1993/3017

N93-22207

S1-63

150401

p. 27

TWO GENERALIZATIONS OF KOHONEN CLUSTERING

James C. Bezdek*
Nikhil R. Pal+
Eric C.K.Tsao

Division of Computer Science
The University of West Florida
Pensacola, FL 32514

*Research supported by NSF Grant Number IRI-9003252
+On leave from Indian Statistical Institute, Calcutta

KEYWORDS

Fuzzy Clustering
Fuzzy LVQ
Generalized LVQ
Learning Vector Quantization

ABSTRACT

This paper discusses the relationship between the sequential hard c-means (SHCM), learning vector quantization (LVQ), and fuzzy c-means (FCM) clustering algorithms. LVQ and SHCM 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 cluster representation. 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 with Kohonen's self-organizing feature mapping (SOFM), which is not a clustering method, but which often lends ideas to clustering algorithms. Then we present two generalizations of LVQ that are explicitly designed as clustering algorithms; we refer to these algorithms as generalized LVQ = GLVQ; and fuzzy LVQ = FLVQ. Learning rules are derived to optimize an objective function whose goal is to produce "good clusters". GLVQ/FLVQ (may) update every node in the clustering net for each input vector. Neither GLVQ nor FLVQ depends upon a choice for the update neighborhood or learning rate distribution - these are taken care of automatically. Segmentation of a gray tone image is used as a typical application of these algorithms to illustrate the performance of GLVQ/FLVQ .

Volume II

1. INTRODUCTION : LABEL VECTORS AND CLUSTERING

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 Dubes and Jain[6]. Kohonen's work has become timely in recent years because of the widespread resurgence of interest in the theory and applications of neural network structures [7].

Label Vectors. To characterize solution spaces for clustering and classifier design, let c denote the number of clusters, $1 < c < n$, and set :

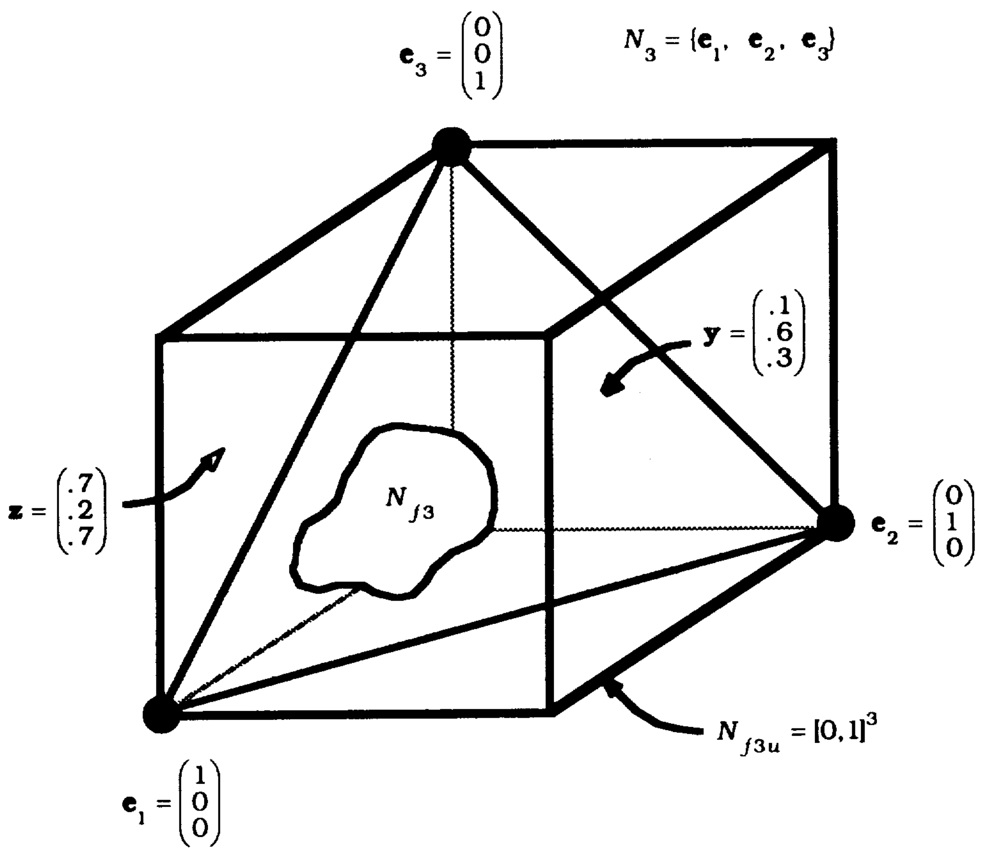
$$N_{fcu} = \{\mathbf{y} \in \Re^c \mid y_k \in [0, 1] \ \forall k\} = \text{(unconstrained) } \textit{fuzzy labels} \quad ; \quad (1a)$$
$$N_{fc} = \{\mathbf{y} \in N_{fcu} \mid \Sigma y_k = 1\} = \text{(constrained) } \textit{fuzzy labels} \quad ; \quad (1b)$$
$$N_c = \{\mathbf{y} \in N_{fc} \mid y_k \in \{0, 1\} \ \forall k\} = \textit{hard labels} \text{ for c classes} \quad . \quad (1c)$$

N_c is the canonical basis of Euclidean c-space; N_{fc} is its convex hull; and N_{fcu} is the unit hypercube in $\Re^c$. Figure 1 depicts these sets for c=3. For example, the vector $\mathbf{y} = (.1, .6, .3)^T$ is a typical constrained fuzzy label vector; its entries lie between 0 and 1, and sum to 1. And because its entries sum to 1, $\mathbf{y}$ may also be interpreted as a *probabilistic* label. The cube $N_{fcu} = [0, 1]^3$ is called *unconstrained* fuzzy label vector space; vectors such as $\mathbf{z} = (.7, .2, .7)^T$ have each entry between 0 and 1, but are otherwise unrestricted.

Cluster Analysis. Given unlabeled data $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ in $\Re^p$, *clustering* in X is assignment of (hard or fuzzy) label vectors to the objects generating X. If the labels are hard, we hope that they identify c "natural subgroups" in X. Clustering is also called *unsupervised learning* , the word *learning* referring here to learning the correct labels (and possibly vector prototypes or quantizers) for "good" subgroups in the data. *c-partitions* of X are characterized as sets of (cn) values $\{u_{ik}\}$ satisfying some or all of the following conditions :

$$0 \le u_{ik} \le 1 \quad \forall \ i,k \quad ; \quad (2a)$$
$$0 < \Sigma u_{ik} < n \quad \forall \ i \quad ; \quad (2b)$$
$$\Sigma u_{ik} = 1 \quad \forall \ k \quad . \quad (2c)$$

Fig. 1. Hard, fuzzy and probabilistic label vectors (for c = 3 classes).


$\mathbf{e}_3 = \begin{pmatrix}0\\0\\1\end{pmatrix}$
$N_3 = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$
$\mathbf{y} = \begin{pmatrix}.1\\.6\\.3\end{pmatrix}$
$\mathbf{z} = \begin{pmatrix}.7\\.2\\.7\end{pmatrix}$
N_{f3}
$\mathbf{e}_2 = \begin{pmatrix}0\\1\\0\end{pmatrix}$
$N_{f3u} = [0,1]^3$
$\mathbf{e}_1 = \begin{pmatrix}1\\0\\0\end{pmatrix}$

Using equations (2) with the values $\{u_{ik}\}$ arrayed as a (cxn) matrix $U = [u_{ik}]$, we define:

$$M_{fcnu} = \{U \in \Re^{cn} \mid u_{ik} \text{ satisfies (2a) and (2b) } \forall\ i, k\} \quad ; \qquad (3a)$$

$$M_{fcn} = \{U \in M_{fcnu} \mid u_{ik} \text{ satisfies (2c) } \forall\ i \text{ and } k\}. \quad ; \qquad (3b)$$

$$M_{cn} = \{U \in M_{fcn} \mid u_{ik} = 0 \text{ or } 1\ \forall\ i \text{ and } k\} \quad . \qquad (3c)$$

Equations (3a), (3b) and (3c) define, respectively, the sets of unconstrained fuzzy, constrained fuzzy (or probabilistic), and crisp *c-partitions of X*. We represent clustering algorithms as mappings $\mathcal{A} : X \rightarrow M_{fcnu}$. Each column of U in M_{fcnu} (M_{fcn}, M_{cn}) is a label vector from N_{fcu} (N_{fc}, N_c). The reason these matrices are called *partitions* follows from the interpretation of u_{ik} as the *membership* of $\mathbf{x}_k$ in the i-th partitioning subset (cluster) of X. M_{fcnu} and M_{fcn} can be more realistic physical models than M_{cn}, for it is common experience that the boundaries between many classes of real objects (e.g., tissue types in magnetic resonance images) are in fact very badly delineated (i.e., really fuzzy) , so M_{fcnu} provides a much richer means for representing and manipulating data that have such structures. We give an example to illustrate hard and fuzzy c-partitions of X. Let X = $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ = {peach, plum, nectarine}, and let c=2. Typical 2-partitions of these three objects are shown in Table 1:

Table 1. 2-partitions of X = $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ = {peach, plum, nectarine}

	Hard $U_1 \in M_{23}$	Fuzzy $U_2 \in M_{f23}$	Fuzzy $U_3 \in M_{f23u}$
Object	$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$	$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$	$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$
Peaches *Plums*	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0.9 & 0.2 & 0.4 \\ 0.1 & 0.8 & 0.6 \end{bmatrix}$	$\begin{bmatrix} 0.9 & 0.5 & 0.5 \\ 0.6 & 0.8 & 0.7 \end{bmatrix}$

The nectarine, $\mathbf{x}_3$, is shown as the last column of each partition, and in the hard case, it must be (erroneously) given full membership in one of the two crisp subsets partitioning this data; in U_1 $\mathbf{x}_3$ is labeled "plum". Fuzzy partitions enable algorithms to (sometimes!) avoid such mistakes. The final column of the first fuzzy partition in Table 1 allocates most (0.6) of the membership of $\mathbf{x}_3$ to the plums class; but also assigns a lesser membership of 0.4 to $\mathbf{x}_3$ as a peach. The last partition in Table 1 illustrates an unconstrained set of membership assignments for the objects in each class. Columns like the one for the nectarine in the two fuzzy partitions serve a useful purpose - lack of strong membership in a single class is a signal to "take a second look". Hard partitions of data cannot suggest this. In the present case, the nectarine is an *hybrid* of peaches and plums, and the memberships shown for it in the last column of either fuzzy partition seem more plausible *physically* than crisp assignment of $\mathbf{x}_3$ to an incorrect class. It is appropriate to note that statistical clustering algorithms - e.g., unsupervised learning with maximum likelihood - also produce solutions in M_{fcn}. Fuzzy clustering began with Ruspini[8] ; see Bezdek and Pal[9] for a number of more recent papers on this topic. Algorithms that produce unconstrained fuzzy partitions of X are relatively new; for example, see the work of Krishnapuram and Keller[10].

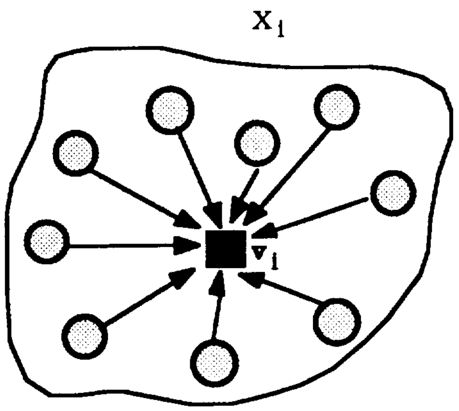
Prototype classification is illustrated in Figure 2. Basically, the vector $\mathbf{v}_i$ is taken as a prototypical representation for all the vectors in the hard cluster $X_i \subset X$. There are many synonyms for the word prototype in the literature: for example, quantizer (hence LVQ), signature, template, paradigm, exemplar. In the context of clustering, of course, we view $\mathbf{v}_i$ as the cluster center of hard cluster $X_i \subset X$. Each of the clustering algorithms discussed in this paper will produce a set of c prototype vectors V = $\{\mathbf{v}_k\}$ from any unlabeled or labeled input data set X in $\Re^p$. Once the prototypes are found (and possibly relabeled if the data have physical labels), they define a hard nearest prototype (NP) classifier, say $\mathbf{D}_{NP,V}$:

Crisp Nearest Prototype (1-NP) Classifier. Given prototypes $V = \{\mathbf{v}_k \mid 1 \le k \le c\}$ and $\mathbf{z} \in \Re^p$:

$$\text{Decide } \mathbf{z} \in i \Leftrightarrow \mathbf{D}_{NP,V}(\mathbf{z}) = \mathbf{e}_i \quad \Leftrightarrow \quad \|\mathbf{z} - \mathbf{v}_i\|_A \le \|\mathbf{z} - \mathbf{v}_j\|_A : 1 \le j \le c, j \ne i \tag{4}$$

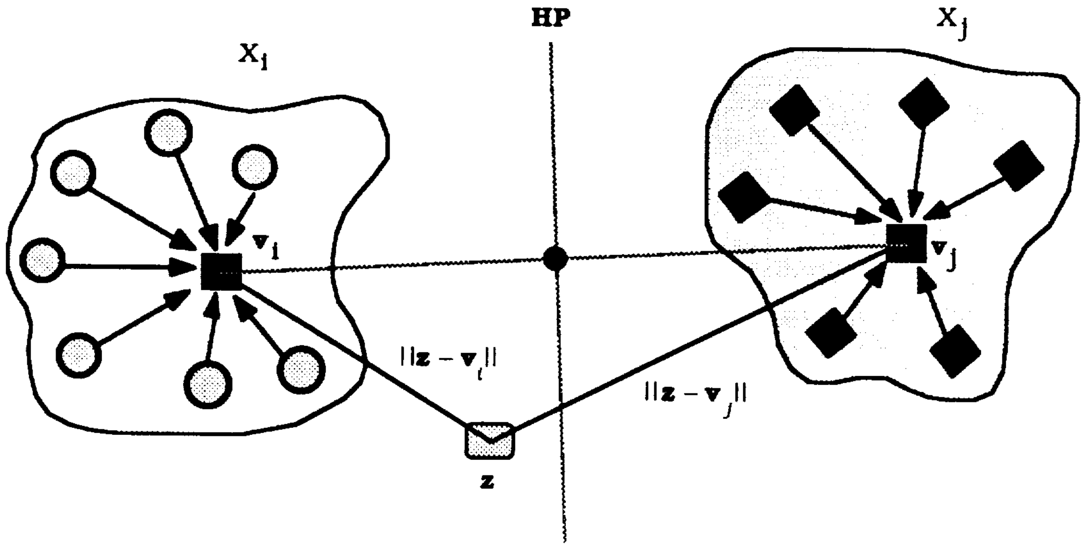
In (4) A is any *positive definite* pxp weight matrix - it renders the norm in (4) an inner product norm. That is, the distance from $\mathbf{z}$ to any $\mathbf{v}_i$ is computed as $\|\mathbf{z} - \mathbf{v}_i\|_A = \sqrt{(\mathbf{z} - \mathbf{v}_i)^T A(\mathbf{z} - \mathbf{v}_i)}$. Equation (4) defines a hard classifier, even though its parameters may come from a fuzzy algorithm. It would be careless to call $\mathbf{D}_{NP,V}$ a fuzzy classifier just because fuzzy c-means produced the prototypes, for example, because (4) can be implemented, and has the same geometric structure, using prototypes $\{\mathbf{v}_k\}$ from *any* algorithm that produces them. The $\{\mathbf{v}_k\}$ can be sample means of hard clusters (HCM); cluster centers of fuzzy clusters (FCM); weight vectors attached to the nodes in the competitive layer of a Kohonen clustering network (LVQ); or estimates of the (c) assumed mean vectors $\{\mu_k\}$ in maximum likelihood decomposition of mixtures.

Figure 2. Representation of many vectors by one prototype (vector quantizer).



The geometry of the 1-NP classifier is shown in Figure 3, using Euclidean distance for (4) - that is A=I, the pxp identity matrix. The 1-NP design erects a linear boundary halfway between and orthogonal to the line connecting the i-th and j-th prototypes, viz., the hyperplane HP through the vector $(\mathbf{v}_i - \mathbf{v}_j)/2$ perpendicular to it. All NP designs defined with inner product norms use (piecewise) linear decision boundaries of this kind.

Figure 3. Geometry of the Nearest Prototype Classifier for Inner Product Norms



Clustering algorithms imaged in M_{fcnu} eventually "defuzzify" or "deprobabilize" their label vectors, usually using the maximum membership (or maximum probability) strategy on the terminal fuzzy (or probabilistic) c-partitions produced by the data:

***Maximum membership* (MM) conversion** of U in M_{fcnu} to U_{MM} in M_{fc} :

$$u_{MM_{ik}} = \begin{cases} 1; & u_{ik} \geq u_{sk},\ 1 \leq s \leq c, s \neq i \\ 0; & \textit{otherwise} \end{cases} \qquad 1 \leq i \leq c;\ 1 \leq k \leq n \qquad . \qquad (5)$$

U_{MM} is always a hard c-partition; we use this conversion to generate a confusion matrix and error statistics when processing labeled data with FCM and FLVQ. For HCM/FCM/LVQ/FLVQ, using (5) instead of (4) with the terminal prototypes secured is fully equivalent- that is, U_{MM} *is* the hard partition that would be created by applying (5) with the final cluster centers to the unlabeled data. This is not true for GLVQ.

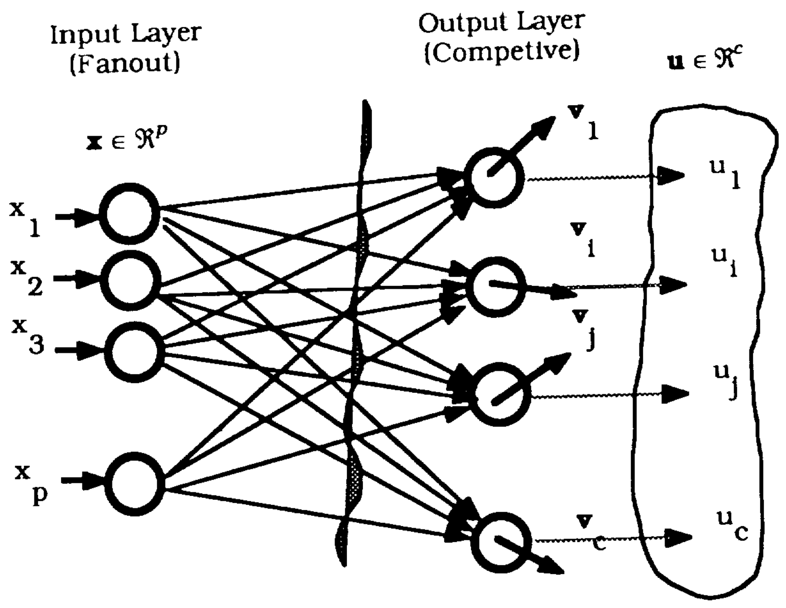
2. LEARNING VECTOR QUANTIZATION AND SEQUENTIAL HARD C-MEANS

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 using the 1-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 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. The primary goal of LVQ is representation of many points by a few prototypes; identification of clusters is implicit, but not active, in pursuit of this goal. We let $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n\} \subset \Re^p$ 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 Figure 5. The input layer of an LVQ network is connected directly to the output layer. Each node in the output layer has a weight vector (or prototype) attached to it. The prototypes $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c)$ are essentially a network array of (unknown) cluster centers, $\mathbf{v}_i \in \Re^p$ for $1 \le i \le c$. In this context the word *learning* refers to finding values for the $\{v_{ij}\}$. When an input vector $\mathbf{x}$ is submitted to this network, distances are computed between each $\mathbf{v}_r$ and $\mathbf{x}$. The output nodes "compete", a (minimum distance) "winner" node, say $\mathbf{v}_i$, is found; and it is then updated using one of several update rules.

Figure 5. LVQ Clustering Networks



We give a brief specification of LVQ as applied to the data in our examples. 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 $X = \{\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n\} \subset \Re^p$. Fix c, T, and $\varepsilon > 0$.
LVQ2. Initialize $\mathbf{V}_0 = (\mathbf{v}_{1,0}, ..., \mathbf{v}_{c,0}) \in \Re^{cp}$, and learning rate $\alpha_0 \in (1,0)$.
LVQ3. For t = 1,2, ..., T;
        For k = 1,2, ..., n:
            a. Find $\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\| = \min_{1 \le j \le c}\left\{\|\mathbf{x}_k - \mathbf{v}_{j,t-1}\|\right\}$.      (6)
            b. Update the winner : $\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \alpha_t(\mathbf{x}_k - \mathbf{v}_{i,t-1})$      (7)
        Next k
        d. Apply the 1-NP (nearest prototype) rule to the data :
$u_{LVQ_{ik}} = \begin{cases} 1; & \|\mathbf{x}_k - \mathbf{v}_i\| \le \|\mathbf{x}_k - \mathbf{v}_j\|, 1 \le j \le c, j \ne i \\ 0; & otherwise \end{cases}$, $1 \le i \le c$ and $1 \le k \le n$.      (8)
        e. Compute $E_t = \|\mathbf{V}_t - \mathbf{V}_{t-1}\|_1 = \sum_{r=1}^{c}\|\mathbf{v}_{r,t} - \mathbf{v}_{r,t-1}\|_1 = \sum_{k=1}^{n}\sum_{r=1}^{c}|\upsilon_{rk,t} - \upsilon_{rk,t-1}|$.
        f. If $E_t \le \varepsilon$ stop; Else adjust learning rate $\alpha_t$;
    Next t
```

The numbers $U_{LVQ} = \left[u_{LVQ_{ik}}\right]$ at (8) are a cxn matrix that define a hard c-partition of X using the 1-NP classifier assignment rule shown in (4). The vector **u** shown in Figure 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, non-iteratively, 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 (7) , since $\nabla_{\mathbf{v}}(\|\mathbf{x} - \mathbf{v}\|_I^2) = -2I(\mathbf{x} - \mathbf{v}) = -2(\mathbf{x} - \mathbf{v})$. The origin of this rule comes about by assuming that each $\mathbf{x} \in \Re^p$ is distributed according to a probability density function $f(\mathbf{x})$. LVQ's objective is to find a set of $\mathbf{v}_i$'s such that the expected value of the square of the discretization error is minimized :

$$E\left(\|\mathbf{x}-\mathbf{v}_i\|^2\right)=\int\int\underset{\Re^p}{\ldots}\int\|\mathbf{x}-\mathbf{v}_i\|^2 f(\mathbf{x})d\mathbf{x} \quad . \tag{9}$$

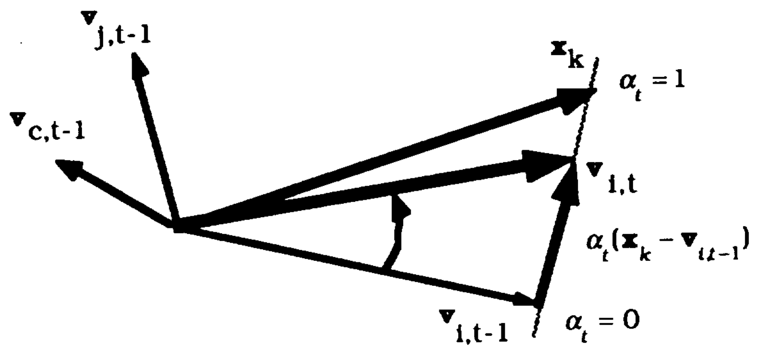
In this expression $\mathbf{v}_i$ is the winning prototype for each $\mathbf{x}$, and will of course vary as $\mathbf{x}$ ranges over $\Re^p$. A sample function of the optimization problem is $e=\|\mathbf{x}-\mathbf{v}_i\|^2$. An optimal set of $\mathbf{v}_i$'s can be approximated by applying local gradient descent to a finite set of samples drawn from f. The extant theory for this scheme is contained in Kohonen[12], which states that LVQ converges in the sense that the prototypes $\mathbf{V}_t=(\mathbf{v}_{1,t}, \mathbf{v}_{2,t}, \ldots, \mathbf{v}_{c,t})$ generated by the LVQ iterate sequence converge, i.e., $\{\mathbf{V}_t\}\xrightarrow{t\to\infty}\hat{\mathbf{V}}$, provided two conditions are met by the sequence $\{\alpha_t\}$ of learning rates used in (7) :

$$\sum_{t=0}^{\infty}\alpha_t=\infty \quad : \quad \text{and} \tag{10a}$$

$$\sum_{t=0}^{\infty}\alpha_t^2<\infty \quad . \tag{10b}$$

One choice for the learning rates that satisfies these conditions is the harmonic sequence $\alpha_t=1/t$ for t ≥ 1; $\alpha_0\in(0,1)$. Kohonen has shown that (under some assumptions) steepest descent optimization of the average expected error function (9) is possible, and leads to the update rule (7). The update scheme shown in equation (7) has the simple geometric interpretation shown in Figure 6.

Figure 6. Updating the winning LVQ Prototype.



The winning prototype $\mathbf{v}_{i,t-1}$ is simply rotated towards the current data point by moving along the vector $(\mathbf{x}_k - \mathbf{v}_{i,t-1})$ which connects it to $\mathbf{x}_k$. The amount of shift depends on the value of a "learning rate" parameter α_t, which varies from 0 to 1. As seen in Figure 2, there is no update if $\alpha_t=0$, and when $\alpha_t=1$, $\mathbf{v}_{i,t}$ becomes $\mathbf{x}_k$ ($\mathbf{v}_{i,t}$ is just a convex combination of $\mathbf{x}_k$ and $\mathbf{v}_{i,t-1}$). This process continues until termination via LVQ3.f, at which time the terminal prototypes yield a "best" hard c-partition of X via (3).

Comments on LVQ :

1. **Limit point property** : Kohonen[12] refers to [13,14], and mentions that LVQ converges to a unique limit if and only if conditions (10) 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 $\{\mathbf{V}_t\} \xrightarrow{t\to\infty} \hat{\mathbf{V}}$ does not insure that the limit vector $\hat{\mathbf{V}}$ 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 **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 SHCM algorithm is similar to LVQ[15]. In MacQueen's algorithm the weight vectors are initialized with the first c samples in the data set X. In other words, $\mathbf{v}_{r,0} = \mathbf{x}_r$, $r=1,\dots,c$. Let $q_{r,0}=1$ for $r=1,\dots,c$ ($q_{r,t}$ represents the number of samples that have so far been used to update $\mathbf{v}_{r,t}$). Suppose $\mathbf{x}_{t+1}$ is a new sample point such that $\mathbf{v}_{i,t}$ is closest (with respect to, and without loss, the Euclidean metric) to it. MacQueen's algorithm updates the $\mathbf{v}_r$'s as follows (again, index i identifies the winner at this t):

$$\mathbf{v}_{i,t+1} = (\mathbf{v}_{i,t}\, q_{i,t} + \mathbf{x}_{t+1})/(q_{i,t}+1) \quad ; \quad (11a)$$
$$q_{i,t+1} = q_{i,t}+1 \quad ; \quad (11b)$$
$$\mathbf{v}_{r,t+1} = \mathbf{v}_{r,t} \quad \text{for } r \neq i, \quad ; \quad (11c)$$
$$q_{r,t+1} = q_{r,t} \quad \text{for } r \neq i. \quad . \quad (11d)$$

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 (11a), one can rewrite Macqueen's update equation :

$$\mathbf{v}_{i,t+1} = \mathbf{v}_{i,t} + (\mathbf{x}_{t+1} - \mathbf{v}_{i,t}) / q_{i,t+1} . \tag{12}$$

Writing $1/q_{i,t+1}$ as $\alpha_{i,t+1}$, equation (12) takes exactly the same form as equation (7) . 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 (other variants of this algorithm pass through the data set many times[16]). (ii) In MacQueen's algorithm $\alpha_{i,t+1}$ is inversely proportional to the number of points found closest to $\mathbf{v}_{i,t}$, 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 $\Re^p$ into c subregions, say $(S_1,\ldots,S_c)$, in such a way as to minimize the functional

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

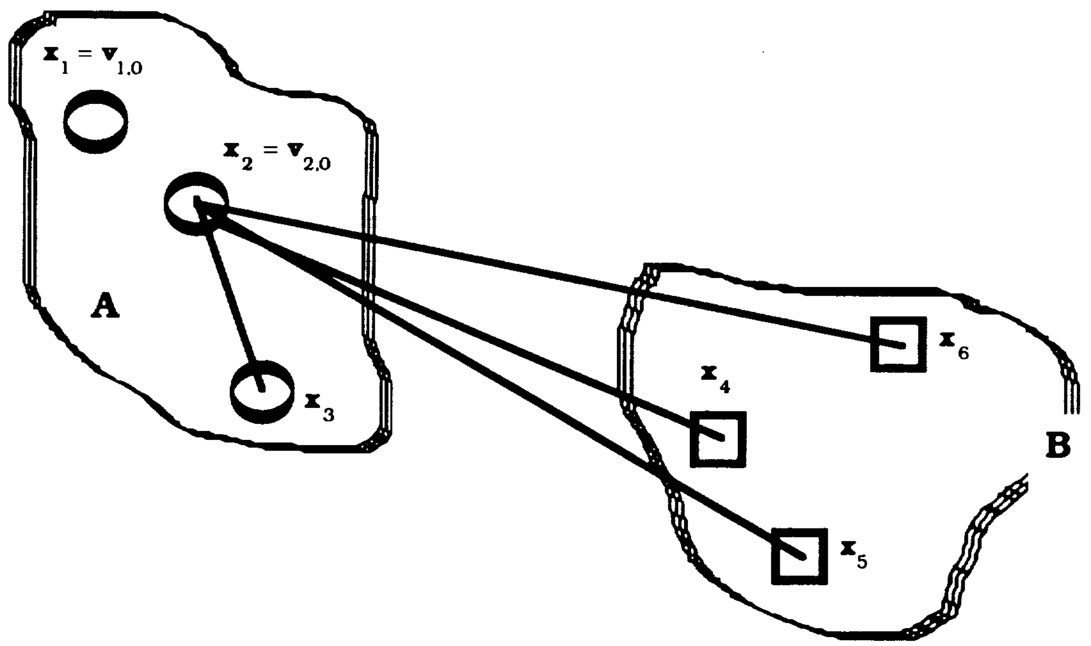
where f is a density function as in LVQ, and $\hat{\mathbf{v}}_i$ is the (conditional) mean of the pdf f_i obtained by restricting f to S_i, normalized in the usual way, i.e., $f_i(\mathbf{x}) = f(\mathbf{x}) |_{S_i} / P(S_i)$; and $\hat{\mathbf{V}} = (\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_c) \in \Re^{cp}$. Let $\mathbf{V}_t = (\mathbf{v}_{1,t},\ldots,\mathbf{v}_{c,t})$; $S_t = (S_1(\mathbf{v}_t),\ldots,S_c(\mathbf{v}_t))$ be the minimum distance partition relative to $\mathbf{v}_t$; $P(S_j) = \text{prob}(\mathbf{x} \in S_j)$, $P_{j,t} = P(S_j(\mathbf{v}_t)) = \text{prob}(\mathbf{x} \in S_j(\mathbf{v}_t))$; and $\hat{\mathbf{v}}_{j,t}$, the conditional mean of $\mathbf{x}$ over $S_j(\mathbf{v}_t)$, is $\hat{\mathbf{v}}_{j,t} = \int_{S_j(\mathbf{v}_t)} \mathbf{x} df(\mathbf{x})/P(S_j)$ when $P(S_j) > 0$, or $\hat{\mathbf{v}}_{j,t} = \mathbf{v}_{j,t}$ when $P(S_j) = 0$. MacQeen proved that for the algorithm described by equations (11a-d) ,

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

Since { $\hat{\mathbf{v}}_j$ } are conditional means, the partition obtained by applying the nearest prototype labeling method at (4) 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 suffer from a common problem that can be quite serious. Suppose the input data $\mathbf{X} = \{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3,\mathbf{x}_4,\mathbf{x}_5,\mathbf{x}_6\} \subset \Re^2$ contains the two classes $\mathbf{A} = \{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3\}$ and $\mathbf{B} = \{\mathbf{x}_4,\mathbf{x}_5,\mathbf{x}_6\}$ as shown in Figure 7. The initial positions of the centroids $\mathbf{v}_{1,0}$ and $\mathbf{v}_{2,0}$ are also depicted in Figure 7. Since the initial centroid for class 2 ($\mathbf{v}_{2,0}$) is closer to the remaining four input points than $\mathbf{v}_1$, each of them will update (modify) $\mathbf{v}_2$ only; $\mathbf{v}_1$ will not be changed on the first pass through the data. Moreover, both update schemes result in the updated centroid being pulled towards the data point some distance along the line joining the two points. Consequently, the chance for $\mathbf{v}_{1,0}$ to get updated on succeeding passes is very low. Although this results in a locally optimal solution, it is hardly a desirable one.

Figure 7. An initialization problem for LVQ/SHCM


$\mathbf{x}_1 = \mathbf{v}_{1,0}$
$\mathbf{x}_2 = \mathbf{v}_{2,0}$
A
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$
$\mathbf{x}_6$
B

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 $\mathbf{v}_i$'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[17] attempted to solve problem (ii) by updating the winner and some of its neighbors (not topological, but metrical neighbors in $\Re^p$) with each input in FLVQ. In their approach, the learning coefficient was reduced both with time and distance from the winner. FLVQ, in turn, raised general 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 motivated the development of the GLVQ algorithm.

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 $\mathbf{v}_{r,t} \in \Re^p$ is associated with a display node ,say $\mathbf{d}_{r,t} \in \Re^2$. The vector $\mathbf{v}_{i,t}$ that best matches (in the sense of minimum Euclidean distance in the feature space) an incoming input vector $\mathbf{x}_k$ is then identified as in (4). $\mathbf{v}_{i,t}$ has an "image" $\mathbf{d}_{i,t}$ in display space. Next, a topological (spatial) neighborhood $\mathcal{N}(\mathbf{d}_{i,t})$ centered at $\mathbf{d}_{i,t}$ is defined in display space, and its display node neighbors are located. Finally, the vector $\mathbf{v}_{i,t}$ and other prototype vectors in the inverse image $[\mathcal{N}(\mathbf{d}_{i,t})]^{-1}$ of spatial neighborhood $\mathcal{N}(\mathbf{d}_{i,t})$ are updated using a generalized form of update rule (7) :

$$\mathbf{v}_{r,t} = \mathbf{v}_{r,t-1} + \alpha_{rk,t}(\mathbf{x}_k - \mathbf{v}_{r,t-1}), \qquad \mathbf{d}_{r,t} \in \mathcal{N}(\mathbf{d}_{i,t}). \tag{13}$$

The function $\alpha_{rk,t}$ defines a *learning rate distribution* on indices (r) of the nodes to be updated for each input vector $\mathbf{x}_k$ at each iterate t. These numbers *impose* (by their definition) a sense of the strength of interaction between (output) nodes. If the $\{\mathbf{v}_{r,t}\}$ are initialized with random values and the external inputs $\mathbf{x}_k = \mathbf{x}_k(t)$ are drawn from a time invariant probability density function $f(\mathbf{x})$, then the point density function of $\mathbf{v}_{r,t}$ (the number of $\mathbf{v}_{r,t}$'s in the ball $\mathbf{B}(\mathbf{x}_k,\varepsilon)$ centered at the point $\mathbf{x}_k$ with radius ε) tends to approximate $f(\mathbf{x})$. It has also been shown that the $\mathbf{v}_{r,t}$'s attain their values in an "orderly fashion" according to $f(\mathbf{x})$[12]. 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.

4. GENERALIZED LEARNING VECTOR QUANTIZATION (GLVQ)

In this section we describe a new clustering algorithm which avoids or fixes several of the limitations mentioned earlier. The learning rules are derived from an optimization problem. Let $\mathbf{x} \in \Re^p$ be a stochastic input vector distributed according to a time invariant probability distribution $f(\mathbf{x})$, and let i be the best matching node as in (7). Let $L_{\mathbf{x}}$ be a loss function which measures the locally weighted mismatch (error) of $\mathbf{x}$ with respect to the winner :

$$L_{\mathbf{x}} = L(\mathbf{x} \ ; \ \mathbf{v}_1, \ldots, \mathbf{v}_c) = \sum_{r=1}^{c} g_{ir} \left\| \mathbf{x} - \mathbf{v}_r \right\|^2 \qquad \text{, where} \qquad (14a)$$

$$g_{ir} = \left\{ \begin{array}{ll} 1 & if \quad r = i \\ \dfrac{1}{\sum_{j=1}^{c} \left\| \mathbf{x} - \mathbf{v}_j \right\|^2} & , \ otherwise \end{array} \right\} . \qquad (14b)$$

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

$$\text{Minimize} \ : \quad \Gamma(\mathbf{V}) = \int\int \ldots \int_{\Re^p} \sum_{r=1}^{c} g_{ir} \left\| \mathbf{x} - \mathbf{v}_r \right\|^2 f(\mathbf{x}) d\mathbf{x} \qquad (15)$$

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

$$\text{Minimize} \ : \quad \Gamma(\mathbf{V}) = \frac{\sum_{t=1}^{n} \sum_{r=1}^{c} g_{ir} \left\| \mathbf{x}_t - \mathbf{v}_r \right\|^2}{n} \ . \qquad (16)$$

Here $L_{\mathbf{x}}$ is a random functional for each realization of $\mathbf{x}$, and $\Gamma(\mathbf{V})$ 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 $\mathbf{x}$ and all of $\mathbf{v}_r$ s. The function $L_{\mathbf{x}}$ is well defined. If we assume that $\mathbf{x}$ has a unique distance from each $\mathbf{v}_r$, then i and g_{ir} are uniquely determined, and hence $L_{\mathbf{x}}$ 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 $L_{\mathbf{x}}$, exists. As most learning algorithms do[18], we approximate the gradient of $\Gamma(\mathbf{V})$ by the gradient of the sample function $L_{\mathbf{x}}$. In other words, We attempt to minimize Γ by local gradient descent search using the sample function $L_{\mathbf{x}}$. It is our conjecture that the optimal values of $\mathbf{v}_r$'s can be approximated in an iterative, stepwise fashion by moving in the direction of gradient of $L_{\mathbf{x}}$. The algorithm is derived as follows (for notational simplicity the subscript for $\mathbf{x}$ will be ignored). First rewrite L as :

$$L = \sum_{r=1}^{c} g_{ir} \|\mathbf{x} - \mathbf{v}_r\|^2 = \|\mathbf{x} - \mathbf{v}_i\|^2 + \sum_{\substack{r=1 \\ r\neq i}}^{c} \|\mathbf{x} - \mathbf{v}_r\|^2 \Big/ \sum_{j=1}^{c} \|\mathbf{x} - \mathbf{v}_j\|^2$$

$$= \|\mathbf{x} - \mathbf{v}_i\|^2 + \sum_{r=1}^{c} \|\mathbf{x} - \mathbf{v}_r\|^2 \Big/ \sum_{j=1}^{c} \|\mathbf{x} - \mathbf{v}_j\|^2 - \|\mathbf{x} - \mathbf{v}_i\|^2 \Big/ \sum_{j=1}^{c} \|\mathbf{x} - \mathbf{v}_j\|^2$$

$$= \|\mathbf{x} - \mathbf{v}_i\|^2 + 1 - \|\mathbf{x} - \mathbf{v}_i\|^2 \Big/ \sum_{j=1}^{c} \|\mathbf{x} - \mathbf{v}_j\|^2 . \tag{17}$$

Differentiating L with respect $\mathbf{v}_i$ yields (after some algebraic manipulations) :

$$\nabla_{\mathbf{v}} L(\mathbf{v}_i) = -2(\mathbf{x} - \mathbf{v}_i) \frac{D^2 - D + \|\mathbf{x} - \mathbf{v}_i\|^2}{D^2} \tag{18}$$

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

$$\nabla_{\mathbf{v}} L(\mathbf{v}_j) = -2(\mathbf{x} - \mathbf{v}_j) \frac{\|\mathbf{x} - \mathbf{v}_i\|^2}{D^2} \tag{19}$$

Update rules based on (17) and (18) are :

$$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \alpha_t \, (\mathbf{x} - \mathbf{v}_{i,t-1}) \frac{D^2 - D + \|\mathbf{x} - \mathbf{v}_{i,t-1}\|^2}{D^2} \quad \text{for the winner node i, and} \tag{20}$$

$$\mathbf{v}_{j,t} = \mathbf{v}_{j,t-1} + \alpha_t \, (\mathbf{x} - \mathbf{v}_{j,t-1}) \frac{\|\mathbf{x} - \mathbf{v}_{i,t-1}\|^2}{D^2} \quad \text{for the other (c-1) nodes, } j \neq i . \tag{21}$$

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 $\Sigma\, \alpha_t \to \infty$)[19]. On the other hand, in the presence of noise, under a suitable assumption about subgradients, the search becomes successful if the conditions in (10) are satisfied. We recommend a decreasing sequence of α_t ($0 < \alpha_t < 1$) satisfying (10) , which insure that α_t is neither reduced too fast nor too slow. From the point of view of learning, the 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*)[20]. Condition (10a) enables plasticity, while (10b) enforces stability . In other words, an incoming input should not affect the parameters of a learning system too strongly, thereby enabling it to 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 $\alpha_0(1\text{-}t/T)$, 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 (20), 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 $\mathbf{x}$ and the winner node $\mathbf{v}_i$ 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 $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n\} \subset \Re^p$. Fix c, T, and $\varepsilon > 0$.
GLVQ2. Initialize $\mathbf{V}_0 = (\mathbf{v}_{1,0}, \ldots, \mathbf{v}_{c,0}) \in \Re^{cp}$, and learning rate $\alpha_0 \in (1,0)$ .
GLVQ3. For t = 1, 2, ...., T.
        a. Compute $\alpha_t = \alpha_0\,(1\text{-}t/T)$ .
            While $k \leq n$
            b. Find $\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\| = \min_{1\leq j\leq c}\left\{\|\mathbf{x}_k - \mathbf{v}_{j,t-1}\|\right\}$.
            c. Update all (c) weight vectors $\{\mathbf{v}_{r,t}\}$ with
               $\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \alpha_t\,(\mathbf{x}_k - \mathbf{v}_{i,t-1})\,\dfrac{D^2 - D + \|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|^2}{D^2}$ , $D = \sum_{r=1}^{c}\|\mathbf{x} - \mathbf{v}_r\|^2$
```

$$\mathbf{v}_{r,t} = \mathbf{v}_{r,t-1} + \alpha_t \, (\mathbf{x}_k - \mathbf{v}_{r,t-1}) \frac{\left\|\mathbf{x}_k - \mathbf{v}_{r,t-1}\right\|^2}{D^2} \quad (r \neq i) \qquad , D = \sum_{r=1}^{c} \left\|\mathbf{x} - \mathbf{v}_r\right\|^2$$

```
        Wend
        d. Compute $\left\|\mathbf{V}_t - \mathbf{V}_{t-1}\right\| = \sum_{r=1}^{c}\left\|\mathbf{v}_{r,t} - \mathbf{v}_{r,t-1}\right\|_1 = \sum_{k=1}^{n}\sum_{r=1}^{c}\left|\upsilon_{rk,t} - \upsilon_{rk,t-1}\right|$.
        e. If $E_t \le \varepsilon$ stop; Else
    Next t.
```

GLVQ4. Compute non-iteratively the nearest prototype GLVQ c-partition of X :

$$u_{GLVQ_{ik}} = \begin{cases} 1; & \left\|\mathbf{x}_k - \mathbf{v}_i\right\| \le \left\|\mathbf{x}_k - \mathbf{v}_j\right\| , 1 \le j \le c, j \ne i \\ 0; & otherwise \end{cases} \qquad ,1 \le i \le c \text{ and } 1 \le k \le n.$$

Comments on GLVQ :

1. There is no need to choose an update neighborhood .

2. Reduction of the learning coefficient with distance (either topological or in $\Re^p$) from the winner node is not required. Instead, reduction is done automatically and adaptively 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 the impact to weight vectors associated with other nodes. Quantization error is the error 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.

4. FUZZY LEARNING VECTOR QUANTIZATION (FLVQ)

Huntsberger and Ajjimarangsee[11] used SOFMs to develop clustering algorithms. Algorithm 1 in [11] 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 data point to that class. A second algorithm in their paper used the necessary conditions for FCM to assign a membership value in [0,1] to each data point. Specifically, Huntsberger and Ajjimarangsee suggested fuzzification of LVQ by replacing the learning rates $\{\alpha_{ik,t}\}$ usually found in rules such as (7) with fuzzy membership values $\{u_{ik,t}\}$ computed with the FCM formula [2]:

$$\alpha_{ik,t} = u_{ik,t} = \left(\sum_{j=1}^{c} \frac{D_{ik,t}}{D_{jk,t}} \right)^{\frac{-2}{m-1}} \quad . \qquad (22)$$

where $D_{ik,t} = \left\| \mathbf{x}_k - \mathbf{v}_{i,t} \right\|_A$. Numerical results reported in Huntsberger and Ajjimarangsee suggest that in many cases their algorithms and standard LVQ produce very similar answers. Their scheme was a partial integration of LVQ with FCM that showed some interesting results. However, it fell short of realizing a *model* for LVQ clustering; and no properties regarding terminal points or convergence were established. Moreover, since the objective of these LVQ is to find cluster centroids (prototypes), and hence clusters, there is no need to have a topological ordering of the weight vectors. Consequently, the approach taken in [11] seems to mix two objectives, feature mapping and clustering, and the overall methodology is difficult to interpret in either sense.

Integration of FCM with LVQ can be more fully realized by defining the learning rate for Kohonen updating as :

$$\alpha_{ik,t} = (u_{ik,t})^{m_t} = \left(\sum_{j=1}^{c} \frac{D_{ik,t}}{D_{jk,t}} \right)^{\frac{-2m_t}{m_t-1}} \quad , \qquad \text{where} \qquad (23a)$$

$$m_t = m_0 + t[(m_f - m_0)/T] = m_0 + t\Delta m \ ; \qquad m_f, m_0 \geq 1; \qquad t=1,2,\ldots T. \qquad (23b)$$

m_t replaces the (fixed) parameter m in (22). This results in three families of *Fuzzy LVQ* or FLVQ algorithms, the cases arising by different treatments of paramerer m_t. In particular, for

$t \in \{1,2,\ldots,T\}$, we have three cases depending on the choice of the initial (m_0) and final (m_f) values of m:

1.	$m_0 > m_f \Rightarrow \{m_t\} \downarrow m_f$	: Descending FLVQ	(24a)
2.	$m_0 < m_f \Rightarrow \{m_t\} \uparrow m_f$	: Ascending FLVQ	(24b)
3.	$m_0 = m_f \Rightarrow m_t \equiv m_0 \equiv m$	: FLVQ ≡ FCM	(24c)

Cases 1 and 3 are discussed at length by Bezdek et. al.[17]. Case 2 is fully discussed in Tsao et. al.[21]. Equation (24c) asserts that when $m_0 = m_f$, FLVQ reverts to FCM; this results from defining the learning rates via (23a), and using them in FLVQ3.b below. FLVQ is not a direct generalization of LVQ because it does not revert to LVQ in case all of the $u_{ik,t}$'s are either 0 or 1 (the crisp case). Instead, if $m_0 = m_f = 1$, FCM reverts to HCM, and the HCM update formula, which *is* driven by finding unique winners, as is LVQ, is a different formula than (7). FLVQ is perhaps the closest possible link between LVQ and c-Means type algorithms. We provide a formal description of FLVQ :

Fuzzy LVQ (FLVQ)

```
FLVQ1. Given unlabeled data set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$. Fix c, T, $\| \|_A$ and $\varepsilon > 0$.
FLVQ2. nitialize $\mathbf{v}_0 = (\mathbf{v}_{1,0}, \ldots, \mathbf{v}_{c,0}) \in \Re^{cp}$. Choose $m_0, m_f \geq 1$.
FLVQ3. For t = 1, 2, ..., T.
           a. Compute all (cn) learning rates $\{\alpha_{ik,t}\}$ with (23).
           b. Update all (c) weight vectors $\{\mathbf{v}_{i,t}\}$ with $\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \sum_{k=1}^{n} \alpha_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,t-1}) / \sum_{s=1}^{n} \alpha_{is,t}$
           c. Compute $E_t = \|\mathbf{v}_t - \mathbf{v}_{t-1}\| = \sum_{i=1}^{c} \|\mathbf{v}_{i,t} - \mathbf{v}_{i,t-1}\|$.
           d. If $E_t \leq \varepsilon$ stop; Else
     Next t.
```

For fixed c, $\{\mathbf{v}_{i,t}\}$ and m_t, the learning rates $\alpha_{ik,t} = (u_{ik,t})^{m_t}$ at (23a) satisfy the following :

$$\alpha_{ik,t} = (u_{ik,t})^{m_t} = \left(\frac{\kappa}{D_{ik,t}}\right)^{\frac{2m_t}{m_t - 1}} \qquad , \tag{25}$$

where κ is a positive constant. Apparently the contribution of $\mathbf{x}_k$ to the next update of the node weights is inversely proportional to their distances from it. The "winner" in (29) is the $\mathbf{v}_{i,t-1}$

closest to $\mathbf{x}_k$, and it will be moved further along the line connecting $\mathbf{v}_{i,t-1}$ to $\mathbf{x}_k$ than any of the other weight vectors. Since $\sum u_{ik,t} = 1 \Rightarrow \sum \alpha_{ik,t} \le 1$, this amounts to distributing partial updates across all c nodes for each $\mathbf{x}_k \in X$. This is in sharp contrast to LVQ, where only the winner is updated for each data point.

In *descending* FLVQ (24a), for large values of m_t (near m_0), all c nodes are updated with lower individual learning rates, and as $m_t \to 1$, more and more of the update is given to the "winner" node. In other words, the lateral distribution of learning rates is a function of t, which in the descending case "sharpens" at the winner node (for each $\mathbf{x}_k$) as $m_t \xrightarrow{+} 1$. Finally, we note again that for fixed m_t, FLVQ updates the $\{\mathbf{v}_{i,t}\}$ using the conditions that are necessary for FCM; each step of FLVQ is one iteration of FCM.

Figure 8. Updating Feature Space Prototypes in FLVQ Clustering Nets.

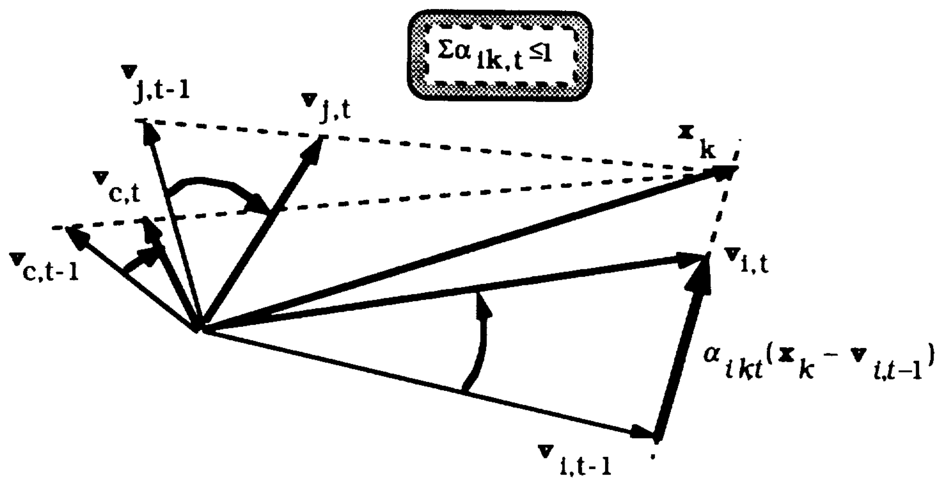


Figure 8 illustrates the update geometry of FLVQ; note that *every* node is (potentially) updated at every iteration, and the sum of the learning rates is always less than or equal to one.

Comments on FLVQ :

1. There is no need to choose an update neighborhood .

2. Reduction of the learning coefficient with distance (either topological or in $\Re^p$) from the winner node is not required. Instead, reduction is done automatically and adaptively by the learning rules.

3. The greater the mismatch to the winner (i.e., the higher the quantization error), the *smaller* the impact to the weight vectors associated with other nodes (recall (25) and (2c)). This is directly opposite to the situation in GLVQ.

4.The learning process attempts to minimize a well-defined objective function (stepwise).

5. 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.

6. This procedure depends on generation of a fuzzy c-partition of the data, so it is an iterative clustering model - indeed, stepwise, it is exactly fuzzy c-means [17].

5. IMAGE SEGMENTATION WITH GLVQ AND FLVQ

In this section we illustrate the (FLVQ and GLVQ) algorithms with image segmentation, which can be achieved either by finding spatially compact homogeneous regions in the image; or by detecting boundaries of regions, i.e., detecting the edges of each region. We have applied our clustering strategies to both paradigms. Image segmentation by clustering raises the important issue of feature extraction / selection. Generally, features relevant for identifying compact regions are different from those useful for the edge detection approach.

Feature selection for homogeneous region extraction

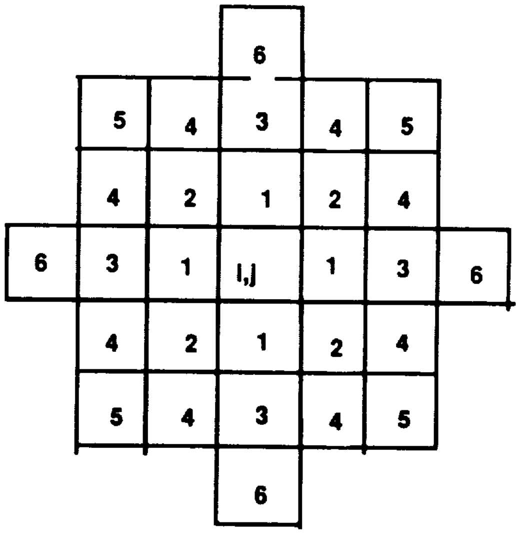
When looking for spatially compact regions, feature vectors should incorporate information about the spatial distribution of gray values. For pixel (i,j) of a digital image $F = \{(i,j) \mid 1 \le i \le M ; 1 \le j \le N\}$, we define the d^{th} *order neighborhood* of (i,j), where $d > 0$ is an integer as ;

$$N^d_{i,j} = \{(k,l) \in F\} \text{ such that } (i,j) \notin N^d_{i,j} \text{ and if } (k,l) \in N^d_{i,j} \text{ then } (i,j) \in N^d_{k,l} . \qquad (26)$$

Several such neighborhoods are depicted in Figure 9, where $N^d_{i,j}$ consists of all pixels marked with an index $\le d$. For example N^1 is obtained by taking the four nearest neighbor pixels to (i,j). Similarly, N^2 is defined by its eight nearest neighbors, and so on. $N^d_{i,j}$ as defined in (26) is the standard neighborhood definition for modeling digital images using Gibbs or Markov Random Fields. To define feature vectors for segmentation, we extend the definition of a d-th order neighborhood at (26) to include the center pixel (i,j):

$$N^{d'}_{i,j} = N^d_{i,j} \cup \{(i,j)\} \qquad ; \qquad D_{ij} = \left|N^{d'}_{i,j}\right| \qquad (27)$$

Figure 9 . An Ordered Neighborhood system



Next, let L= {1,2,..., G} be the set of gray values that can be taken by pixels in the image, and let f(i,j) be the intensity at (i,j) in F, that is, $f: F \mapsto L$. We define the collection of gray values of all pixels that belong to $N^{d'}_{i,j}$ as:

$$S^{d}_{i,j} = \{ f(k,l) | (k,l) \in N^{d'}_{i,j} \} \qquad . \tag{28}$$

Note that $S^{d}_{i,j}$ may contain the same gray value more than once. We say two neighborhoods $N^{d'}_{i,j}$ and $N^{d'}_{k,l}$ are *equally homogeneous* in case $S^{d}_{i,j}$ and $S^{d}_{k,l}$ are identical up to a permutation. This assumption is natural and useful as long as the neighborhood size is small. To see this, consider two 100x100 neighborhoods that contain 5000 pixels with gray value 1 and 5000 with value G. Satisfaction of this property gives the impression of two perfectly homogeneous regions ; but in fact one of these neighborhoods might have all 5000 pixels of each intensity in, say, the upper and lower halves of the image, while other neighborhood has a completely random mixture of black and white spots. When the neighborhood size is small, however, spatial rearrangement of a few gray values among many more in the entire image will not create a much different impression to the human visual system as far as homogeneity of the region is concerned. Therefore, for small values of d we can derive features for (i,j) from $S^{d}_{i,j}$ which are relatively independent of permutation of its elements (typically, such features might include the mean, standard deviation, etc. of the intensity values in $S^{d}_{i,j}$). Subsequently, these features are arrayed into a *pixel vector* $\mathbf{x}_{ij}$ for each pixel. In this investigation, we used the gray values in $S^d_{i,j}$ themselves as the feature vector for pixel (i,j); thus, each (i,j) in F (excluding boundaries) is associated with $\mathbf{x}_{ij}$ in $\Re^{D'}$.

Since FLVQ and GLVQ both use distances between feature vectors, we *sorted* the values in $S^d_{i,j}$ to get each $\mathbf{x}_{ij}$. Sorting can be done either in ascending or in descending order, but the same strategy must be used for all pixels. We remark that an increase in the d-size of the neighborhood will obscure finer details in the segmented image; conversely, a very low value of d usually results in too many small regions. Experimental investigation suggests that $3 \le d \le 5$ provides a reasonable tradeoff between fine and gross structure.

Feature selection for edge extraction

Loosely speaking edges are regions of abrupt changes in gray values. Therefore, features used for extraction of homogeneous regions are not suitable for edge-nonedge classification. For this approach, we nominate a feature vector $\mathbf{x}_{ij}$ in $\Re^3$ with three components : standard deviation, gradient 1 and gradient 2. In other words, each pixel is represented by a 3-tuple $\mathbf{x}_{ij}$ = $(\sigma(i,j), G1(i,j), G2(i,j))$. The standard deviation is defined on $S^d_{i,j}$ as follows:

$$\sigma(i,j) = \{\frac{1}{|S^d_{i,j}|} \sum_{g \in S^d_{i,j}} (g - \mu_{i,j})^2\}^{1/2} \quad . \tag{29}$$

where $\mu_{i,j}$ is the average gray value over $S^d_{i,j}$. Since standard deviation measures variation of gray values over the neighborhood, using too large a neighborhood will destroy its utility for edge detection. The two gradients are defined as :

$$G1(i,j) = |f_{i+1,j} - f_{i-1,j}| + |f_{i,j-1} - f_{i,j+1}| \quad \text{; and} \tag{30}$$
$$G2(i,j) = |f_{i+1,j+1} - f_{i-1,j-1}| + |f_{i+1,j-1} - f_{i-1,j+1}| . \tag{31}$$

Note that G1 measures intensity changes in the horizontal and vertical directions, while G2 takes into account diagonal edges; this justifies the use of both G1 and G2.

Implementation

FLVQ (ascending strategy) and GLVQ were used for segmentation of the house image depicted in Figure 10(a). This image is a very complex image for segmentation into homogeneous regions, because it has some textured portions (the trees) behind the house. For the region extraction scheme we used neighborhoods of order d=3 and d=5. The number of classes chosen was c=8. The computing protocols used for different runs are summarized in Table 2.

Table 2. Computing protocols for the segmentations

Since FLVQ produces fuzzy labels for each pixel vector, the fuzzy label vector is defuzzified using the maximum membership rule at (5). Thus, each pixel receives a crisp label corresponding to one of the c classes in the segmented image. Coloring of the segmented image is done by using c distinct gray values, one for each class. Defuzzification is not required for the GLVQ algorithm as it produces hard labels.

Figure 10 contains some typical outputs of both FLVQ and GLVQ using the region-based segmentation approach. To show the effect of sorting we ran both algorithms with unsorted and sorted feature vectors. Figure 10(b) represents the segmented output produced by FLVQ with d=3 and unsorted features; while figure 10(c) displays the output under the same conditions, but with sorted features. Comparing figures 10(b) and (c) one sees that the noisy patches on the roof of the house that appear in Fig. 10(b) are absent in Fig. 10(c). Similar occurences can be found in other portions of the image. This demonstrates that sorted pixel vectors seem to afford some noise cleaning ability. Figure 10(d) was produced with FLVQ using sorted neighborhoods of size 5. Note that the textured tree areas have been segmented more compactly; this illustrates the effect of increasing the neighborhood size. Figures 10 (e) and (f) are produced by the GLVQ algorithm with sorted neighborhoods of orders 3 and 5, respectively.

FLVQ	**norm**	**c**	m_0	Δm	**T**	ε	**iterations**
Fig. 10(b)	Euclidean	8	1.05	0.2	80	0.5	25
Fig. 10(c)	Euclidean	8	1.05	0.2	80	0.5	24
Fig. 10(d)	Euclidean	8	1.05	0.2	80	0.5	29
Fig.11(a)	Euclidean	2	1.05	0.2	80	0.5	17
GLVQ	**norm**	**c**	α_0	$\Delta\alpha$	**T**	ε	**iterations**
Fig.10(e,f)	Euclidean	8	0.6	0.06	100	0.5	100
Fig. 11(b)	Euclidean	2	0.6	0.06	100	0.5	100

Comparing figures 10(c) and (e) we find that FLVQ and GLVQ are comparable for the house, but GLVQ extracts more compact regions for the tree areas. Another interesting thing to note is that for GLVQ with a window of size 5x5, the roof of the house is very nicely segmented with sharp inter-region boundaries; this is not true for all other cases using either algorithm.

Fig. 10 (a) Input image of a house

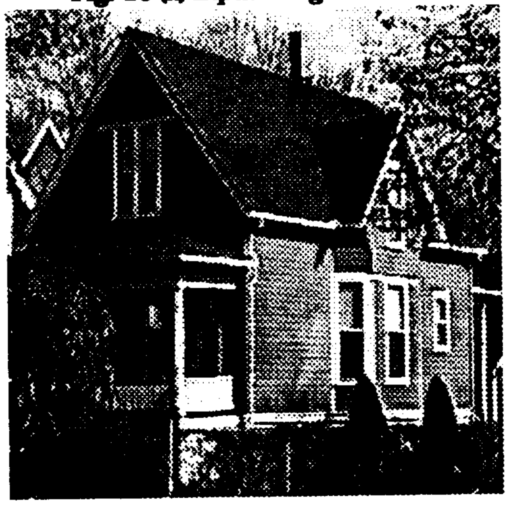


Fig. 10(b) FLVQ with N^3 (unsorted)

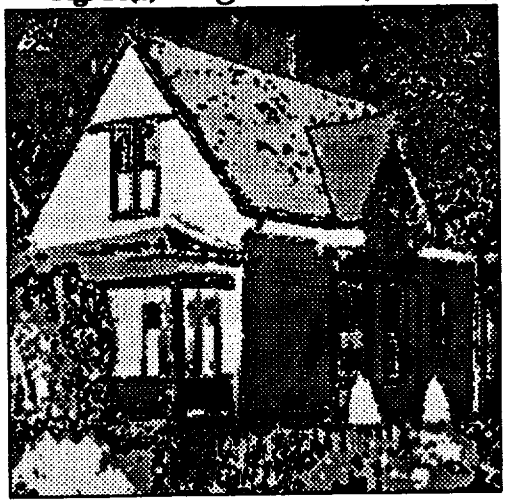


Fig. 10(c) FLVQ with N^3 (sorted)

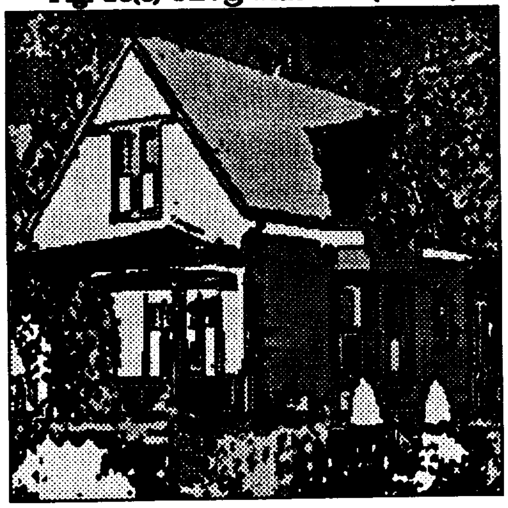


Fig. 10(d) FLVQ with N^5 (sorted)

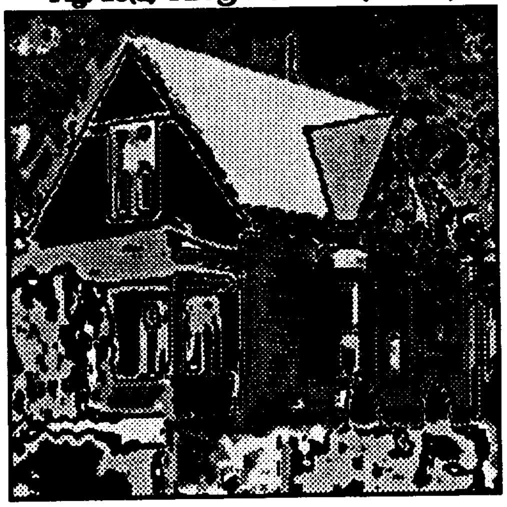


Fig. 10 (e) GLVQ with N^3 (sorted)

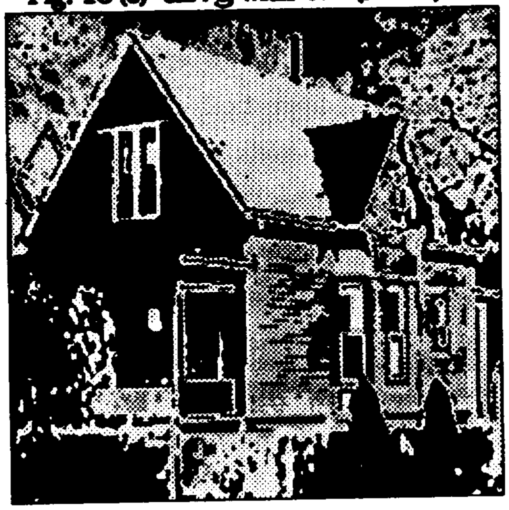
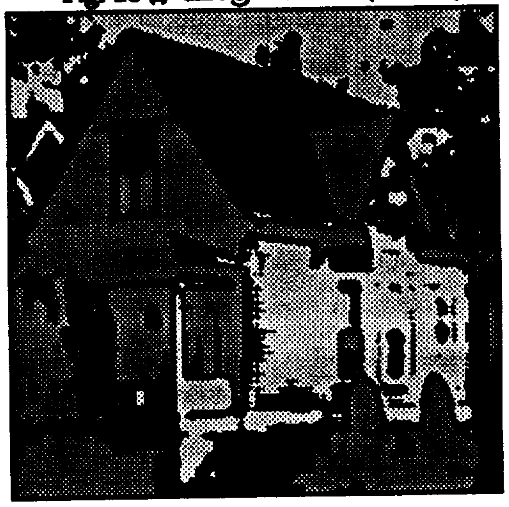


Fig. 10 (f) GLVQ with N^5 (sorted)



We used the same image (Figure 10 (a)) to test the edge-based approach. The results produced by FLVQ and GLVQ are shown in Figures 11(a) and (b), respectively. Comparing these two figures, one can see that both algorithms have extracted the compact regions nicely. A careful analysis of the images shows that FLVQ detects more edges than GLVQ. As a result of this FLVQ produces some noisy edges and GLVQ fails to extract some important edges. To summarize, both algorithms produce reasonably good results, but GLVQ has a tendency to produce larger compact (homogeneous) areas than that by the FLVQ. It appears that GLVQ is less sensitive to noise which might cause a failure to extract finer details.

Fig. 11(a) FLVQ (edge/nonedge)

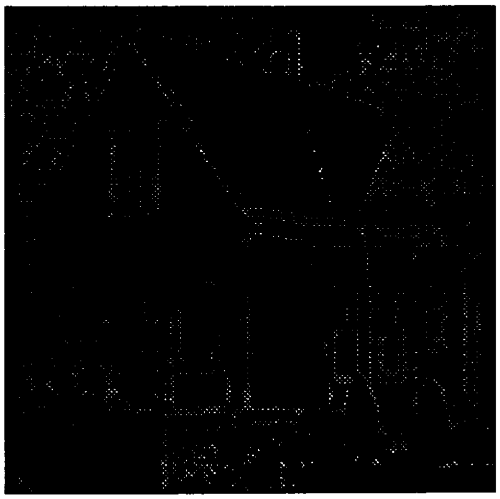
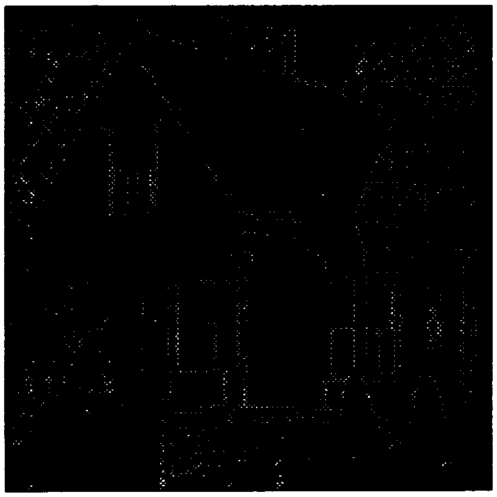


Fig. 11(b) GLVQ (edge/nonedge)



6. CONCLUSIONS

We have considered the role of and interaction between fuzzy and neural-like models for clustering, and have illustrated two generalizations of LVQ with an application in image segmentation. Unlike methods that utilize Kohonen's SOFM idea, both algorithms avoid the necessity of defining an update neighborhood scheme. Both methods are designed to optimize performance goals related to clustering, and both have update rules that allocate and distribute learning rates to (possibly) all c nodes at each pass through the data. Ascending and descending FLVQ updates all nodes at each pass, and learning rates are related to the fuzzy c-means clustering algorithm. This yields automatic control of the learning rate distribution and the update neighborhood is effectively all c nodes at each pass through the data. FLVQ can be considered a (stepwise) implementation of FCM. GLVQ needs only a specification of the learning rate sequence and an initialization of the c protoytpes. 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.

7. REFERENCES

1. Kohonen, T. *Self-Organization and Associative Memory*, 3rd Edition, Springer-Verlag, Berlin, 1989.

2. Bezdek, J. *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum,NY, 1981.

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

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

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

6. Dubes, R. and Jain, A. *Algorithms that Cluster Data*, Prentice Hall,

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

8. Ruspini, E.H. A New Approach to Clustering, *Inform. Control*, 15, 22-32, 1969.

9. Bezdek, J.C. and Pal, S.K. Fuzzy Models for Pattern Recognition, IEEE Press, Piscataway, 1992.

10. Krishnapuram, R. and Keller, J. A Possibilistic Approach to Clustering, in press, *IEEE Trans. Fuzzy Systems*, 1993.

11. Huntsberger, T. and Ajjimarangsee, P. Parallel Self-Organizing Feature Maps for Unsupervised Pattern Recognition, *Int'l. Jo. General Systems*, vol. 16, pp. 357-372, 1989.

12. Kohonen, T, Self-organizing maps: optimization approach, Artificial Neural networks, *Elsevier Sc. Pub.*, (Eds. T. Kohonen, K. Makisara, O. Simula and J. Kangas), 1991, 981-990.

13. Robbins, H. and Monro, S., A stochastic approximation method, *Ann. Math.Stat.*, 22, 400-407,1951.

14. Albert, A. E. and Gardner, L. A., Jr., Stochastic approximation and nonlinear regression, *MIT Press*, Cambridge, MA, 1967.

15. MacQueen J., Classification and analysis of multivariate observations, Some methods for classification and analysis of multivariate observations, *Proc. 5th Berkeley Symp.on Math. Stat. and Prob.*, 281-297, 1967.

16. Forgy, E. , Cluster analysis of multivariate dada : efficiency vs interpretability of classifications, WNAR meetings, Univ. of Cali - Riverside, June 22-23, 1965 (Biometrics, 21(3)).

17. Bezdek, J. C., Tsao, E. C. and Pal, N. R., Fuzzy Kohonen Clustering Networks, *Proc. IEEE Inter. Conf. of Fuzzy Syst.* , 1035-1041, March 1992, San Diego, USA .

18. Tsypkin, Y. Z., Foundations of the theory of learning systems, Trans. Z. J. Nikolic, *Academic Press*, NY, 1973.

19. Polyak, B.T., Introduction to Optimization, *Optimization Software Inc.*, New York, 1987.

20. Grossberg, S., *Studies of mind and brain*, Reidel, Boston, 1982.

21. Tsao, E.C.K., Bezdek, J.C. and Pal, N.R. Image Segmentation using Fuzzy Kohonen Clustering Networks, in press, *Proc. NAFIPS*, 1992.

22. Anderson, E., The IRISes of the Gaspe Peninsula, *Bulletin of the American IRIS Society*, 59, 2-5, 1939.

23. Pal, N.R. , Bezdek, J. C. and Tsao, E.C.K., Generalized Clustering networks and Kohonen's Self-Organizing Scheme, in press, *IEEE Trans. NN*, 1992.