Generalized Clustering Networks and Kohonen's Self-Organizing Scheme

Nikhil R. Pal, *Member, IEEE,* James C. Bezdek, *Fellow, IEEE,* and Eric C.-K. Tsao

***Abstract*—This paper first discusses the relationship between the sequential hard *c*-means (SHCM) and learning vector quantization (LVQ) clustering algorithms. These methods suffer from several major problems. For example, they depend heavily on initialization. If the initial values of the cluster centers are outside the convex hull of the input data, such algorithms, even if they terminate, may not produce meaningful results in terms of prototypes for clustering. This is due in part to the fact that they update only the winning prototype for every input vector. We also discuss the impact and interaction of these two families of methods with Kohonen's self-organizing feature mapping (SOFM), which is not a clustering method, but which often lends ideas to clustering algorithms. Finally, we propose a generalization of LVQ which (may) update all nodes for a given input vector. Moreover, our network attempts to find a minimum of a well-defined objective function. The learning rules depend on the degree of distance match to the winner node; the lesser the degree of match with the winner, the more is the impact on nonwinner nodes. Numerical results indicate that the terminal prototypes generated by this modification of LVQ are generally insensitive to initialization and independent of any choice of learning coefficient(s). We use Anderson's IRIS data to illustrate our method; and we compare our results with the standard LVQ approach.**

***Index Terms*— Cluster analysis, Kohonen, self-organization, learning vector quantization.**

I. Introduction

Clustering algorithms attempt to organize unlabeled feature vectors into clusters or "natural groups" such that points within a cluster are more similar to each other than to vectors belonging to different clusters. Treatments of many classical approaches to this problem include the texts by Kohonen [1], Bezdek [2], Duda and Hart [3], Tou and Gonzalez [4], Hartigan [5], and Jain and Dubes [6]. Kohonen's work has become particularly timely in recent years because of the widespread resurgence of interest in the theory and applications of neural network structures [7].

Kohonen's name is associated with two very different, widely studied and often confused families of algorithms. Specifically, Kohonen initiated study of the prototype generation algorithm called *learning vector quantization* (LVQ); and he also introduced the concept of *self-organizing feature maps* (SOFM) for visual display of certain one- and two-dimensional data sets [1]. LVQ is not a clustering algorithm per se; rather, it can be used to generate crisp (conventional or hard) c-partitions of unlabeled data sets in conjunction with the nearest prototype (NP) classifier designed with its terminal prototypes. LVQ is applicable to p-dimensional unlabeled data. SOFM, on the other hand, attempts to find topological structure hidden in the data and display it in one or two dimensions.

We shall review LVQ and its c-means relative carefully, and SOFM in sufficient detail to understand its intervention in the development of generalized network clustering algorithms. Specifically, SOFM was combined with LVQ by Huntsberger and Ajjimarangsee [8] for clustering. Their algorithm required choosing several parameters such as learning rate, size of an update neighborhood, and a strategy to alter these two parameters during learning. These parameters must be varied from one data set to another to achieve useful results. Moreover, their algorithms are heuristic procedures that are not tied to the optimization of an objective function, and their termination procedure does not guarantee estimates of points having any well defined properties connected to cluster substructure.

This paper presents a generalization of LVQ that is explicitly designed as a clustering algorithm; we refer to this algorithm as GLVQ. Learning rules are derived to optimize an objective function whose goal is to produce "good clusters" with the nearest prototype classifier rule. GLVQ (may) update every node in the clustering net for each input vector. If there is a perfect match between the incoming input and the winner node, GLVQ reduces to LVQ. On the other hand, the greater the mismatch to the winner, the larger the impact of an input vector on updating of nonwinner nodes. We use Anderson's IRIS data to compare the performance of GLVQ with a standard version of LVQ. For this data the final centroids produced by GLVQ are independent of node initialization and learning coefficients. Unlike the methods in [8] GLVQ does not need or depend upon a choice for the update neighborhood—this aspect is taken care of automatically by our approach.

II. Clustering Networks

In LVQ one tries to discover cluster substructure hidden in unlabeled p-dimensional data. We let $X = \{x_1, x_2, \cdots, x_n\} \subset \mathcal{R}^p$ denote the samples at hand, and use c to denote the number of nodes (and clusters in X) in the competitive layer. The salient features of the LVQ model are contained in Fig. 1. The input layer of an LVQ network is connected directly

Manuscript received May 21, 1992; revised July 23, 1992. The work of J. C. Bezdek was supported by NSF Grant IRI-9003252.

N. R. Pal is with the Division of Computer Science, University of West Florida, Pensacola, FL 32514, on leave from the Indian Statistical Institute, Calcutta, India.

J. C. Bezdek and E. C.-K. Tsao are with the Division of Computer Science, University of West Florida, Pensacola, FL 32514.

IEEE Log Number 9203740.

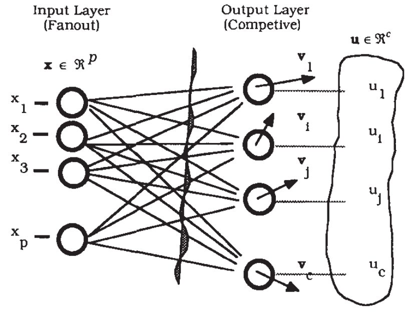


Fig. 1. LVQ clustering networks.

to the output layer. Each node in the output layer has a weight vector (or prototype) attached to it. The prototypes $\boldsymbol{V} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_c)$ are essentially a network array of (unknown) cluster centers $\boldsymbol{v}_i \in \mathcal{R}^p$ for $1 \le i \le c$. In this context the word *learning* refers to finding values for the $\{v_{ij}\}$. When an input vector $\boldsymbol{x}$ is submitted to this network, distances are computed between each $\boldsymbol{v}_r$ and $\boldsymbol{x}$. The output nodes compete, a (minimum distance) winner node, say $\boldsymbol{v}_i$, is found; and it is then updated using one of several update rules. We give a brief specification of LVQ as applied to the data in Section IV. There are other versions of LVQ; this one is usually regarded as the standard form.

The LVQ Clustering Algorithm [1]:

```
LVQ1. Given unlabeled data set $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, , \cdots, \boldsymbol{x}_n\} \subset \mathcal{R}^p$. Fix $c$, $T$, and $\epsilon > 0$.
LVQ2. Initialize $\boldsymbol{V}_0 = (\boldsymbol{v}_{1,0}, \cdots, \boldsymbol{v}_{c,0}) \in \mathcal{R}^{cp}$, and learning rate $\alpha_0 \in (1, 0)$.
LVQ3. For $t = 1, 2, \cdots, T$;
      For $k = 1, 2, \cdots, n$;
          a. Find
             $\|\boldsymbol{x}_k - \boldsymbol{v}_{i,t-1}\| = \min_{1 \le j \le c} \{\|\boldsymbol{x}_k - \boldsymbol{v}_{j,t-1}\|\}$.   (1)
          b. Update the winner:
             $\boldsymbol{v}_{i,t} = \boldsymbol{v}_{i,t-1} + \alpha_t(\boldsymbol{x}_k - \boldsymbol{v}_{i,t-1})$   (2)
          c. Next $k$.
          d. Apply the 1-NP (nearest prototype) rule to the data using
             $u_{\mathrm{LVQ}_{ik}} = \left\{ \begin{array}{ll} 1; & \|\boldsymbol{x}_k - \boldsymbol{v}_i\| \le \|\boldsymbol{x}_k - \boldsymbol{v}_j\|, \quad 1 \le j \le c, j \ne i \\ 0; & \text{otherwise.} \end{array} \right\}$
             $1 \le i \le c \quad \text{and} \quad 1 \le k \le n$.   (3)
LVQ4. Compute $E_t = \|\boldsymbol{V}_t - \boldsymbol{V}_{t-1}\|_1 = \sum_{r=1}^{c} \|\boldsymbol{v}_{r,t} - \boldsymbol{v}_{r,t-1}\|_1 = \sum_{k=1}^{n} \sum_{r=1}^{c} |v_{rk,t} - v_{rk,t-1}|$.
LVQ5. If $E_t \le \epsilon$ stop; Else adjust learning rate $\alpha_t$; Next $t$.
```

The numbers $U_{\mathrm{LVQ}} = [u_{\mathrm{LVQ}_{ik}}]$ are a $c \times n$ matrix that defines a hard or crisp partition of X using the 1-NP classifier assignment rule shown in (3). The vector $\boldsymbol{u}$ shown in Fig. 1 represents a crisp *label vector* that corresponds to one column of this matrix; it contains a 1 in the winner row i at each k; and zeroes otherwise. Our inclusion of the computation of the hard 1-NP c-partition of X at the end of each pass through the data (step LVQ3.d) is **not** part of the LVQ algorithm—that is, the LVQ iterate sequence does not depend on cycling through U's. Ordinarily this computation is done once, noniteratively, outside and after termination of LVQ. Note that LVQ uses the Euclidean distance in step LVQ3.a. This choice corresponds roughly to the update rule shown in (2), since $\nabla_{\boldsymbol{v}}(\|\boldsymbol{x} - \boldsymbol{v}\|_I^2) = -2I(\boldsymbol{x} - \boldsymbol{v}) = -2(\boldsymbol{x} - \boldsymbol{v})$, where I stands for the $p \times p$ identity matrix. The origin of this learning rule comes about by assuming that each $\boldsymbol{x} \in \mathcal{R}^p$ is distributed according to a probability density function $f(\boldsymbol{x})$. LVQ's objective is to find a set of $\boldsymbol{v}_i$'s such that the expected value of the square of the discretization error is minimized:

$$E(\|\boldsymbol{x} - \boldsymbol{v}_i\|^2) = \iint_{\Re^p} \cdots \int \|\boldsymbol{x} - \boldsymbol{v}_i\|^2 f(\boldsymbol{x})\, dx. \tag{4}$$

In this expression, $\boldsymbol{v}_i$ is the winning prototype for each $\boldsymbol{x}$, and will of course vary as $\boldsymbol{x}$ ranges over $\mathcal{R}^p$. A sample function of the above optimization problem is $e = \|\boldsymbol{x} - \boldsymbol{v}_i\|^2$. An optimal set of $\boldsymbol{v}_i$'s can be approximated by applying local gradient descent to a finite set of samples drawn from f. The extant theory for this scheme is contained in [9], which states that LVQ converges in the sense that the prototypes $\boldsymbol{V}_t = (\boldsymbol{v}_{1,t}, \boldsymbol{v}_{2,t}, \cdots, \boldsymbol{v}_{c,t})$ generated by the LVQ iterate sequence converge, i.e., $\{\boldsymbol{V}_t\} \xrightarrow{t \to \infty} \hat{\boldsymbol{V}}$, provided two conditions are met by the sequence $\{\alpha_t\}$ of learning rates used in (2). In particular, this sequence must satisfy

$$\sum_{t=0}^{\infty} \alpha_t = \infty \tag{5a}$$

and

$$\sum_{t=0}^{\infty} \alpha_t^2 < \infty \tag{5b}$$

One choice for the sequence of learning rates that satisfies these conditions is the harmonic sequence, that is, $\alpha_t = 1/t$ for $t \ge 1$; $\alpha_0 \in (0, 1)$. Kohonen has shown that (under some assumptions) steepest descent optimization of the average expected error function (4) is also possible, and both strategies (stochastic approximation and steepest descent) lead to the same update rule (2). The update scheme shown in equation (2) has the simple geometric interpretation shown in Fig. 2.

The winning prototype $\boldsymbol{v}_{i,t-1}$ is simply rotated towards the current data point by moving along the vector $(\boldsymbol{x}_k - \boldsymbol{v}_{i,t-1})$ which connects it to $\boldsymbol{x}_k$. The amount of shift depends on the value of a "learning rate" parameter α_t, which varies from 0 to 1. As seen in Fig. 2, there is no update if $\alpha_t = 0$, and when $\alpha_t = 1$, $\boldsymbol{v}_{i,t}$ becomes $\boldsymbol{x}_k$ ($\boldsymbol{v}_{i,t}$ is just a convex combination of $\boldsymbol{x}_k$ and $\boldsymbol{v}_{i,t-1}$). This process continues until termination via LVQ5, at which time the terminal prototypes yield a "best" hard c-partition of X via (3). Generally, we can make the following observations about LVQ.

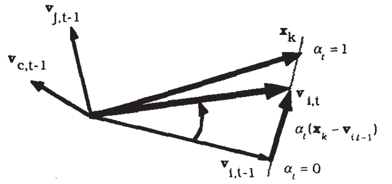


Fig. 2. Updating the winning LVQ prototype.

1) Limit point property: Kohonen in [9] refers to [11], [12] and mentions that LVQ converges to a unique limit if and only if conditions (5) are satisfied. However, nothing was said about what sort or type of points the final weight vectors produced by LVQ are. Since LVQ does not model a well defined property of clusters (in fact, LVQ does not maintain a partition of the data at all), the fact that $\{\boldsymbol{V}_t\} \xrightarrow{t\to\infty} \hat{\boldsymbol{V}}$ does not insure that the limit vector $\hat{\boldsymbol{V}}$ is a good set of prototypes in the sense of representation of clusters or clustering tendencies. All the theorem guarantees is that the sequence HAS a limit point. Thus "good clusters" in $\boldsymbol{X}$ will result by applying the 1-NP rule to the final LVQ prototypes only if, by chance, these prototypes are good class representatives. In other words, the LVQ model is not *driven* by a well specified clustering goal.

2) Learning rate α: Different strategies for α_t often produce different results. Moreover, LVQ seldom terminates unless $\alpha_t \to 0$ (i.e., it is *forced* to stop because successive iterates are necessarily close).

3) Termination: LVQ often runs to its iterate limit, and actually passes the optimal (clustering) solution in terms of minimal apparent label error rate. This is called the "over-training" phenomenon in the neural network literature.

Another, older, clustering approach that is often associated with LVQ is *sequential hard c-means* (SHCM). The updating rule of MacQueen's [14] SHCM algorithm is similar to LVQ. In MacQueen's algorithm the weight vectors are initialized with the first c samples in the data set X. In other words, $\boldsymbol{v}_{r,0} = \boldsymbol{x}_r$, $r = 1, \cdots, c$. Let $q_{r,0} = 1$ for $r = 1, \cdots, c$ ($q_{r,t}$ represents the number of samples that have so far been used to update $\boldsymbol{v}_{r,t}$). Suppose $\boldsymbol{x}_{t+1}$ is a new sample point such that $\boldsymbol{v}_{i,t}$ is closest (with respect to, and without loss, the Euclidean metric) to it. MacQueen's algorithm updates the $\boldsymbol{v}_r$'s as follows (again, index i identifies the winner at this t):

$$\boldsymbol{v}_{i,t+1} = (\boldsymbol{v}_{i,t}q_{i,t} + \boldsymbol{x}_{t+1})/(q_{i,t} + 1) \tag{6a}$$

$$q_{i,t+1} = q_{i,t} + 1 \tag{6b}$$

$$\boldsymbol{v}_{r,t+1} = \boldsymbol{v}_{r,t} \qquad \text{for } r \neq i, \tag{6c}$$

$$q_{r,t+1} = q_{r,t} \qquad \text{for } r \neq i. \tag{6d}$$

MacQueen's process terminates when all the samples have been used once (i.e., when $t = n$). The sample points are then labeled on the basis of nearness to the final mean vectors (that is, using (3) to find a hard c-partition U_{SHCM}). Rearranging (6a), one can rewrite MacQueen's update equation:

$$\boldsymbol{v}_{i,t+1} = \boldsymbol{v}_{i,t} + (\boldsymbol{x}_{t+1} - \boldsymbol{v}_{i,t})/q_{i,t+1}. \tag{7}$$

Writing $1/q_{i,t+1}$ as $\alpha_{i,t+1}$, (7) takes exactly the same form as (2). However, there are some differences between LVQ and MacQueen's algorithm. i) in LVQ sample points are used repeatedly until termination is achieved, while in MacQueen's method sample points are used only once. There are other variants of this c-means algorithm which pass through the data set many times [18]. ii) In MacQueen's algorithm $\alpha_{i,t+1}$ is inversely proportional to the number of points found closest to $\boldsymbol{v}_{i,t}$, so it is possible to have $\alpha_{i,t_1} < \alpha_{j,t_2}$ when $t_1 > t_2$. This is not possible in LVQ. MacQueen attempted to partition feature space $\mathcal{R}^p$ into c subregions, say $(S_1, \cdots, S_c)$, in such a way as to minimize the functional

$$J_M(\boldsymbol{x}, \hat{\boldsymbol{V}}) = \sum_{i=1}^{c} \iint_{\Re^p} \cdots \int \|\boldsymbol{x} - \hat{\boldsymbol{v}}_i\|^2 \, df(\boldsymbol{x})$$

where f is a density function as in LVQ, and $\hat{\boldsymbol{v}}_i$ is the (conditional) mean of the pdf f_i obtained by restricting f to S_i, normalized in the usual way, i.e., $f_i(\boldsymbol{x}) = f(\boldsymbol{x})|_{S_i}/P(S_i)$; and $\hat{\boldsymbol{V}} = (\hat{\boldsymbol{v}}_1, \hat{\boldsymbol{v}}_2, \cdots, \hat{\boldsymbol{v}}_c) \in \mathcal{R}^{cp}$. Let $\boldsymbol{V}_t = (\boldsymbol{v}_{1,t}, \cdots, \boldsymbol{v}_{c,t})$; $S_t = (S_1(\boldsymbol{v}_t), \cdots, S_c(\boldsymbol{v}_t))$ be the minimum distance partition relative to $\boldsymbol{v}_t$; $P(S_j) = \text{prob}\,(\boldsymbol{x} \in S_j)$, $P_{j,t} = P(S_j(\boldsymbol{v}_t)) = \text{prob}\,(\boldsymbol{x} \in S_j(\boldsymbol{v}_t))$; and $\hat{\boldsymbol{v}}_{j,t}$, the conditional mean of $\boldsymbol{x}$ over $S_j(\boldsymbol{v}_t)$, is $\hat{\boldsymbol{v}}_{j,t} = \int_{S_j}(\boldsymbol{v}_t)\boldsymbol{x}\,df(\boldsymbol{x})/P(S_j)$ when $P(S_j) > 0$, or $\hat{\boldsymbol{v}}_{j,t} = \boldsymbol{v}_{j,t}$ when $P(S_j) = 0$. MacQueen proved that for the algorithm described by (6a)–(6d)

$$\lim_{n\to\infty} \left\{ \frac{\sum_{t=1}^{n} \left(\sum_{j=1}^{c} P_{j,t} \|\boldsymbol{v}_{j,t} - \hat{\boldsymbol{v}}_{j,t}\| \right)}{n} \right\} = 0.$$

Since $\{\hat{\boldsymbol{v}}_j\}$ are conditional means, the partition obtained by applying the nearest prototype labeling method at (3) to them may not always be desirable from the point of view of clustering. Moreover, this result does not eliminate the possibility of slow but indefinite oscillation of the centroids (limit cycles).

LVQ and SHCM have some nice theoretical properties. However, the following example shows that both of these approaches suffer from a common problem that can be quite serious. Suppose the input data $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6\} \subset \mathcal{R}^2$ contains the two classes $\boldsymbol{A} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$ and $\boldsymbol{B} = \{\boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6\}$ as shown in Fig. 3. The initial positions of the centroids $\boldsymbol{v}_{1,0}$ and $\boldsymbol{v}_{2,0}$ are also depicted in Fig. 3. Since the initial centroid for class 2 ($\boldsymbol{v}_{2,0}$) is closer to the remaining four input points than $\boldsymbol{v}_1$, each of them will update (modify) $\boldsymbol{v}_2$ only; $\boldsymbol{v}_1$ will not be changed on the first pass through the data. Moreover, because both update schemes result in the updated centroid being pulled towards the data point some distance along the convex combination of the two points, for

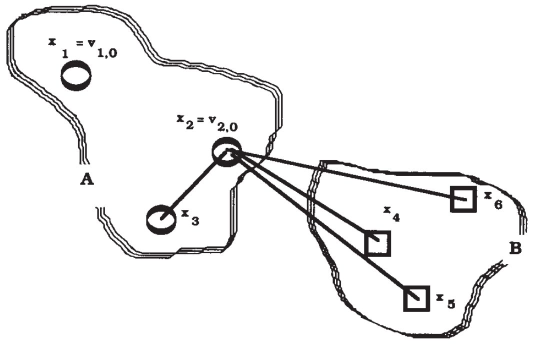


Fig. 3. An initialization problem for LVQ/SHCM.

succeeding passes of LVQ, the chance for $v_{1,0}$ to get updated is very low. Although this strategy gives a locally optimal solution, it is not a desirable solution.

There are two causes for this problem. i) An improper choice of the initial centroids, and ii) each input updates only the winner node. To circumvent problem i), initialization of the v_i's is often done with random input vectors; this reduces the probability of occurrence of the above situation, but does not eliminate it. Bezdek *et al.* [10] attempted to solve problem ii) by updating the winner and some of its neighbor (not topological, but metrical neighbors in $\mathcal{R}^p$) nodes with each input. In their approach, the learning coefficient was reduced both with time and distance from the winner. Their algorithm thus raises two issues: defining an appropriate neighborhood system, and deciding on strategies to reduce the learning coefficient with distance from the winner node. These two issues motivate our consideration of the GLVQ algorithm proposed in the next section.

We conclude this section with a brief description of the SOFM scheme, again using t to stand for iterate number (or time). In this algorithm each prototype $v_{r,t} \in \mathcal{R}^p$ is associated with a display node, say $d_{r,t} \in \mathcal{R}^2$. The vector $v_{i,t}$ that best matches (in the sense of minimum Euclidean distance in the feature space) an incoming input vector x_k is then identified as in (1). $v_{i,t}$ has an "image" $d_{i,t}$ in display space. Next, a topological (spatial) neighborhood $\mathcal{N}(d_{i,t})$ centered at $d_{i,t}$ is defined in display space, and its display node neighbors are located. Finally, the vector $v_{i,t}$ and other prototype vectors in the inverse image $[\mathcal{N}(d_{i,t})]^{-1}$ of spatial neighborhood $\mathcal{N}(d_{i,t})$ are updated using a generalized form of update rule (2):

$$v_{r,t} = v_{r,t-1} + \alpha_{rk,t}(x_k - v_{r,t-1}), \qquad d_{r,t} \in \mathcal{N}(d_{i,t}). \quad (6)$$

The function $\alpha_{rk,t}$ defines a *learning rate distribution* on indices (r) of the nodes to be updated for each input vector x_k at each iterate t. These numbers *impose* (by their definition) a sense of the strength of interaction between (output) nodes. If the $\{v_{r,t}\}$ are initialized with random values and the external inputs $x_k = x_k(t)$ are drawn from a time invariant probability density function $f(x)$, then the point density function of $v_{r,t}$ (the number of $v_{r,t}$'s in the ball $B(x_k, \epsilon)$ centered at the point x_k with radius ϵ) tends to approximate $f(x)$. It has also been shown that the $v_{r,t}$'s attain their values in an "orderly fashion" according to $f(x)$ [9]. This process is continued until the weight vectors "stabilize." In this method then, a learning rate distribution over time and spatial neighborhoods must be defined which decreases with time in order to force termination (to make $\alpha_{rk,t} = 0$). The update neighborhood also decreases with time. While this is clearly not a clustering strategy, the central tendency property of the prototypes often tempts users to assume that terminal weight vectors offer compact representation to clusters of feature vectors; in practice, this is often false.

Huntsberger and Ajjimarangsee [8] used SOFM's to develop clustering algorithms. However, feature mapping is conceptually different from clustering. In feature mapping, as mentioned already, the objective is to extract and visually display hidden topological structure in the p-dimensional data. On the other hand, clustering seeks groups of homogeneous patterns in the feature space. For example, some algorithms attempt to find the centroids (or modes) of different classes. The set of feature vectors closest to a centroid then forms a cluster. One of the algorithms (Algorithm 1) proposed by Huntsberger and Ajjimarangsee [8] is the SOFM algorithm with an additional layer of neurons. This additional set of neurons does not participate in weight updating. After the self-organizing network terminates, the additional layer, for each input, finds the weight vector (prototype) closest to it and assigns the input to that class. Another algorithm in [8], instead of assigning an input vector completely to a class, makes partial assignments to all c classes. In other words, a membership value in [0, 1] is assigned for each class according to the necessary condition for memberships in the fuzzy c-means algorithm [2].

Each of these algorithms updates not only the winner node but some other neighboring nodes. However, since the objective of these algorithms is to find cluster centroids (prototypes), and thence clusters, there is no need to have a topological ordering of the weight vectors. Consequently, the approach taken in [8] seems to mix two objectives, feature mapping and clustering, and the overall methodology is difficult to interpret in either sense.

III. Generalized Learning Vector Quantization GLVQ

In this section we develop a new clustering algorithm which avoids or fixes several of the limitations mentioned earlier. The learning rules are derived from an optimization problem. Let $x \in \mathcal{R}^p$ be a stochastic input vector distributed according to a time invariant probability distribution $f(x)$, and let i be the best matching node as in (2). Let L_x be a loss function which measures the locally weighted mismatch (error) of x with respect to the winner:

$$L_x = L(x;\ v_1, \cdots, v_c) = \sum_{r=1}^{c} g_{ir} \|x - v_r\|^2 \qquad \text{(8a)}$$

where

$$g_{ir} = \left\{ \begin{array}{ll} 1 & \text{if } r = i \\ \dfrac{1}{\sum_{j=1}^{c} \|\boldsymbol{x} - \boldsymbol{v}_j\|^2}, & \text{otherwise} \end{array} \right\}. \tag{8b}$$

Let $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n, \cdots\}$ be a set of samples from $f(\boldsymbol{x})$ drawn at time instants $t = 1, 2, \cdots, n, \cdots$. Our objective is to find a set of c $\boldsymbol{v}_r$'s, say $\boldsymbol{V} = \{\boldsymbol{v}_r\}$, such that the locally weighted error functional $L_{\boldsymbol{x}}$ defined with respect to the winner $\boldsymbol{v}_i$ is minimized over $\boldsymbol{X}$. In other words, we seek to

$$\text{Minimize:}\, \hat{\Gamma}(\boldsymbol{V}) = \iint_{\Re^p} \cdots \int \sum_{r=1}^{c} g_{ir} \|\boldsymbol{x} - \boldsymbol{v}_r\|^2 f(\boldsymbol{x})\, d\boldsymbol{x}. \tag{9}$$

For a fixed set of points $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$ the problem reduces to the unconstrained optimization problem:

$$\text{Minimize:}\ \Gamma(\boldsymbol{V}) = \frac{\sum_{t=1}^{n} \sum_{r=1}^{c} g_{ir} \|\boldsymbol{x}_t - \boldsymbol{v}_r\|^2}{n}. \tag{10}$$

Here $L_{\boldsymbol{x}}$ is a random functional for each realization of $\boldsymbol{x}$, and $\Gamma(\boldsymbol{V})$ is its expectation. Hence exact optimization of Γ using ordinary gradient descent is difficult. We have seen that i, the index for the winner, is a function of $\boldsymbol{x}$ and all of $\boldsymbol{v}_r$'s. The function $L_{\boldsymbol{x}}$ is well defined. If we assume that $\boldsymbol{x}$ has a unique distance from each $\boldsymbol{v}_r$, then i and g_{ir} are uniquely determined, and hence $L_{\boldsymbol{x}}$ is also uniquely determined. However, if the above assumptions are not met, then i and g_{ir} will have discontinuities. In the following discussion we assume that g_{ir} does not have discontinuities so that the gradient of $L_{\boldsymbol{x}}$ exists. As most learning algorithms do [15], we approximate the gradient of $\Gamma(\boldsymbol{V})$ by the gradient of the sample function $L_{\boldsymbol{x}}$. In other words, we attempt to minimize Γ by local gradient descent search using the sample function $L_{\boldsymbol{x}}$. It is our conjecture that the optimal values of $\boldsymbol{v}_r$'s can be approximated in an iterative, stepwise fashion by moving in the direction of gradient of $L_{\boldsymbol{x}}$. The algorithm can now be derived as follows (for notational simplicity the subscript for $\boldsymbol{x}$ will be ignored); first rewrite L as

$$\begin{aligned} L &= \sum_{r=1}^{c} g_{ir} \|\boldsymbol{x} - \boldsymbol{v}_r\|^2 = \|\boldsymbol{x} - \boldsymbol{v}_i\|^2 + \sum_{\substack{r=1 \\ r \neq i}}^{c} \|\boldsymbol{x} - \boldsymbol{v}_r\|^2 \Big/ \\ &\quad \cdot \sum_{j=1}^{c} \|\boldsymbol{x} - \boldsymbol{v}_j\|^2 \\ &= \|\boldsymbol{x} - \boldsymbol{v}_i\|^2 + \sum_{r=1}^{c} \|\boldsymbol{x} - \boldsymbol{v}_r\|^2 \Big/ \\ &\quad \cdot \sum_{j=1}^{c} \|\boldsymbol{x} - \boldsymbol{v}_j\|^2 - \left(\|\boldsymbol{x} - \boldsymbol{v}_i\|^2 \Big/ \sum_{j=1}^{c} \|\boldsymbol{x} - \boldsymbol{v}_j\|^2\right) \\ &= \|\boldsymbol{x} - \boldsymbol{v}_i\|^2 + 1 - \left(\|\boldsymbol{x} - \boldsymbol{v}_i\|^2 \Big/ \sum_{j=1}^{c} \|\boldsymbol{x} - \boldsymbol{v}_j\|^2\right). \end{aligned} \tag{11}$$

Now differentiating L with respect $\boldsymbol{v}_i$ we get the following expression (after some algebraic manipulations):

$$\nabla_{\boldsymbol{v}} L(\boldsymbol{v}_i) = -2(\boldsymbol{x} - \boldsymbol{v}_i) \frac{D^2 - D + \|\boldsymbol{x} - \boldsymbol{v}_i\|^2}{D^2} \tag{12}$$

where $D = \sum_{r=1}^{c} \|\boldsymbol{x} - \boldsymbol{v}_r\|^2$. On the other hand, differentiation of L with respect to $\boldsymbol{v}_j$ $(j \neq i)$ yields

$$\nabla_{\boldsymbol{v}} L(\boldsymbol{v}_j) = -2(\boldsymbol{x} - \boldsymbol{v}_j) \frac{\|\boldsymbol{x} - \boldsymbol{v}_i\|^2}{D^2}. \tag{13}$$

Based on equations (12) and (13), the update rules can be formulated as

$$\boldsymbol{v}_{i,\,t} = \boldsymbol{v}_{i,\,t-1} + \alpha_t(\boldsymbol{x} - \boldsymbol{v}_{i,\,t-1}) \frac{D^2 - D + \|\boldsymbol{x} - \boldsymbol{v}_{i,\,t-1}\|^2}{D^2} \quad \text{for the winner node } i, \text{ and} \tag{14}$$

$$\boldsymbol{v}_{j,\,t} = \boldsymbol{v}_{j,\,t-1} + \alpha_t(\boldsymbol{x} - \boldsymbol{v}_{j,\,t-1}) \frac{\|\boldsymbol{x} - \boldsymbol{v}_{i,\,t-1}\|^2}{D^2} \quad \text{for the other } (c-1) \text{ nodes.} \tag{15}$$

To avoid possible oscillations of the solution, the amount of correction should be reduced as iteration proceeds. Moreover, like optimization techniques using subgradient descent search, as one moves closer to an optimum the amount of correction should be reduced (in fact, α_t should satisfy the following two conditions: as $t \to \infty$; $\alpha_t \to 0$ and $\sum \alpha_t \to \infty$) [16].

On the other hand, in the presence of noise, under a suitable assumption about subgradients, the search becomes successful if the conditions in (5) are satisfied. Thus we recommend a decreasing sequence of α_t $(0 < \alpha_t < 1)$ satisfying the conditions in (5). Conditions (5) ensure that α_t is neither reduced too fast nor too slow. From the point of view of learning, as Grossberg [17] pointed out, the learning system should be stable enough to remember old learned patterns, and yet plastic enough to learn new patterns (Grossberg calls it the *stability-plasticity dilemma*). Condition (5a) helps to make the system plastic, while (5b) enforces stability. In other words, an incoming input should not affect the parameters of a learning system too strongly, thereby enabling it remember old learned patterns (stability); at the same time the system should be responsive enough to recognize any new trend in the input (plasticity). Hence, α_t can be taken as $\alpha_0(1 - t/T)$, where T is the maximum number of iterations the learning process is allowed to execute and α_0 is the initial value of the learning parameter. Referring to (15), we see that when the match is perfect then nonwinner nodes are not updated; in other words, this strategy then reduces to LVQ. On the other hand, as the match between $\boldsymbol{x}$ and the winner node $\boldsymbol{v}_i$ decreases, the impact on other (nonwinner) nodes increases. This seems to be an intuitively desirable property. We summarize the GLVQ algorithm as follows.

GLVQ Clustering Algorithm:

GLVQ1. Given unlabeled data set $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, , \cdots, \boldsymbol{x}_n\} \subset \mathcal{R}^p$. Fix c, T, and $\epsilon > 0$.

GLVQ2. Initialize $\boldsymbol{V}_0 = (\boldsymbol{v}_{1,0}, \cdots, \boldsymbol{v}_{c,0}) \in \mathcal{R}^{cp}$, and learning rate $\alpha_0 \in (1, 0)$.

GLVQ3. For $t = 1, 2, \cdots, T$. Compute $\alpha_t = \alpha_0\,(1 - t/T)$.

For $k = 1, 2, \cdots, n$:

a. Find $\|\boldsymbol{x}_k - \boldsymbol{v}_{i,\,t-1}\| = \min\limits_{1 \le j \le c} \{\|\boldsymbol{x}_k - \boldsymbol{v}_{j,\,t-1}\|\}$. (1)

b. Update all (c) weight vectors $\{\boldsymbol{v}_{r,\,t}\}$ with

$$\boldsymbol{v}_{i,\,t} = \boldsymbol{v}_{i,\,t-1} + \alpha_t(\boldsymbol{x}_k - \boldsymbol{v}_{i,\,t-1}) \cdot \frac{D^2 - D + \|\boldsymbol{x}_k - \boldsymbol{v}_{i,\,t-1}\|^2}{D^2} \tag{14}$$

$$\boldsymbol{v}_{r,\,t} = \boldsymbol{v}_{r,\,t-1} + \alpha_t(\boldsymbol{x}_k - \boldsymbol{v}_{r,\,t-1}) \cdot \frac{\|\boldsymbol{x}_k - \boldsymbol{v}_{r,\,t-1}\|^2}{D^2} (r \neq i) \tag{15}$$

Next k.

GLVQ4. Compute $E_t = \|\boldsymbol{V}_t - \boldsymbol{V}_{t-1}\|_1 = \sum_{r=1}^{c} \|\boldsymbol{v}_{r,t} - \boldsymbol{v}_{r,t-1}\|_1 = \sum_{k=1}^{n} \sum_{r=1}^{c} |v_{rk,\,t} - v_{rk,\,t-1}|$.

GLVQ5. If $E_t \le \epsilon$ stop; Else Next t.

GLVQ6. Compute noniteratively the nearest prototype GLVQ c-partition of X:

$$u_{\mathrm{GLVQ}_{ik}} = \left\{ \begin{array}{ll} 1; & \|\boldsymbol{x}_k - \boldsymbol{v}_i\| \le \|\boldsymbol{x}_k - \boldsymbol{v}_j\|, \quad 1 \le j \le c, j \neq i \\ 0; & \text{otherwise} \end{array} \right\}$$

$$1 \le i \le c \quad \text{and} \quad 1 \le k \le n.$$

Comments on GLVQ:

1) There is no need to choose an update neighborhood.
2) No reduction of the learning coefficient with distance (either topological or in $\mathcal{R}^p$) from the winner node is required. However, an implicit reduction is automatically and adaptively done by the learning rules.
3) For each input vector, either all nodes get updated or no node does. When there is a perfect match to the winner node, no node is updated. In this case GLVQ reduces to LVQ.
4) The greater the mismatch to the winner (i.e., the higher the quantization error), the greater is the impact to the weight vectors associated with other nodes. Here by quantization error we mean the error involved in representing a set of input vectors by a prototype—in the above case the weight vector associated with the winner node.
5) The learning process attempts to minimize a well-defined objective function.
6) Our termination strategy is based on small successive changes in the cluster centers. This method of algorithmic control offers the best set of centroids for compact representation (quantization) of the data in each cluster.

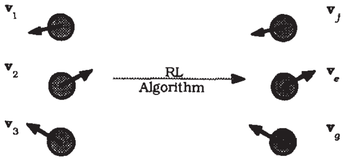


Fig. 4. Illustration of the relabeling algorithm.

Next, we turn to some numerical experiments that illustrate the difference between LVQ and GLVQ.

IV. Experimental Results

We use Anderson's IRIS data [13] as an experimental data set. Let $\boldsymbol{X}$ be the IRIS data. $\boldsymbol{X}$ contains 50 (labeled) vectors in $\boldsymbol{R}^4$ for each $c = 3$ classes of IRIS subspecies. Properties of the data are well known [3]. $\boldsymbol{X}$ has been used in many papers to illustrate various clustering (unsupervised) and classifier (supervised) designs. Typical error rates for supervised designs are 0–5 mistakes (resubstitution); and for unsupervised designs, around 15 "mistakes." Our comparison below is based on training LVQ and GLVQ with all of the data (until termination occurs); and then classifying each data point in $\boldsymbol{X}$ with the 1-NP rules shown above.

All clustering algorithms produce clusters that have *numerical* (not physical) labels. In LVQ and GLVQ, it is the c output nodes that have unidentified labels. When an algorithm is being tested on labeled data (which, of course, we presume to be unlabeled during training), we use the following algorithm to (re)label terminal weight vectors so that terminal prototypes correspond to the correct physical labels of the subclasses.

Relabeling Algorithm (RL):

```
Given   $n_i$ physically labeled vectors in $\boldsymbol{X}_i \in \boldsymbol{R}^p$;
        $1 \le i \le c$. Let $\sum_{i=1}^{c} n_i = n$. Let
        $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_c\}$ be any (c) prototypes
        in $\boldsymbol{R}^p$.
Find    Physical labels for the $\{\boldsymbol{v}_i\}$.
Let     $p_{ij}$ =percentage of points from class $i$
        closest to $\boldsymbol{v}_j$ via the nearest prototype rule.
        $i \leftarrow 1$.
While   $i \le c$:
            1) Find the maximum value in $P = [p_{ij}]$,
               say $p_{i*j*}$.
            2) Relabel $\boldsymbol{v}_{j*} \mapsto \boldsymbol{v}_{i*}$.
            3) Delete row $i*$ and column $j^*$ from $P$.
            4) $i \leftarrow i + 1$.
Wend
```

We illustrate the RL algorithm. Let $\{1, 2, 3\}$ =algorithmic labels, and $\{e, f, g\}$ =physical labels. Suppose the confusion

TABLE I
CENTROIDS (GLVQ) FOR DIFFERENT INITIALIZATIONS WITH $\alpha = 0.6$ AND $T = 500$

I1	I2	I3
v_1: 5.007 3.424 1.472 0.251	5.007 3.424 1.472 0.251	5.007 3.424 1.472 0.251
v_2: 5.882 2.742 4.387 1.433	5.882 2.742 4.387 1.433	5.882 2.742 4.387 1.433
v_3: 6.848 3.078 5.708 2.058	6.848 3.078 5.708 2.058	6.848 3.078 5.708 2.058

TABLE II
CENTROIDS (LVQ) FOR DIFFERENT INITIALIZATIONS WITH $\alpha = 0.6$ AND $T = 500$

I1	I2	I3
v_1: 0.245 0.037 0.830 0.706	5.005 3.426 1.463 0.247	5.005 3.426 1.463 0.247
v_2: 5.836 3.057 3.752 1.201	5.882 2.743 4.385 1.432	5.882 2.743 4.385 1.432
v_3: 0.554 0.432 0.564 0.224	6.849 3.079 5.710 2.061	6.849 3.079 5.710 2.061

matrix C associated with a run on the IRIS data is as follows:

$$C = \begin{array}{c} \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 28 & 12 & 10 \\ 46 & 3 & 1 \\ 7 & 5 & 38 \end{bmatrix} \end{array}$$

Convert C to percentage matrix P:

$$P = \begin{array}{c} \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} .56 & .24 & .20 \\ .92 & .06 & .02 \\ .14 & .10 & .76 \end{bmatrix} \end{array}.$$

Applying the RL algorithm to P results in the following assignments:

$$P = \begin{array}{c} \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} .56 & .24 & .20 \\ .92 & .06 & .02 \\ .14 & .10 & .76 \end{bmatrix} \end{array} \rightarrow \begin{array}{c} \\ e \\ g \end{array} \begin{array}{c} \begin{array}{cc} 2 & 3 \end{array} \\ \begin{bmatrix} .24 & .20 \\ .10 & .76 \end{bmatrix} \end{array} \rightarrow \begin{array}{c} 2 \\ e[.24]. \end{array}$$

That is, $1 = f$, $3 = g$ and $2 = e$. Thus the output node labels are transformed as shown in Fig. 4.

We count "errors" by submitting $\boldsymbol{X}$ to the nearest prototype rule using the *physical* labels of the output nodes. This affords a means for insuring that the prototypes really represent the "right" subclasses. In this study, to investigate the effect of initialization, we initialized LVQ and GLVQ with four different strategies as follows.

I1: Using random samples from a (pseudo) uniform distribution on [0, 1].

I2: Using c randomly selected input vectors from $\boldsymbol{X}$.

TABLE III
CENTROIDS FOR INITIALIZATIONS WITH IDENTICAL VECTORS, $\alpha = 0.6$ AND $T = 500$

	Final Centroids	
Initial centroids	GLVQ	LVQ
v_1: 0.0 0.0 0.0 0.0	5.007 3.424 1.472 0.251	5.836 3.057 3.752 1.201
v_2: 0.0 0.0 0.0 0.0	5.882 2.742 4.387 1.433	0.0 0.0 0.0 0.0
v_3: 0.0 0.0 0.0 0.0	6.848 3.078 5.708 2.058	0.0 0.0 0.0 0.0
v_1: 9.0 9.0 9.0 9.0	5.007 3.424 1.472 0.251	5.836 3.057 3.752 1.201
v_2: 9.0 9.0 9.0 9.0	5.882 2.742 4.387 1.433	9.0 9.0 9.0 9.0
v_3: 9.0 9.0 9.0 9.0	6.848 3.078 5.708 2.058	9.0 9.0 9.0 9.0

I3: For each feature, the range of values in the data was uniformly subdivided into $c = 3$ subintervals, and the three initial vectors had features that were midpoints of these subintervals.

I4: Using c vectors well outside the convex hull of $\boldsymbol{X}$.

For each of the first three initialization strategies, both LVQ and GLVQ were run with three different initial values of α (0.4, 0.6, 0.8); and for each of these 9 combinations, both algorithms were run for three different numbers of iterations (200, 500, 5000). Throughout the investigation $\epsilon = 0.0001$. As typical illustrations Tables I and II display the centroids ($\boldsymbol{v}_r$) obtained by GLVQ and LVQ, respectively, for the first three initializations with $\alpha = 0.6$ and 500 iterations. (Note that the same initializations were used for Tables I and II.)

Table I shows that GLVQ terminates at the same centroids (up to three decimal places) for three initializations, whereas LVQ terminates at a very different set of centroids for I1 than it does for I2 and I3, as shown in Table II. To gauge the quality of the GLVQ terminal centroids in Table I we list the actual centers of the 50 points in each of the three labeled classes:

$$\begin{aligned} \boldsymbol{v}_{1,\,\text{actual}} &= (5.006 \quad 3.428 \quad 1.462 \quad 0.246) \\ \boldsymbol{v}_{2,\,\text{actual}} &= (5.936 \quad 2.770 \quad 4.260 \quad 1.326) \\ \boldsymbol{v}_{3,\,\text{actual}} &= (6.588 \quad 2.974 \quad 5.552 \quad 2.026). \end{aligned}$$

Thus after 500 iterations, GLVQ is stopped with a maximum deviation in any coordinate of 0.26 (in the first component of $\boldsymbol{v}_{3,\,\text{actual}}$). A glance at Table II on the other hand, shows that LVQ does not produce estimates of quality in any sense of the word for initialization I1.

To test the robustness of the proposed scheme, we initialized the three $\boldsymbol{v}_{i,\,0}$'s with identical vectors that were well outside the convex hull of X. Table III gives the centroids obtained by both algorithms when initializations were done with all *zero* (0.0) vectors or with all *nine* (9.0) vectors. Note that these initial vectors are not only identical to each other, but are far away from the convex hull of the samples (the IRIS

TABLE IV
CLASSIFICATION ERRORS FOR DIFFERENT INITIALIZATIONS, α AND NUMBER OF ITERATIONS

Iterations	Initialization	α_0	Number of errors	
			GLVQ	LVQ
200	I1	0.4	17	100
		0.6	17	100
		0.8	17	50
	I2	0.4	17	17
		0.6	17	17
		0.8	17	17
	I3	0.4	17	17
		0.6	17	17
		0.8	17	17
500	I1	0.4	17	100
		0.6	17	100
		0.8	17	50
	I2	0.4	17	17
		0.6	17	17
		0.8	17	17
	I3	0.4	17	17
		0.6	17	17
		0.8	17	17
5000	I1	0.4	17	100
		0.6	17	100
		0.8	17	50
	I2	0.4	17	17
		0.6	17	17
		0.8	17	17
	I3	0.4	17	17
		0.6	17	17
		0.8	17	17

data is wholly contained in the four dimensional hyperbox $[4.3, 7.9] \times [2.0, 4.4] \times [1.0, 6.9] \times [0.1, 2.5]$). In both of these cases the terminal vectors produced by GLVQ are found to be almost identical with those produced by other initializations. However, LVQ fails badly in these two cases. Note, for example, that the centers for classes 2 and 3 remain at their inital values for both initializations.

Table IV contains errors in classification produced by the nearest prototype rule after the centroids obtained by GLVQ and LVQ were corrected using the relabeling algorithm. Note that GLVQ is essentially finished in 200 iterations in the sense that it always produces centroids that yield 17 errors in 150 tries. The error rate obtained with LVQ centroids, on the other hand, varies with α, and does not show marked improvement for even large numbers of iterations when using I1.

V. CONCLUSIONS

We have proposed a generalization of LVQ for clustering. The proposed algorithm needs only a specification of the learning rate sequence. Unlike the methods in [8] we avoid the necessity of defining an update neighborhood scheme; GLVQ either updates all nodes for an input vector, or it does not update any. When an input vector exactly matches the winner node, GLVQ reduces to LVQ. Otherwise, all nodes are updated inversely proportionally to their distances from the input vector. In an experimental investigation with the well known IRIS data, GLVQ always produces results as good as those obtained by LVQ. Conversely, there are situations in which LVQ fails, but GLVQ still produces good results. GLVQ-generated centroids are relatively invariant to the number of iterations, learning coefficient and the choice of initial centroids. Based on our numerical results we offer the following concluding remarks.

1) GLVQ final centroids do not seem sensitive to initialization (up to three digits of precision) for fixed learning coefficients and/or numbers of iterations. This is not the case with LVQ.
2) In GLVQ, for all posible combinations tried, the final centroids were found to be identical at least up to two digits of precision.
3) For LVQ, with initializations I2 and I3, the centroids were usually the same up to at least 2 digit precision. However, with I1 the observed LVQ centroids are completely different and thereby result in 33% to 67% classification errors.
4) Except for initialization I1, both LVQ and GLVQ result in 17 ($\approx$ 11%) classification errors.
5) Except for initialization I1, the centroid vectors produced by LVQ and GLVQ were almost identical (up to 2 digit precision).
6) Even when GLVQ is initialized with identical vectors well outside the convex hull generated by the input sample points, it produces the same results, but LVQ fails.

Behavior of GLVQ under batch updating, its mathematical properties and its performance on various other data sets constitute subjects for future investigations.

REFERENCES

[1] T. Kohonen, *Self-Organization and Associative Memory*. Berlin, Germany: Springer-Verlag, 1989, 3rd ed.

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

[3] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[4] J. Tou and R. Gonzalez, *Pattern Recognition Principles*. Reading, PA: Addison-Wesley, 1974.

[5] J. Hartigan, *Clustering Algorithms*. New York: Wiley, 1975.

[6] A. Jain and R. Dubes, *Algorithms that Cluster Data*. Englewood Cliffs, NJ: Prentice-Hall.

[7] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks*. Reading, PA: Addison-Wesley, 1989.

[8] T. Huntsberger and P. Ajjimarangsee, "Parallel self-organizing feature maps for unsupervised pattern recognition," *Int. J. General Systems*, vol. 16, pp. 357–372, 1989.

[9] T. Kohonen, "Self-organizing maps: optimization approach," in *Artificial Neural Networks*, T. Kohonen, K. Makisara, O. Simula, and J. Kangas, Eds. New York: Elsevier, 1991, pp. 981–990.

[10] J. C. Bezdek, E. C. Tsao, and N. R. Pal, "Fuzzy Kohonen clustering networks," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Mar. 1992, San Diego, CA, pp. 1035–1041.

[11] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Stat.*, vol. 22, pp. 400–407, 1951.

[12] A. E. Albert and L. A. Gardner, Jr., *Stochastic Approximation and Nonlinear Regression*. Cambridge, MA: MIT Press, 1967.

[13] E. Anderson, "The IRISes of the Gaspe Peninsula," *Bulletin of the American IRIS Society*, vol. 59, pp. 2–5, 1939.

[14] J. MacQueen, "Classification and analysis of multivariate observations, Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Stat. and Prob.*, 1967, pp. 281–297.

[15] Y. Z. Tsypkin, *Foundations of the Theory of Learning Systems.* New York: Academic, 1973 (translated by Z. J. Nikolic).

[16] B. T. Polyak, *Introduction to Optimization.* New York: Optimization Software Inc., 1987.

[17] S. Grossberg, *Studies of Mind and Brain.* Boston, MA: Reidel, 1982.

[18] E. Forgy, "Cluster analysis of multivariate data: Efficiency versus interpretability of classifications," WNAR Meetings, Univ. of California, Riverside, June 22–23, 1965 (Biometrics, 21(3)).

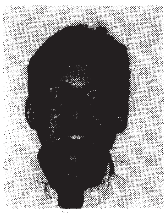
Nikhil R. Pal (M'91) received the B.Sc. (Hons.) degree in physics and the M.B.M. degree in operations research in 1979 and 1982, respectively, from the University of Calcutta. He received the M. Tech and Ph.D. degrees in computer science from the Indian Statistical Institute, Calcutta, in 1984 and 1991, respectively.

He is with the Electronics and Communication Sciences Unit of the Indian Statistical Institute, and is currently visiting the University of West Florida, Pensacola, FL. He is also a guest lecturer at the University of Calcutta. His research interests include image processing, pattern recognition, artificial intelligence, fuzzy sets and systems, uncertainty measures, and neural networks.

James C. Bezdek (M'80–SM'90–F'92) received the B.S. degree from the University of Nevada, Reno, in 1969, and the Ph.D. degree from Cornell University, Ithaca, NY, in 1973.

His research interests include pattern recognition algorithms, neural networks, image processing and machine vision, medical computing, and expert systems.

Dr. Bezdek is the founding editor of the *International Journal of Approximate Reasoning* and the IEEE TRANSACTIONS ON FUZZY SYSTEMS, and is an associate editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS, and International Journals of Man-Machine Studies, Applied Intelligence, General Systems, and Fuzzy Sets and Systems. He is a past president of the International Fuzzy Systems Association (IFSA) and the North American Fuzzy Information Processing Society (NAFIPS), and was an ACM national lecturer for the 1990–1992 program years.



Eric C.-K. Tsao was born in Taipei, Taiwan, Republic of China, on November 30, 1962. He received the B.E. degree in electronics engineering from Tamkang University, Taiwan, in 1984, the M.E.E. degree in electrical engineering from the University of Houston, Texas, in 1987, and the Ph.D. degree in electrical engineering from Northwestern University, Illinois, in 1991.

During 1988–1989, he was a Research Programmer and System Manager in the Cardiology Cath. Laboratory at the University of Chicago Medical Center. During 1989–1991, he was a Research Assistant in the Computer Vision and Image Processing Laboratory at Northwestern University and in the Department of Radiology at the University of Chicago. Since 1991, he has been an Assistant Professor in the Department of Computer Science at the University of West Florida. He was to join the faculty of the Department of Computer Science and Information Engineering at National Chung Cheng University, Chiayi, Taiwan, in 1992. His research interests include neural networks, image analysis and processing, fuzzy theory, computer vision, and signal processing.

Dr. Tsao is a member of the IEEE Computer Society, the IEEE SMC Society, the Pattern Recognition Society, ACM, and INNS.