighter class boundary improves the performance of the system drastically. Tighter boundary also reduces ambiguous choices. For example, $\alpha = 2$ results in 79 cases for the training data where our system suggests three classes and this 79 is reduced to zero for $\alpha = 1$. For the test data 864 3-class cases are reduced to just five.

A tighter boundary can reduce the number of undecided cases (the system declines to classify a test point in an area not well

TABLE XII
RESULTS ON SAT-IMAGE BY OUR METHOD (RUN -I)

	1-class	2-class	3-class	4-class	5-class	Undecided	Soft correct	Total Soft	Miscl.
Trng	251	126	79	4	0	1	31	39	90(18%)
Test	2712	1353	864	14	2	420	413	570	1246(20.9 %)

TABLE XIII
RESULTS ON SAT-IMAGE BY OUR METHOD (RUN -II)

	1-class	2-class	3-class	4-class	5-class	Undecided	Soft correct	Total Soft	Miscl.
Trng	384	82	0	0	0	13	16	21	70 (14.0 %)
Test	3865	960	5	0	0	826	205	279	1211 (20.4 %)

Fig. 14. (a) Scatterplot of Scattered_P1. (b) Scatterplot of Scattered.

supported by training data). Of the 1211 misclassifications that are obtained by taking the maximum output to decide the class, 826 cases are undecided. Of these 826 cases only 213 points are classified correctly if the decision is made based on the maximum response.

Analysis of Table X reveals that the use of TEST to interpret the output of a conventional MLP does not result in 2-class and 3-class cases—this behavior is exactly the same as that of cone-torus data set. The misclassifications even with a large architecture like (4:65:6), is comparable to that of our system. One may think, since we are using several networks probably our system has more free parameters—*this is just the opposite*. Note that a (4:65:6), MLP have 650 learnable weights and 71 biases, i.e., a total of 721 parameters, whereas our network has 300 weights and 66 biases, i.e., only 366 parameters—about half that of the net (4:65:6)!

B. Demonstration of Incremental Learning

1) *Scattered*: With this data set we shall demonstrate the incremental learning capabilities of the proposed scheme. Suppose Scattered is obtained in two phases. In the first phase the data set has 600 points equally distributed in three classes, we

call this dataset as Scattered_P1. Further 300 points are added to it to get the final data set Scattered. The scatterplot of both data sets are shown in Fig. 14. Fig. 14(b) shows that the addition of 300 points changes the class structure drastically. Our objective is to train a network $\mathcal{M}_1$ which can classify the points in Scattered_P1 and then augment $\mathcal{M}_1$ to a new network $\mathcal{M}_2$ to classify all points in Scattered. The augmentation will be done using only the additional 300 points.

Since Scattered_P1 has three classes our first network $\mathcal{M}_1$ consists of three subnets. Each subnet was trained with the data points in each class of Scattered_P1 along with 2000 point generated outside the boundary of each class. The generalization produced by $\mathcal{M}_1$ on Scattered_P1 is shown in Fig. 15. Then we get the additional 300 points all of which are from the existing three classes. We tried to classify them by $\mathcal{M}_1$. But none of the 300 points gets classified by $\mathcal{M}_1$. Hence, we conclude that the 300 points obtained later though belong to the set of classes already present in Scattered_P1, they lie outside the boundary of the classes represented by Scattered_P1. Fig. 14(b) shows that this is indeed the case. Thus we train three new subnets with the 300 data points in the three classes along with 2000 points generated outside the boundary of each class. These subnets are merged with the three sets of $\mathcal{M}_1$ by a compound merge and

Fig. 15. (a) Generalization on Scattered_P1 by $\mathcal{M}_1$. (b) Generalization by $\mathcal{M}_2$ on Scattered.

we call the new network as $\mathcal{M}_2$. Fig. 15 shows the generalization achieved by $\mathcal{M}_2$ on Scattered. Fig. 15(b) exhibits an excellent performance of our scheme in terms of incremental learning ability and generalization capability. In the given example, the classes are well separated. In case the classes have overlap, the discussion made in Section IV is equally applicable here.

VI. DISCUSSION

In this paper, we proposed a novel scheme to train an MLP so that it does not respond to data points which lie far from the training sample. The training scheme also equips the MLP with incremental learning capability. Also, unlike conventional MLP, it can detect data points that fall in overlapped regions. Our method is based on training several nets with simpler tasks and then merging them for the complete task expected of the network. In this context we have proposed two merging schemes. The crucial point in the scheme is a method of generating points outside the “boundary” of a pattern class. There are a few issues which have not been adequately addressed in this work which we state next.

Here we have proposed a naive definition of the boundary of a pattern class and also proposed an algorithm to generate points outside the boundary of a given set of points. The proposed algorithm although simple, works quite well. This problem of generating points outside the boundary of a given set of points itself is a quite interesting but difficult problem which needs further investigations. Another important problem, estimation of the number of points to be generated outside the boundary, for proper training has also not been addressed in the present work. The number of points to be generated has been chosen in an adhoc manner considering the size of the inflated hyperbox of the data.

An important characteristic of the method is that it can detect points in the area of overlap of two or more classes and can thus deal with overlapped classes in a better way than the conventional MLP. Apparently it may seem that this method may not work for some data sets. For example, if we have a data set with two touching classes, a conventional MLP or any other classifier will be able to arrive at zero misclassification for all the training points. But in our method the points from both classes which lies near the separating plane of the classes will be classified

into both classes. In such cases, the classifier which produces zero misclassification on training data is very sensitive to the training data, i.e., if we give a little perturbation to the points near the separating plane, they will change classes, which we think is *not* desirable and is an indicator of bad generalization.

Our computation experience shows that the training time for each of the subnets dealing with a simple two class problem is much less than that of the training time for the complex multi-class problem. Again, the subnets can be easily trained in parallel. Thus, the method also provides a considerable savings in training time though it uses more data points than present in the training set.

This paper aimed at testing the feasibility of a new training scheme for MLPs which can guarantee “proper” generalization and incremental learning. The simulation results show that our training scheme serves the purpose to a large extent.

REFERENCES

- [1] J. C. Bezdek, J. Keller, R. Krishnapuram, and N. R. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Boston, MA: Kluwer, 1999.
- [2] B. Firtzke, “Fast learning with incremental RBF networks,” *Neural Processing Lett.*, vol. 1, no. 1, pp. 2–5, 1994.
- [3] L. Fu, H. Hsu, and J. C. Principe, “Incremental backpropagation learning networks,” *IEEE Trans. Neural Networks*, vol. 7, pp. 757–761, May 1996.
- [4] S. Haykin, *Neural Networks—A Comprehensive Foundation*. New York: Proc. Conc., 1994.
- [5] L. Kuncheva, *Fuzzy Classifiers*. Heidelberg, Germany: Physica-Verlag, 2000.
- [6] A. Laha and N. R. Pal, “Some novel classifiers designed using prototypes extracted by a new scheme based on self organizing feature map,” *IEEE Trans. Syst. Man, Cybern. B*, vol. 31, pp. 881–890, Aug. 2001.
- [7] D. J. C. MacKay, “Bayesian interpolation,” *Neural Comput.*, vol. 4, no. 3, pp. 415–447, 1992.
- [8] —, “A practical Bayesian framework for backprop networks,” *Neural Comput.*, vol. 4, no. 3, pp. 448–472, 1992.
- [9] —, “The evidence framework applied to classification networks,” *Neural Comput.*, vol. 4, no. 5, pp. 698–714, 1992.
- [10] J. Mao and A. K. Jain, “Artificial neural networks for feature extraction and multivariate data projection,” *IEEE Trans. Neural Networks*, vol. 6, pp. 296–317, Mar. 1995.
- [11] B. D. Ripley and J. P. Rasson, “Finding the edge of a poisson forest,” *J. Appl. Probability*, vol. 14, pp. 483–491, 1977.
- [12] S. Schaal and C. G. Atkeson, “Constructive incremental learning from only local information,” *Neural Comput.*, vol. 10, pp. 2047–2084, 1998.
- [13] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, 2001.

- [14] S. P. Smith and A. K. Jain, "Testing of uniformity in multidimensional data," *IEEE Trans. Pattern Anal. Machine Learning*, vol. PAML-6, pp. 73–81, Jan. 1984.
- [15] F. F. Soulie and P. Gallinari, Eds., *Industrial Applications of Neural Networks*, Singapore: World Scientific, 1998.
- [16] Data set [Online]. Available: <http://www.bangor.ac.uk/mas00a/Z.txt> and <http://www.bangor.ac.uk/mas00a/Zte.txt>
- [17] Data set [Online]. Available: <ftp://ftp.dice.ucl.ac.be/pub/neural-nets/ELENA/databases>
- [18] Data set [Online]. Available: <http://www.dice.ucl.ac.be/neural-nets/Research/Projects/ELENA/elena.htm>

Debrup Chakraborty received the B.E. degree in mechanical engineering from Jadavpur University, Calcutta, India, in 1997 and the M.Tech. degree in computer science from the Indian Statistical Institute, Calcutta, in 1999. He is currently working toward the Ph.D. degree in the Electronics and Communication Sciences Unit of the Indian Statistical Institute.

His primary research interests include pattern recognition, neural networks, neuro-fuzzy integration schemes, and medical image analysis.

Nikhil R. Pal (M'91–SM'00) received the B.Sc. degree with honors in physics and the Master of Business Management degree from the University of Calcutta, Calcutta, India, in 1978 and 1982, respectively. He received the M.Tech. (Comp. Sc.) and Ph.D. (Comp. Sc.) degrees from the Indian Statistical Institute in 1984 and 1991, respectively.

Currently, he is a Professor in the Electronics and Communication Sciences Unit of the Indian Statistical Institute, Calcutta. From September 1991 to February 1993, July 1994 to December 1994, October 1996 to December 1996, and January 2000 to July 2000, he was with the Computer Science Department of the University of West Florida, Pensacola. His research interest includes image processing (LADAR, mammograms, MRIs), pattern recognition, fuzzy sets theory, measures of uncertainty, neural networks, genetic algorithms, and fuzzy logic controllers. He has coauthored a book titled *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing* (Boston, MA: Kluwer, 1999); coedited two volumes titled *Advances in Pattern Recognition and Digital Techniques* (Narosa, India: ICAPRDT99, 1999) and *Advances in Soft Computing* (New York: Springer-Verlag, 2002), and edited a book titled *Pattern Recognition in Soft Computing Paradigm* (Singapore: World Scientific, 2001). He is an Associate Editor of the *International Journal of Fuzzy Systems*, *International Journal of Approximate Reasoning*, *IEEE TRANSACTIONS ON FUZZY SYSTEMS*, and *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B*. He is an Area Editor of *Fuzzy Sets and Systems*. He is a Steering Committee member of the journal *Applied Soft Computing*, Elsevier Science, an independent theme chair of the World Federation of Soft Computing and a governing board member of the Asia Pacific Neural Net Assembly. He was the Program Chair of the Fourth International Conference on Advances in Pattern Recognition and Digital Techniques in December 1999 and was the General Chair of 2002 AFSS International Conference on Fuzzy Systems.