Repairs to GLVQ: A New Family of Competitive Learning Schemes

Nicolaos B. Karayiannis, *Member, IEEE*, James C. Bezdek, *Fellow, IEEE*, Nikhil R. Pal, *Member, IEEE*, Richard J. Hathaway, *Member, IEEE*, and Pin-I Pai

***Abstract*—First, we identify an algorithmic defect of the generalized learning vector quantization (GLVQ) scheme that causes it to behave erratically for a certain scaling of the input data. We show that GLVQ can behave incorrectly because its learning rates are reciprocally dependent on the sum of squares of distances from an input vector to the node weight vectors. Finally, we propose a new family of models—the GLVQ-F family—that remedies the problem. We derive competitive learning algorithms for each member of the GLVQ-F model and prove that they are invariant to all scalings of the data. We show that GLVQ-F offers a wide range of learning models since it reduces to LVQ as its weighting exponent (a parameter of the algorithm) approaches one from above. As this parameter increases, GLVQ-F then transitions to a model in which either all nodes may be excited according to their (inverse) distances from an input or in which the winner is excited while losers are penalized. And as this parameter increases without limit, GLVQ-F updates all nodes equally. We illustrate the failure of GLVQ and success of GLVQ-F with the IRIS data.**

I. Introduction

PROTOTYPE generating clustering algorithms attempt to organize unlabeled feature vectors into natural groups and represent them compactly with one or more prototypes (codevectors, paradigms, templates, vector quantizers, etc.) for each cluster. Treatments of many classical approaches to this problem are given in 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 on learning vector quantization (LVQ) for unlabeled data has become particularly timely in recent years [7].

Let $X = \{x_1, x_2, \cdots, x_n\} \subset \Re^p$ denote the unlabeled data, and c denote the number of nodes in the competitive layer. The LVQ model is shown in Fig. 1. The input layer is connected directly to the competition or output layer. The ith node in the output layer is associated with a weight (or prototype) vector v_i. The p components $\{v_{ji}\}$ of v_i are often regarded as weights or connection strengths of the edges that connect the p inputs to node i.

Manuscript received February 3, 1995; revised September 11, 1995. This work was supported in part by the Faculty Research Committee of Georgia Southern University and the Georgia Southern Foundation.

N. B. Karayiannis and P.-I Pai are with the Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77204 USA.

J. C. Bezdek is with the Department of Computer Science, University of West Florida, Pensacola, FL 32514 USA.

N. R. Pal is with the Machine Intelligence Unit, Indian Statistical Institute, Calcutta 700035, India.

R. J. Hathaway is with the Mathematics and Computer Science Department, Georgia Southern University, Statesboro, GA 30460-8093 USA.

Publisher Item Identifier S 1045-9227(96)06601-5.

The prototypes $V = (v_1, v_2, \cdots, v_c), v_i \in \Re^p$ for $1 \le i \le c$, are the (unknown) vector quantizers we seek. In this context *learning* refers to finding values for the $\{v_{ji}\}$. When an input vector x is submitted to this network, distances are computed between x and each v_i. The output nodes "compete," a (minimum distance) winner node, say v_i, is found, and it is then updated using one of several update rules. Fig. 2 gives a brief specification of the LVQ algorithm (some writers call it simple competitive learning) that is used in Section IV.

The update scheme in (2), shown in Fig. 2, has the simple geometric interpretation shown in Fig. 3. The winning prototype $v_{i,t-1}$ is simply rotated toward the current data point x_k by moving along the vector $(x_k - v_{i,t-1})$ which connects it to x_k.

The amount by which $v_{i,t-1}$ is shifted to arrive at $v_{i,t}$ depends on the value of the *learning rate* parameter α_{t-1}, which varies from zero to one. There is no update if $\alpha_{t-1} = 0$, and when $\alpha_{t-1} = 1, v_{i,t}$ becomes x_k ($v_{i,t}$ is just a convex combination of x_k and $v_{i,t-1}$). This process continues until termination via LVQ4 (or by iterate limit T), at which time the terminal prototypes can be used, for example, to find the nearest prototype hard c-partition of X via LVQ5.

LVQ ordinarily uses the Euclidean distance in (1), shown in Fig. 2. This choice corresponds roughly to the update rule shown in (2), since $\nabla_v(\|x - v\|^2) = -2(x - v)$. The origin of this learning rule comes about by assuming that each $x \subset \Re^p$ is distributed according to an unknown time-invariant probability density function $f(x)$. LVQ's objective is to find a set of v_i's such that the expected value of the square of the discretization error

$$E(\|x - v_i\|^2) = \int_{\Re^p} \cdots \int \|x - v_i\|^2 f(x)\, dx \qquad (3)$$

is minimized. In this expression v_i is the winning prototype for each x, and will of course vary as x ranges over $\Re^p$. A sample function of the optimization problem defined by minimization of E over $\Re^p$ is $e = \|x - v_i\|^2$. An optimal set of v_i's can be approximated by applying local gradient descent to $e = \|x - v_i\|^2$ for every data point x drawn from f. Kohonen has shown [8] that under certain assumptions on the learning rates, steepest descent optimization of E is also possible, and both strategies (stochastic approximation and steepest descent) lead to update rule (2).

LVQ attempts to minimize an objective function that places all of its emphasis on the winning prototype for each data point. This is reflected in (2), which alters only the winner

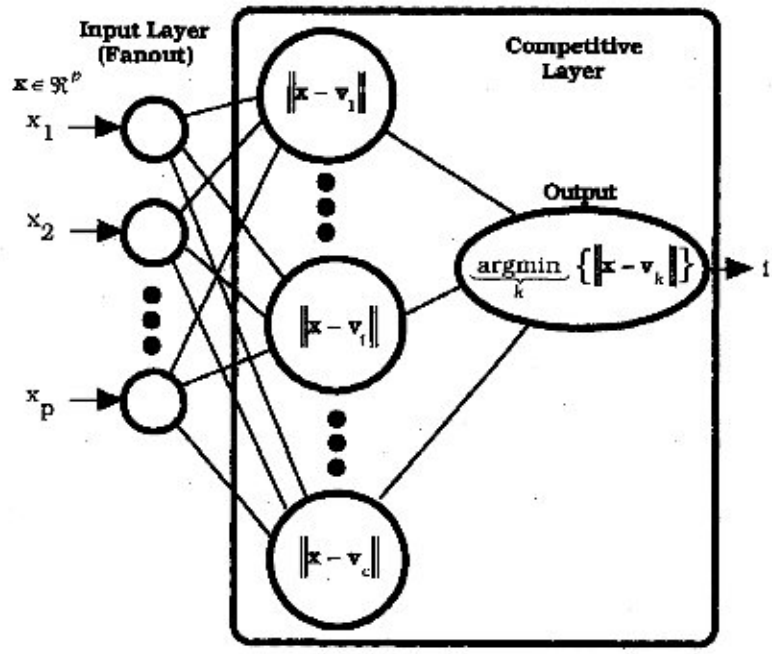

Fig. 1. The LVQ competitive learning network.

LVQ1. Given unlabeled data set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \Re^p$. Pick c, T = maximum number of updating steps, and $\varepsilon > 0$.

LVQ2. Initialize $\mathbf{V}_0 = (\mathbf{v}_{1,0}, \ldots, \mathbf{v}_{c,0}) \in \Re^{cp}$; Initialize learning rate $\alpha_0 \in (1,0)$

For t = 1,2,..., T:

For k = 1,2, ..., n:

a. Find $i = \underbrace{\arg\min}_{r} \{\|\mathbf{x}_k - \mathbf{v}_r\|\}$. (1)

b. Update the winner by

$$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \alpha_t(\mathbf{x}_k - \mathbf{v}_{i,t-1}) . \quad (2)$$

Next k

LVQ3. Compute

$$E_t = E_t = \|\mathbf{V}_t - \mathbf{V}_{t-1}\| = \sum_{k=1}^{n} \sum_{r=1}^{c} |v_{rk,t} - v_{rk,t-1}|,$$

LVQ4. If $E_t \le \varepsilon$ stop;

Else $\alpha_t \leftarrow \alpha_t(1 - t / T)$;

Next t.

Labeling Phase (optional)

LVQ5. Find $i = \underbrace{\arg\min}_{s} \{\|\mathbf{x}_k - \mathbf{v}_{s,t}\|\}$, and mark $\mathbf{x}_k$ in X with label i .

Fig. 2. The (Unlabeled Data) LVQ Algorithm. LVQ1–4 are the learning phase.

for each x submitted. This ignores global information about the geometric structure of the data that is represented in

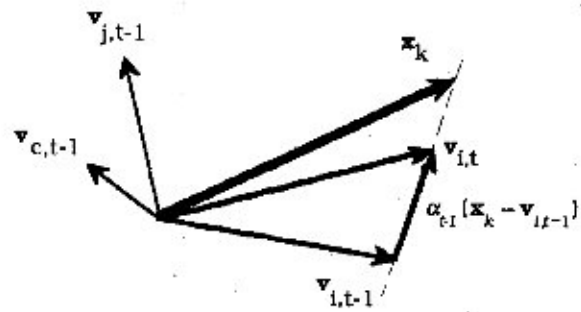

Fig. 3. Updating the winning LVQ prototype.

the remaining $(c - 1)$ distances from x to the nonwinner prototypes. In this sense LVQ updating is somewhat like using the sup norm, $\|x - v\|_\infty = \max_{j=1}^{p} \{|x_j - v_j|\}$, to measure distance in $\Re^p$. The extreme value in the absolute difference between pairs of coordinates dominates all others, and the distance produced simply ignores them. In the same way, LVQ updating is a very harsh local strategy that ignores global relationships between the input datum and nonwinner prototypes.

Using an inner product norm on $\Re^p$ ameliorates the harshness of the sup norm by counting contributions from each pair of coordinate differences in the overall distance calculation. In the same way, we think that nonwinner nodes in an LVQ network should be allowed to influence the update of the winner, and perhaps, be updated themselves. This presents two questions. First, which nodes should be accounted for (*what is the update neighborhood*)? Second, how much influence should each nonwinner node exert (*what is the learning rate distribution*)? Modifications of LVQ are usually motivated by a desire to answer one or both of these questions. Variations of LVQ that update all c quantizers simultaneously during each

updating epoch include: the soft competition scheme (SCS) of Yair *et al.* [9]; the fuzzy learning vector quantization (FLVQ) method of Tsao *et al.* [10]; and the generalized learning vector quantization (GLVQ) algorithm of Pal *et al.* [11].

The objectives of this paper are to demonstrate, analyze and remedy a defect of GLVQ that occurs for a particular scaling of the input data. Independently and at the roughly the same time as this study, Gonzalez *et al.* performed a somewhat different analysis and presented further examples of the erratic behavior of GLVQ [12]. Their results are different than the ones given in this article in several respects. First, the data used in their examples that showed sensitivity to data scaling were randomly generated subsets of $\Re^2$. They also demonstrated computationally that GLVQ could exhibit descent, ascent and even limit cycle behavior with respect to its loss function, and that the algorithm was sensitive to the number (c) of prototypes being sought. On the other hand, no replacement for or correction to GLVQ as given here was offered by Gonzalez *et al.* Nonetheless, their paper is well worth reading as a companion paper to this one, and readers are encouraged to study both.

Section II reviews GLVQ, demonstrates the problem with a numerical example and identifies the cause of the problem. Section III defines, illustrates, and analyzes a new infinite family of competitive learning models that overcomes the defect. We prove that algorithms used to optimize members of this new family are invariant to positive scalings of the data; and give a rough analysis of the limiting behavior of these algorithms as one of their parameters approaches its limits from above and below. Section IV contains our conclusions and some ideas for further research.

II. Generalized LVQ

Let $\boldsymbol{x} \in \Re^p$ be a stochastic input vector distributed according to a time invariant probability distribution $f(\boldsymbol{x})$, and let i be the best matching node as in (2). Let $L_{\boldsymbol{x}}$ be a loss function which measures the locally weighted mismatch (error) of $\boldsymbol{x}$ with respect to the winner

$$L_{\boldsymbol{x}} = L(\boldsymbol{x}; \boldsymbol{v}_1, \cdots, \boldsymbol{v}_c) = \sum_{r=1}^{c} g_r \|\boldsymbol{x} - \boldsymbol{v}_r\|^2, \quad \text{where} \tag{4a}$$

$$g_i = \left\{ \begin{array}{ll} 1 & \text{if } i = \underbrace{\arg\min}_{r} \{\|\boldsymbol{x} - \boldsymbol{v}_r\|\} \\ \dfrac{1}{\sum_{j=1}^{c} \|\boldsymbol{x} - \boldsymbol{v}_j\|^2}, & \text{otherwise} \end{array} \right\}. \tag{4b}$$

Notice especially that (4b) specifies only two values for the (c) weights $\{g_r\}$; the weight $g_i = 1$ for the winner, and a different but equal weight g_r for all $(c-1)$ nonwinners. When $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n, \cdots,\}$ is a set of samples from $f(\boldsymbol{x})$ drawn at times $t = 1, 2, \cdots, n, \cdots$, the objective of GLVQ 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 a solution for the optimization problem

$$\underbrace{\text{minimize}}_{\boldsymbol{V} \in \Re^{cp}} \left\{ 1'(\boldsymbol{V}) = \iint_{\Re^p, \cdots, \Re^p} \sum_{r=1}^{c} g_r \|\boldsymbol{x} - \boldsymbol{v}_r\|^2 f(\boldsymbol{x})\, d\boldsymbol{x} \right\}. \tag{5}$$

Pal *et al.* derived update rules for solving (5) based on minimization of $L_{\boldsymbol{x}_k}$ via the method of steepest descent [11]. These rules are

$$\boldsymbol{v}_{i,t} = \boldsymbol{v}_{i,t-1} + \alpha_{t-1} \underbrace{\left[\frac{D^2 - D + \|\boldsymbol{x}_k - \boldsymbol{v}_{i,t-1}\|^2}{D^2} \right]}_{\beta_{ik,t-1}} \cdot (\boldsymbol{x}_k - \boldsymbol{v}_{i,t-1}) \quad \text{for the winner node } i, \text{ and} \tag{6a}$$

$$\boldsymbol{v}_{j,t} = \boldsymbol{v}_{j,t-1} - \alpha_{t-1} \underbrace{\left[\frac{\|\boldsymbol{x}_k - \boldsymbol{v}_{i,t-1}\|^2}{D^2} \right]}_{\gamma_{ik,t-1}} (\boldsymbol{x}_k - \boldsymbol{v}_{j,t-1}) \cdot$$

$$\text{for } j = 1, 2, \cdots c, j \neq i \tag{6b}$$

where $\boldsymbol{x}_k$ is the current input vector, and

$$D = \sum_{r=1}^{c} \|\boldsymbol{x}_k - \boldsymbol{v}_{r,t-1}\|^2.$$

The GLVQ algorithm is defined by replacing update (2) in LVQ by (6). To avoid possible oscillations of the solution, the recommendation in [11] for learning rate α_t was that it should satisfy two conditions: as $t \to \infty$; $\alpha_t \to 0$ and $\Sigma \alpha_t = \infty$. For example, the learning rate $\alpha_t \leftarrow \alpha_0(1 - t/T)$ at LVQ4 is fine—note that it depends only on t and T, the iteration counter and limit, respectively. The overall learning rates in (6) thus become products of two factors; for the winner, $\alpha_{t-1} \cdot \beta_{ik,t-1}$; and for the nonwinners, $\alpha_{t-1} \cdot \gamma_{ik,t-1}$, where β and γ are as shown in (6a) and (6b). It is important to observe that β and γ both contain D^2 in their denominators.

From (6b) it follows that when the match to the winner is perfect ($\|\boldsymbol{x} - \boldsymbol{v}_{i,t-1}\|^2 = 0$), nonwinner nodes are not updated and GLVQ reduces to LVQ. In [11] it was also asserted that as the match between $\boldsymbol{x}$ and the winner node $\boldsymbol{v}_i$ decreased, the updating impact on nonwinner nodes increased. In view of (4b), this impact was always thought to produce nonwinner updates that were less than the update applied to the winner. However, such is not always the case. Karayiannis and Pai [13], [14] discovered that using GLVQ on the IRIS data after scaling it with ($\boldsymbol{x}_k \leftarrow \boldsymbol{x}_k/10$) produced far different results than those reported in [11].

To appreciate this, we again use Anderson's IRIS data as an experimental data set [15]. IRIS contains 50 (physically labeled) vectors in $\Re^4$ for each of $c = 3$ classes of IRIS subspecies, and it is important for this example to know that the minimum and maximum feature values in IRIS are 0.1 and 7.7. IRIS has been used in many papers to illustrate various clustering (unsupervised) and classifier (supervised) designs. Typical (resubstitution) error rates for supervised designs are zero–five mistakes; and for unsupervised designs, around 16 "mistakes" on classifiers that are subsequently designed with clustering outputs (usually prototypes).

TABLE I
GLVQ ON IRIS AND IRIS/10 FOR $T = 500$ AND $\alpha_0 = 0.6$

	Initial prototypes				Terminal V on IRIS/10				Physical Class Means/10			
v_1	0.430	0.200	0.100	0.010	0.588	0.320	0.348	0.113	0.500	0.342	0.146	0.024
v_2	0.610	0.320	0.395	0.130	0.586	0.321	0.349	0.113	0.593	0.277	0.426	0.132
v_3	0.790	0.440	0.690	0.250	0.586	0.321	0.347	0.112	0.658	0.297	0.555	0.202
					Terminal V on IRIS				Physical Class Means			
v_1	4.300	2.000	1.000	0.100	5.007	3.423	1.472	0.250	5.006	3.428	1.462	0.246
v_2	6.100	3.200	3.950	1.300	5.881	2.742	4.387	1.433	5.936	2.770	4.260	1.326
v_3	7.900	4.400	6.900	2.500	6.847	3.078	5.708	2.058	6.588	2.974	5.552	2.026

Nearest prototype classifier design based on (any) unsupervised prototype generating algorithm is done as follows. First, the prototypes are found by exercising the training phase on all of the *unlabeled* data until termination. At this point, the terminal prototypes $\boldsymbol{V} = \{\boldsymbol{v}_k: 1 \le k \le c\}$ have *numerical* (not physical) labels. Once the prototypes are relabeled (if needed) using a relabeling algorithm such as the one discussed in [11], they are used to define a crisp nearest prototype (1-NP) classifier. This classifier is simple to implement, needing, besides $\boldsymbol{V}$, only a definition of how to interpret (compute) the concept of nearest. The usual choice is to measure distances using a norm on $\Re^p$; and among the norms, the most common choice is an inner product norm. This is the classifier we will use to compare sets of prototypes.

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

$$\text{Decide } \boldsymbol{z} \in i \text{ (i.e., label } \boldsymbol{z} \text{ as class } i) \Leftrightarrow \|\boldsymbol{z} - \boldsymbol{v}_i\|_A < \|\boldsymbol{z} - \boldsymbol{v}_j\|_A; 1 \le j \le c, j \ne i. \quad (7)$$

In (7) A is any *positive definite* $p \times p$ weight matrix—it renders the norm in (7) an inner product norm, $\|\boldsymbol{z} - \boldsymbol{v}_i\|_A = \sqrt{(\boldsymbol{z} - \boldsymbol{v}_i)^T A (\boldsymbol{z} - \boldsymbol{v}_i)}$. Equation (7) defines a crisp classifier that has piecewise-linear decision boundaries, even though its parameters $(\boldsymbol{V})$ may come from a fuzzy or probabilistic algorithm. Our calculations use $A = I$, the $p \times p$ identity matrix, which gives the Euclidean norm. Ties in (7) are resolved arbitrarily. Note that (7) is the same operation as LVQ5.

How do we use 1-NP machines to compare unsupervised learning algorithms? The method employed here is to derive prototypes $\boldsymbol{V}$ from labeled data without using the labels (that is, we pretend there are no labels) during the training phase. Then, the data are submitted a 1-NP design based on $\boldsymbol{V}$ and crisp labels are assigned to each datum via (7). Finally, we count the number of errors by comparing the computed labels to the target (given) labels. Error counts are conveniently tabulated using the confusion matrix C that can be constructed during this process. The observed resubstitution error rate (in percent) is

$$E(X, \boldsymbol{V}) = 100 * \left(1 - \left(\frac{\#\text{ right}}{\#\text{ tried}}\right)\right) = 100 * (1 - (tr(C))) \quad (8)$$

where C is the $(c \times c)$ *confusion matrix*

$$C = [c_{ij}] = [\#\text{ labeled class } j | \text{but were really class } i]. \quad (9)$$

In the examples below where X = IRIS, errors made by (7) using any $\boldsymbol{V}$ are aggregated by resubmitting IRIS to the nearest prototype rule and comparing the label assigned by (7) to the *physical* label of each vector in the data.

Good initialization of competitive learning schemes that find vector quantizers from unlabeled data is very important [11]. A typical choice for the initial prototypes is to make random draws from the uniform distribution on $[0, 1]$ for each coordinate of each prototype. This might work well when the data have values in this range, and random draws can always be scaled to the general range of the data. Another popular method is to simply draw (c) distinct vectors from data set X, and use them as the initial prototypes.

Random initialization or c draws from X do not generally allocate prototypes uniformly over the input space of the data. An initialization method that guarantees this follows. For data set $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\} \subset \Re^p$, let data point k and initial prototype i be $\boldsymbol{x}_k = \{x_{1k}, x_{2k}, \cdots, x_{pk}\}^T$ and $\boldsymbol{v}_i = \{v_{1i}, v_{2i}, \cdots, v_{pi}\}^T$, respectively. Compute the feature ranges

Minimum of feature j:

$$m_j = \underbrace{\min}_{k}\{x_{jk}\}: j = 1, 2, \cdots, p; \quad \text{and} \quad (10)$$

Maximum of feature j:

$$M_j = \underbrace{\max}_{k}\{x_{jk}\}: j = 1, 2, \cdots, p. \quad (11)$$

With these, compute the jth component of the ith initial prototype as

$$v_{ji} = m_j + (i - 1)\left(\frac{M_j - m_j}{c - 1}\right) \quad i = 1, 2, \cdots, c; \quad j = 1, 2, \cdots p. \quad (12)$$

Formula (12) disperses initial prototype values uniformly along each feature range $[m_j, M_j]$. For example, $\boldsymbol{v}_1 = \boldsymbol{m} = (m_1, m_2, \cdots, m_p)^T$, $\boldsymbol{v}_c = \boldsymbol{M} = (M_1, M_2, \cdots, M_p)^T$, and so on.

Table I shows the results of $T = 500$ iterations of GLVQ using the initialization in (12), with an initial learning rate $\alpha_0 = 0.6$ on IRIS and IRIS/10, where the latter data set is produced from IRIS by the scaling $(\boldsymbol{x}_k \leftarrow \boldsymbol{x}_k/10)$. You can see that the prototypes produced by GLVQ on IRIS are very good estimates of the physically labeled subclass mean vectors, while the estimates produced by the same algorithm on IRIS/10 are: 1) nearly equal to each other; and 2) not close to any subclass centroid. This will have an obvious (disastrous!)

TABLE II
ERROR RATES AND CONFUSION MATRICES USING GLVQ ON IRIS AND IRIS/10

	IRIS/10	IRIS
Confusion Matrix	$\begin{pmatrix} 50 & 0 & 0 \\ 1 & 1 & 48 \\ 0 & 1 & 49 \end{pmatrix}$	$\begin{pmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 14 & 36 \end{pmatrix}$
Error rate	$\hat{\epsilon}$(IRIS/10, $\mathbf{V}$) = 33.3%	$\hat{\epsilon}$(IRIS, $\mathbf{V}$) = 11.3%

effect on any nearest prototype classifier designed with the terminal quantizers for IRIS/10 shown in Table I.

Table II shows the confusion matrices and error rates produced by the nearest prototype rule (7) after the centroids obtained by GLVQ in these two cases were relabeled as needed by the relabeling algorithm in [11]. Simply dividing each datum by 10 triples the error rate!

Table II shows that GLVQ is very sensitive to simple scaling of the data. To see why, let $\Delta \boldsymbol{v}_{r,t} = \boldsymbol{v}_{r,t} - \boldsymbol{v}_{r,t-1}$ for $r = 1$ to c, and recall that $\gamma_{ik,t-1} = \|\boldsymbol{x}_k - \boldsymbol{v}_{i,t-1}\|^2/D^2$ in (6b). With these conventions equations (6a) and (6b) become

$$\Delta \boldsymbol{v}_{i,t} = \alpha_{t-1}\left[\gamma_{ik,t-1} - 1 - \frac{1}{D}\right](\boldsymbol{x}_k - \boldsymbol{v}_{i,t-1}) \quad \text{for the winner node } i, \text{ and} \tag{6a'}$$

$$\Delta \boldsymbol{v}_{r,t} = \alpha_{t-1} \cdot \gamma_{ik,t-1}(\boldsymbol{x}_k - \boldsymbol{v}_{r,t-1}) \quad \text{for } r = 1, 2, \cdots, c,\ r \neq i. \tag{6b'}$$

It is easy to see that if

$$D = \sum_{r=1}^{c} \|\boldsymbol{x}_k - \boldsymbol{v}_{r,t-1}\|^2 < 1$$

$$\left[\gamma_{ik,t-1} + 1 - \frac{1}{D}\right] < \gamma_{ik,t-1}.$$

Since α_{t-1} is the same for winners and nonwinners, the (percent) change in the winner node vector may be *less than* changes that are made to the other $(c-1)$ node vectors in this case. Since the minimum and maximum values in IRIS/10 are 0.01 and 0.77, this has apparently happened during the application of GLVQ to IRIS/10. More generally, there may be a scaling of *any* data set which produces feature vector magnitudes that make this happen. In this case, GLVQ may terminate at a very undesirable or misleading set of vector quantizers of cluster structure in X because the nonwinner nodes will be pulled to the data more strongly than the winner node is. This results in all node vectors migrating to the same point in $\Re^p$ as they did for IRIS/10 in Table I.

III. A FUZZY MODIFICATION OF GLVQ

The failure of GLVQ on IRIS/10 leads back to the objective function shown in (4). Substitution of

$$D = \sum_{r=1}^{c} \|\boldsymbol{x}_k - \boldsymbol{v}_{r,t-1}\|^2$$

into (4b) shows the problem just exposed

$$g_i = \begin{cases} 1 & \text{if } i = \underbrace{\arg\min}_{r}\{\|\boldsymbol{x}_k - \boldsymbol{v}_r\|\} \\ \dfrac{1}{D}, & \text{otherwise} \end{cases}, \qquad 1 \le i \le c. \tag{4b'}$$

In this form, it is easy to see that when $D < 1$ GLVQ behaves *exactly opposite* to what was desired.[1] Thus, we are led to propose a new model and algorithm that is related to GLVQ in that it is defined by the same optimization problem —viz. (5). The modification arises (as do others proposed by various authors) by changing the definition of the weights $\{g_r\}$ at (4b).

Here are the properties we want the $\{g_r\}$ to have: 1) the magnitude of g_r should be *inversely proportional* to $\|\boldsymbol{x}_k - \boldsymbol{v}_r\|$; 2) each g_r should be in [0, 1]; and 3) the sum of the $\{g_r\}$ should be one. There are many choices for the $\{g_r\}$ that satisfy 1)–3). For example, Yair *et al.* [9] proposed in their SCS scheme learning rates that use posterior probabilities (which satisfy these constraints) as one of their learning rate factors.

Many authors, beginning with Huntsberger and Ajjimarangsee [16], have felt that a natural choice for g_r which satisfies 1)–3) is (some function of) the membership value obtained from the fuzzy c-means (FCM) formula [2] for any $m > 1$. The basic formula for the memberships of an input $\boldsymbol{x}$ over the c nodes is (assuming $\|\boldsymbol{x} - \boldsymbol{v}_j\| > 0\ \forall j$ and $m > 1$)

$$u_r = \left(\sum_{j=1}^{c}\left(\frac{\|\boldsymbol{x} - \boldsymbol{v}_r\|^{2/(m-1)}}{\|\boldsymbol{x} - \boldsymbol{v}_j\|^{2/(m-1)}}\right)\right)^{-1}, \qquad r = 1, 2, \cdots, c. \tag{13}$$

Various investigators have used *functions of* this formula in different ways: e.g., the FLVQ (or FKCN) [10], GBFCM [17] and UFCL [18] algorithms are all based on (13) in one way or another. The use of (13) is intuitively appealing because FCM is a successful and robust batch clustering algorithm, and we may hope that part of its utility can be transferred to competitive learning schemes via the formula itself. Using (13) as we have here is difficult to justify mathematically (except as it does satisfy 1)–3) above), so this modification is best viewed as a heuristically based idea. Substituting the weights in (13) in (4a) yields

$$L_{\boldsymbol{x}} = L(\boldsymbol{x}; \boldsymbol{v}_1, \cdots, \boldsymbol{v}_c) = \sum_{r=1}^{c} u_r^m \|\boldsymbol{x} - \boldsymbol{v}_r\|^2 = \sum_{r=1}^{c}\left(\sum_{j=1}^{c}\left(\frac{\|\boldsymbol{x} - \boldsymbol{v}_r\|^{2/(m-1)}}{\|\boldsymbol{x} - \boldsymbol{v}_j\|^{2/(m-1)}}\right)\right)^{-m} \|\boldsymbol{x} - \boldsymbol{v}_r\|^2. \tag{14}$$

Assuming that $\|\boldsymbol{x} - \boldsymbol{v}_j\| > 0\ \forall j$ and $m > 1$, the gradient of $L_{\boldsymbol{x}}$ with respect to $\boldsymbol{v}_k$ is (see Appendix A)

$$\nabla_{\boldsymbol{v}_k} L = \nabla_{\boldsymbol{v}_k}\left(\sum_{r=1}^{c} u_r^m \|\boldsymbol{x} - \boldsymbol{v}_r\|^2\right) = \left(\frac{-2}{m-1}\right)(\boldsymbol{x} - \boldsymbol{v}_k) \cdot \left[(m-2)u_k^m + \left(\sum_{r}\left(\frac{\|\boldsymbol{x} - \boldsymbol{v}_k\|^{2/m-1)}}{\|\boldsymbol{x} - \boldsymbol{v}_r\|^{2/m-1)}}\right)^{2-m}\right] u_k^2. \tag{15a}$$

And for the important special case $m = 2$, (15a) reduces to

$$\nabla_{\boldsymbol{v}_k} L = -2cu_k^2(\boldsymbol{x} - \boldsymbol{v}_k). \tag{15b}$$

[1] In fact, there are values of D for which $[\gamma_{ik,t} + 1 - 1/D] < 0$. And for $D >> 1$, GLVQ reverts to LVQ.

TABLE III
GLVQ-F ($m = 2$) ON IRIS AND IRIS/10 FOR $T = 500$ AND $\alpha_0 = 0.6$

	Initial prototypes/10				Terminal V on IRIS/10				Physical Class Means/10			
v_1	0.430	0.200	0.100	0.010	0.500	0.341	0.148	0.025	0.500	0.342	0.146	0.024
v_2	0.610	0.320	0.395	0.130	0.588	0.276	0.435	0.139	0.593	0.277	0.426	0.132
v_3	0.790	0.440	0.690	0.250	0.676	0.305	0.563	0.205	0.658	0.297	0.555	0.202
	Initial Prototypes				Terminal V on IRIS				Physical Class Means			
v_1	4.300	2.000	1.000	0.100	5.002	3.412	1.483	0.254	5.006	3.428	1.462	0.246
v_2	6.100	3.200	3.950	1.300	5.884	2.762	4.358	1.393	5.936	2.770	4.260	1.326
v_3	7.900	4.400	6.900	2.500	6.769	3.054	5.637	2.057	6.588	2.974	5.552	2.026

TABLE IV
ERROR RATES AND CONFUSION MATRICES USING GLVQ-F ($m = 2$) ON IRIS AND IRIS/10

		IRIS/10	IRIS
GLVQ - F	Confusion Matrix	$\begin{pmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 13 & 37 \end{pmatrix}$	$\begin{pmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 13 & 37 \end{pmatrix}$
	Error rate	$\mathcal{E}$(IRIS/10, **V**) = 10.7%	$\mathcal{E}$(IRIS, **V**) = 10.7%

Using (15a), the learning rule obtained by applying gradient descent to $L_{\boldsymbol{x}}$ at iterate t for input vector $\boldsymbol{x}$ becomes, under the assumptions just stated

$$\boldsymbol{v}_{j,t} = \boldsymbol{v}_{j,t-1} + \left(\frac{2\alpha_{t-1}}{m-1}\right) \cdot \left[(m-2)u_j + \left(\sum_{r=1}^{c}\left(\frac{\|\boldsymbol{x}-\boldsymbol{v}_{j,t-1}\|^{2/m-1}}{\|\boldsymbol{x}-\boldsymbol{v}_{r,t-1}\|^{2/m-1}}\right)^{2-m}\right)u_j^2\right] \cdot (\boldsymbol{x}-\boldsymbol{v}_{j,t-1}); \qquad j = 1, 2, \cdots, c. \tag{16a}$$

For $m = 2$ gradient (15b) yields the much simpler form

$$\boldsymbol{v}_{j,t} = \boldsymbol{v}_{j,t-1} + \alpha_{t-1}(2cu_j^2)(\boldsymbol{x}-\boldsymbol{v}_{j,t-1}); \qquad j = 1, 2, \cdots, c. \tag{16b}$$

Since $0 < u_{ir} < 1 = \sum_r u_{ir}$ and $u_i < u_j \Leftrightarrow \|\boldsymbol{x}-\boldsymbol{v}_i\| > \|\boldsymbol{x}-\boldsymbol{v}_j\|$, the overall learning rate $\alpha_{t-1}(2cu_r^2)$ for node r (at $m = 2$) is inversely proportional to its distance from the data point—that is, the closer the node vector, the larger the update. This update rule does not have the problem of GLVQ that was illustrated in Section III. Using (16a) instead of (2) in our specification of LVQ results in an infinite family (one for each m) of competitive learning schemes we shall call GLVQ-F. (Perhaps GLVQ1 is a more natural name, but we wish to avoid possible confusion with Kohonen's LVQ1, which is a learning algorithm that uses *labeled* data.) We mention that the factor $(2c)$ in the learning rate $\alpha_{t-1}(2cu_r^2)$ is unimportant. The calculations discussed below used the simpler and equivalent expression $\alpha_{t-1}u_r^2$ as the overall learning rate for node r at iterate t in (16b). The overall learning rates in (16a) at any value of m do not satisfy the desirable properties stipulated for the weights $\{g_r\}$: rather, these properties are built into the rates as part of one factor.

We remark that learning rule (16b) was proposed explicitly by Park and Dagher in [17], and implicitly by Chung and Lee in [18] [they used any $m > 1$ but for $m \neq 2$ they did not arrive at (15a)]. However, neither of these authors justified the use of (16b) via the well-defined optimization problem at (5); and neither used (16b) as the update rule the way we do. Instead, they mistakenly identified (16b) as part of a scheme to optimize the kth term of the fuzzy c-means objective function, i.e.,

$$\min_{(\boldsymbol{U}_i, \boldsymbol{V})}\left\{J_{m,k}(\boldsymbol{U}_k, \boldsymbol{V}; X) = \sum_{i=1}^{c}(u_{ik})^m\|\boldsymbol{x}_k - \boldsymbol{v}_i\|_A^2; m > 1\right\}.$$

A generalized form of (16b), different from (16a), used in the UFCL method proposed in [18] is very unstable to values of the FCM parameter (m) for m less or greater than $m = 2$. We shall return to this point later.

To see that GLVQ-F overcomes the problem of scaling that GLVQ suffers from, consider Tables III and IV, which contain the results of applying GLVQ-F (with $m = 2$) to IRIS and IRIS/10 using exactly the same initializations as in Table I.

The terminal prototypes found by GLVQ-F on both IRIS and IRIS/10 agree well with the physical class means. Differences in the second and third prototypes with the second and third class means indicate that the physical labels of the second and third subsets of IRIS do not correspond exactly with the (geometric) structure of the data that represent the flowers. This is a problem of good data representation, not bad terminal prototypes.

The fact that GLVQ F makes 16 classification errors on IRIS and IRIS/10 instead of 17 is not important, because changes in T and α_0 can affect its outputs. (Applying the 1-NP rule to IRIS using the physical class means for $\boldsymbol{V}$ results in 11 errors.) What is important is that these results show that GLVQ-F does not suffer from the scaling problem that plagues GLVQ.

Table III suggests that GLVQ-F is affected very little by the scale of the data. We can show (the proof is in Appendix B) that GLVQ-F is scale invariant in the sense of the following proposition, which assures us that GLVQ-F cannot suffer from the scaling problem that GLVQ does.

1) Proposition: Let $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\} \subset \Re^p$ and let $V_0 = (\boldsymbol{v}_{1,0}, \boldsymbol{v}_{2,0}, \cdots, \boldsymbol{v}_{c,0}); \boldsymbol{v}_{i,0} \in \Re^p$, be a set of initial

prototypes. Let τ be a fixed positive number and define sets of scaled data and initial prototypes by

$$Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_n\} = \tau X \{\tau \boldsymbol{x}_1, \tau \boldsymbol{x}_2, \cdots, \tau \boldsymbol{x}_n\} \quad \text{and}$$
$$\boldsymbol{W}_0 = (\boldsymbol{w}_{1,0}, \boldsymbol{w}_{2,0}, \cdots, \boldsymbol{w}_{c,0}) = \tau \boldsymbol{V}_0 = (\tau \boldsymbol{v}_{1,0}, \tau \boldsymbol{v}_{2,0}, \cdots, \tau \boldsymbol{v}_{c,0}).$$

Then applying GLVQ F to X, initialized by $\boldsymbol{V}_0$, is equivalent to applying GLVQ-F to Y, initialized by $\boldsymbol{W}_0$, in the sense that

$$\boldsymbol{w}_{j,t} = \tau \boldsymbol{v}_{j,t}; \qquad j = 1, \cdots, c \quad \text{and} \quad t = 0, 1, \cdots. \tag{17}$$

We remark that whenever α_0, $\{\alpha_t\}$ and T are the same for the X and Y problems in this proposition, and when the termination criterion for the two problems satisfies $\varepsilon_Y = \tau \varepsilon_X$, that the terminal prototypes satisfy $\boldsymbol{W} = \tau \boldsymbol{V}$.

Since GLVQ-F is well defined for any m in $(1, \infty)$, users must choose a value for this parameter before the algorithm can be implemented. Computational experiments we have done suggest that $m = 2$ may be the most reliable value. To investigate this further, we can take limits of the gradient in (15a) as m approaches its upper and lower limits. Here are the results. First (see Appendix C), we have that

$$\underbrace{\lim}_{m \to \infty} \{\boldsymbol{v}_{j,t}\} = \boldsymbol{v}_{j,t-1} + \left(\frac{2\alpha_{t-1}}{c}\right)(\boldsymbol{x} - \boldsymbol{v}_{j,t-1}) \qquad j = 1, 2, \cdots, c. \tag{18}$$

This establishes that all c prototypes are adjusted equally in the sense that the *learning rate* is the same for all c weight vectors in (16) as parameter m grows without bound. At the other extreme, when m approaches one from above, we find that (cf. Appendix D)

$$\underbrace{\lim}_{m \to 1^+} \{\boldsymbol{v}_{j,t}\} = \left\{ \begin{array}{ll} \boldsymbol{v}_{i,t-1} + (2\alpha_{t-1})(\boldsymbol{x} - \boldsymbol{v}_{i,t-1}), & \\ \qquad i = \underbrace{\arg\min}_{k} \{\|\boldsymbol{x} - \boldsymbol{v}_{k,t-1}\|^2\} & \\ 0; \qquad j \neq i & \end{array} \right\}. \tag{19}$$

The right side of (19) is, up to the scale factor 2 (which can be dropped without loss), identical to (2). Thus, GLVQ-F reduces to LVQ when m approaches 1 from above.

In some cases we examined, gradient (15a) becomes $(-2)(\boldsymbol{x} - \boldsymbol{v}_i)$ (positive number) for the winner prototype but $(-2)(\boldsymbol{x} - \boldsymbol{v}_k)$ (negative number) for the loser prototypes. This corresponds to rewarding the losers (excitation) and penalizing the winners (inhibition). Combining this observation with (18) and (19) suggests that the parameter m controls the overall learning rates applied to the updates in an interesting way. Evidently, m can be used to transition the model from strict excitation of the winner (m very close to one) to mixed excitation-inhibition (for m close to one) to unequal excitation of all nodes (as m gets larger), to equal excitation of all nodes (as m goes to infinity). Thus, our belief is that the GLVQ-F model can represent a wide range of biologically inspired learning paradigms through adjustments to a simple weighting parameter. At present, however, the exact mechanism for the transition from one state to another is unknown to us (and may be a data-dependent event). This aspect of the model deserves a careful study before further assertions about it can be made with much confidence.

IV. Conclusions

An experimental investigation with the well known IRIS data using the GLVQ algorithm given in [11] shows that GLVQ produces good results on IRIS, but terrible results on IRIS/10. Combining our results with those reported by Gonzalez *et al.* [12] leads to an inescapable conclusion: GLVQ is best regarded as a faltered step on the way to better competitive learning models.

An infinite family of competitive learning schemes called GLVQ-F are given that overcomes the data scaling problem. We have shown that GLVQ-F is invariant to scaling. More importantly, GLVQ-F offers users a wide range of neural-like responses. We have shown that GLVQ-F reduces to LVQ as its weighting exponent approaches one from above. As m increases, it then transitions to a model in which either all nodes may be excited inversely proportionally to their distances from an input or in which the winner is penalized while losers are excited. And as m increases without limit, GLVQ-F updates all nodes equally. This behavior is probably related to epochs of ascent, descent and limit cycling of the optimization being performed, and needs to be carefully studied before stronger conclusions are warranted. A small by-product of this investigation is a very good initialization strategy for iterative algorithms that produce estimates of prototypes from unlabeled data.

A general characteristic of LVQ-like schemes is their sequential nature, which essentially updates one or more quantizers after each look at a datum (this includes the GLVQ-F model). There are batch versions of these schemes that attempt to overcome this limitation—most of them simply average the updates and apply them after a pass through X. This stabilizes the movement of the prototypes somewhat, but it is still the case that each term in the update equation is determined entirely by local information (at a single data point). We believe that batch models such as hard and fuzzy c-means [2] and descending FLVQ [10], whose algorithms attempt to minimize a global function of all n data points, have a better chance of producing useful vector quantizers in most if not all application domains. We plan an investigation that shows this to be the case.

An important point brought up by Gonzalez *et al.* [12] that was not studied here deserves special mention. It is that the quality of the prototypes may importantly depend on c, the number of them. If c is low, the scheme we have outlined here, as well as that proposed by Yair *et al.* [9] may insulate the model from local instability by distributing the updates across all c prototypes. But since the overall learning rates in these schemes satisfy the constraints $0 < \sum_{i=2}^{c} \alpha_{ik,t} < 1$, if the update for any prototype (the winner) is large, the constraints force the other updates to become minute. Thus, as c increases (say, e.g., as is the case in image compression, where $c = 256$ is not uncommon), it may well be that the original LVQ (winner take all) strategy or some less stringent variation than GLVQ-F becomes more and more reliable (and attractive). This important possibility will by the subject of a future investigation.

APPENDIX A
COMPUTATION OF (15a)

See (15a) shown at the bottom of the page.

APPENDIX B
PROOF OF (17)

1) Proposition: Let $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\} \subset \Re^p$ and let $\boldsymbol{V}_0 = (\boldsymbol{v}_{1,0}, \boldsymbol{v}_{2,0}, \cdots, \boldsymbol{v}_{c,0}), \boldsymbol{v}_{i,0} \in \Re^p$, be a set of initial prototypes. Let τ be a fixed positive number and define sets of scaled data and initial prototypes by $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots \boldsymbol{y}_n\} = \tau X = \{\tau \boldsymbol{x}_1, \tau \boldsymbol{x}_2, \cdots, \tau \boldsymbol{x}_n\}$ and $\boldsymbol{W}_0 = (\boldsymbol{w}_{1,0}, \boldsymbol{w}_{2,0}, \cdots, \boldsymbol{w}_{c,0}) = \tau \boldsymbol{V}_0 = (\tau \boldsymbol{v}_{1,0}, \tau \boldsymbol{v}_{2,0}, \cdots, \tau \boldsymbol{v}_{c,0})$. Then applying GLVQ-F to X, initialized by $\boldsymbol{V}_0$, is equivalent to applying GLVQ F to Y, initialized by $\boldsymbol{W}_0$, in the sense that

$$\boldsymbol{w}_{j,t} = \tau \boldsymbol{v}_{j,t}; \qquad j = 1, \cdots, c \text{ and } t = 0, \cdots. \tag{17}$$

Proof: We use induction. Result (17) holds for $t = 0$ by hypothesis. We now assume it holds for arbitrary $t - 1$, and show it must hold for t. By (16a), for $j = 1, \cdots, c$

$$\begin{aligned}
\boldsymbol{w}_{j,t} &= \boldsymbol{w}_{j,t-1} + \left(\frac{2\alpha_t u_j}{m-1}\right)\left\{u_j \sum_{k=1}^{c}\left(\frac{\|\boldsymbol{y} - \boldsymbol{w}_j\|^{2/(m-1)}}{\|\boldsymbol{y} - \boldsymbol{w}_k\|^{2/(m-1)}}\right)^{2-m} + (m-2)\right\}(\boldsymbol{y} - \boldsymbol{w}_{j,t-1}) \\
&= \tau \boldsymbol{v}_{j,t-1} + \left(\frac{2\alpha_t u_j}{m-1}\right) \cdot \left\{u_j \sum_{k=1}^{c}\left(\frac{\|\tau \boldsymbol{x} - \tau \boldsymbol{v}_j\|^{2/(m-1)}}{\|\tau \boldsymbol{x} - \tau \boldsymbol{v}_k\|^{2/(m-1)}}\right)^{2-m} + (m-2)\right\} \cdot (\tau \boldsymbol{x} - \tau \boldsymbol{v}_{j,t-1}) \\
&= \tau\left[\boldsymbol{v}_{j,t-1} + \left(\frac{2\alpha_t u_j}{m-1}\right) \cdot \left\{u_j \sum_{k=1}^{c}\left(\frac{\|\boldsymbol{x} - \boldsymbol{v}_j\|^{2/(m-1)}}{\|\boldsymbol{x} - \boldsymbol{v}_k\|^{2/(m-1)}}\right)^{2-m} + (m-2)\right\} \cdot (\boldsymbol{x} - \boldsymbol{v}_{j,t-1})\right] = \tau \boldsymbol{v}_{j,t}.
\end{aligned}$$

$$\begin{aligned}
\nabla_{\boldsymbol{v}_s} L &= \nabla_{\boldsymbol{v}_s}\left(\sum_{r=1}^{c} u_r \|\boldsymbol{x} - \boldsymbol{v}_r\|^2\right) = \nabla_{\boldsymbol{v}_s}\left(\sum_r \frac{\|\boldsymbol{x} - \boldsymbol{v}_r\|^2}{\sum_j \|\boldsymbol{x} - \boldsymbol{v}_r\|^{2/(m-1)}}{\|\boldsymbol{x} - \boldsymbol{v}_j\|^{2/(m-1)}}\right) = \nabla_{\boldsymbol{v}_s}\left(\sum_r \frac{(\|\boldsymbol{x} - \boldsymbol{v}_r\|^2)^{(m-2)/(m-1)}}{\sum_j (\|\boldsymbol{x} - \boldsymbol{v}_j\|^2)^{-1/(m-1)}}\right) \\
&= \nabla_{\boldsymbol{v}_s}\left(\frac{\sum_r (\|\boldsymbol{x} - \boldsymbol{v}_r\|^2)^{(m-2)/(m-1)}}{\sum_j (\|\boldsymbol{x} - \boldsymbol{v}_j\|^2)^{-1/(m-1)}}\right) = \left[\frac{\left(\frac{m-2}{m-1}\right)\|\boldsymbol{x} - \boldsymbol{v}_k\|^2)^{-1/(m-1)}(-2)(\boldsymbol{x} - \boldsymbol{v}_k)\sum_j (\|\boldsymbol{x} - \boldsymbol{v}_j\|^2)^{-1/(m-1)}}{\left(\sum_j (\|\boldsymbol{x} - \boldsymbol{v}_j\|^2)^{-1/(m-1)}\right)^2}\right. \\
&\quad \left. - \frac{\sum_r (\|\boldsymbol{x} - \boldsymbol{v}_r\|^2)^{(m-2)/(m-1)}\left(\frac{-1}{m-1}\right)(\|\boldsymbol{x} - \boldsymbol{v}_k\|^2)^{-m/(m-1)}(-2)(\boldsymbol{x} - \boldsymbol{v}_k)}{\left(\sum_j (\|\boldsymbol{x} - \boldsymbol{v}_j\|^2)^{-1/(m-1)}\right)^2}\right] \\
&= \left(\frac{-2}{m-1}\right)(\boldsymbol{x} - \boldsymbol{v}_k)\left[\frac{(m-2)(\|\boldsymbol{x} - \boldsymbol{v}_k\|^{-2/(m-1)})}{\sum_j (\|\boldsymbol{x} - \boldsymbol{v}_j\|^{-2/(m-1)})}\right. \\
&\quad \left. + \frac{\sum_r (\|\boldsymbol{x} - \boldsymbol{v}_r\|^{2/(m-1)})^{m-2}(\|\boldsymbol{x} - \boldsymbol{v}_k\|^{2/m-1(m-1)})^{2-m}(\|\boldsymbol{x} - \boldsymbol{v}_k\|^{-2/(m-1)})^2}{\left(\sum_j (\|\boldsymbol{x} - \boldsymbol{v}_j\|^{-2/(m-1)})\right)^2}\right] \\
&= \left(\frac{2}{m-1}\right)(\boldsymbol{x} - \boldsymbol{v}_k)\left[(m-2)u_k + \left(\sum_r \left(\frac{\|\boldsymbol{x} - \boldsymbol{v}_k\|^{2/(m-1)}}{\|\boldsymbol{x} - \boldsymbol{v}_r\|^{2/(m-1)}}\right)^{2-m}\right)u_k^2\right].
\end{aligned} \tag{15a}$$

To complete the proof, observe that the membership values $\{u_r\}$ for X and Y are identical, because they involve ratios of norm values, so that the scaling factor τ cancels out, just as in

$$\frac{\|y-w_r\|^{2/(m-1)}}{\|y-w_k\|^{2/(m-1)}} = \frac{\|\tau x - \tau v_r\|^{2/(m-1)}}{\|\tau x - \tau v_k\|^{2/(m-1)}} = \frac{\tau^{2/(m-1)}\|x-v_r\|^{2/(m-1)}}{\tau^{2/(m-1)}\|x-v_k\|^{2/(m-1)}} = \frac{\|x-v_r\|^{2/(m-1)}}{\|x-v_k\|^{2/(m-1)}}.$$

■

Appendix C
Computation of (18)

Rewrite the gradient equation (15a)

$$\nabla_{v_k} L = \left(\frac{2}{m-1}\right)(x-v_k)\cdot\left[(m-2)u_k - \left(\sum_r \left(\frac{\|x-v_k\|^{2/m-1}}{\|x-v_r\|^{2/m-1}}\right)^{2-m}\right)u_k^2\right] \tag{15a}$$

$$= (-2)(x-v_k)\cdot\left[u_k + \frac{\left(\sum_r \left(\frac{\|x-v_k\|}{\|x-v_r\|}\right)^{(4-2m)/(m-1)}\right) - \frac{1}{u_k}}{m-1}u_k^2\right]$$

$$= (-2)(x-v_k)[u_k - F(k,m)]. \tag{15a'}$$

Evaluating the limit in (15a') as $1 \xleftarrow{+} m$ or $m \to \infty$ is complicated by the fact that u_k depends on m in a fairly messy way. To simplify the argument in the following analysis, it is assumed that $\|x - v_r\| > 0$ for $r = 1, \cdots, c$ with no ties in datum to prototype distances. We first consider the case $m \to \infty$, where it is well known that

$$\lim_{m\to\infty}\{u_k\} = 1/c$$

for all k [2]. The numerator of $F(k,m)$ is easily seen to approach the number

$$\left(\sum_r \left(\frac{\|x-v_k\|}{\|x-v_r\|}\right)^{-2} \cdot c\right)\left(\frac{1}{c}\right)^2$$

as $m \to \infty$, while the denominator $(m-1) \to \infty$. This shows that

$$\lim_{m\to\infty}\{F(k,m)\} = 0$$

for all k, which, combined with

$$\lim_{m\to\infty}\{u_k\} = 1/c$$

for all k, establishes that all c prototypes are adjusted equally by (16a) in the limit, i.e.,

$$\lim_{m\to\infty}\{v_{j,t}\} = v_{j,t-1} + \left(\frac{2\alpha_{t-1}}{c}\right)(x - v_{j,t-1})$$
$$j = 1, 2, \cdots, c. \tag{18}$$

■

Appendix D
Computation of (19)

We consider evaluating (15a') as $1 \xleftarrow{+} m$. Let

$$\delta_{k,r} = \left(\frac{\|x-v_k\|}{\|x-v_r\|}\right)^2 \quad \text{and} \quad \Delta_k = \max\{\delta_{k,1}, \cdots, \delta_{k,c}\}.$$

Note that either $\Delta_k = 1$ or $\Delta_k > 1$. Now, $F(k,m)$ can be written as

$$F(k,m) = \frac{\left(\frac{\sum_r \delta_{k,r}^{-1}\delta_{k,r}^{1/(m-1)}}{\left(\sum_r \delta_{k,r}^{1/(m-1)}\right)^2}\right) - \left(\frac{1}{\sum_r \delta_{k,r}^{1/(m-1)}}\right)}{m-1} - \frac{\left(\frac{1}{m-1}\right) * \left(\sum_r (\delta_{k,r}^{-1} - 1)\delta_{k,r}^{1/(m-1)}\right)}{\left(\sum_r \delta_{k,r}^{1/(m-1)}\right) * \left(\sum_r \delta_{k,r}^{1/(m-1)}\right)}. \tag{D.1}$$

We now argue that the limit of (D.1) is zero for both possible cases (that is, for the winner and nonwinner nodes). If $\Delta_k = 1$, then the denominator of (D.1) equals 1 as $1 \xleftarrow{+} m$, while the summation in the numerator has nonzero terms only for $\delta_{k,r} < 1$. An application of L'Hopital's rule in this case will easily establish that the numerator goes to zero. On the other hand, if $\Delta_k > 1$, then it is easy to show that

$$\frac{\left(\frac{1}{m-1}\right)}{\left(\sum_r \delta_{k,r}^{1/(m-1)}\right)} \to 0 \quad \text{and}$$

$$\frac{\left(\sum_r (\delta_{k,r}^{-1} - 1)\delta_{k,r}^{1/(m-1)}\right)}{\left(\sum_r \delta_{k,r}^{1/(m-1)}\right)} \to \Delta_k - 1.$$

So in both possible cases, we have

$$\lim_{1\xleftarrow{+} m}\{F(k,m)\} = 0.$$

It is well known that

$$\lim_{1\xleftarrow{+} m}\{u_k\} = 1$$

for the winning index and 0 otherwise [2]. This means that in the limit $1 \xleftarrow{-} m$, GLVQ-$\mathcal{F}$ becomes LVQ; that is (16a) does not update loser prototypes, and updates the winner prototype $\boldsymbol{v}_i$ by

$$\underbrace{\lim}_{1 \xleftarrow{+} m} \{\boldsymbol{v}_{i,t}\} = \boldsymbol{v}_{i,t-1} + (2\alpha_{t-1})(\boldsymbol{x} - \boldsymbol{v}_{i,t-1}). \quad (19)$$

■

REFERENCES

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

[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, MA: 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, 1988.

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

[8] 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.

[9] E. Yair, K. Zeger, and A. Gersho, "Competitive learning and soft competition for vector quantizer design," *IEEE Trans. Signal Processing*, vol. 40, pp. 294–309, 1992.

[10] E. C. K. Tsao, J. C. Bezdek, and N. R. Pal, "Fuzzy Kohonen clustering networks," *Pattern Recognition*, vol. 27, no. 5, pp. 757–764, 1994.

[11] N. R. Pal, J. C. Bezdek, and E. C. K. Tsao, "Generalized clustering networks and Kohonen's self-organizing scheme," *IEEE Trans. Neural Networks*, vol. 4, pp. 549–558, 1993.

[12] A. I. Gonzalez, M. Graña, and A. D'Anjou, "An analysis of the GLVQ algorithm," *IEEE Trans. Neural Networks*, vol. 6, pp. 1012–1016, 1995.

[13] N. B. Karayiannis and P.-I. Pai, "A fuzzy algorithm for learning vector quantization," in *Proc. 1994 IEEE Int. Conf. Syst., Man, Cybern.*, Piscataway, NJ, 1994, pp. 126–131.

[14] ——, "A family of fuzzy algorithms for learning vector quantization," in *Intelligent Engineering Systems through Artificial Neural Networks*, H. Dagli, B. R. Fernandez, J. Ghosh, and R. T. Kumara, Eds., vol. 4. New York: Amer. Soc. Mech. Eng., 1994, pp. 219–224.

[15] E. Anderson, "The IRISs of the Gaspe peninsula," *Bull. Amer. IRIS Soc.*, vol. 59, pp. 2–5, 1935.

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

[17] D. C. Park and I. Dagher, "Gradient-based fuzzy c-means (GBFCM) algorithm," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 3, pp. 1626–1631, 1994.

[18] F. L. Chung and T. Lee, "Fuzzy competitive learning," *Neural Networks*, vol. 7, no. 3, pp. 539–551, 1992.

Nicolaos B. Karayiannis (S'85–M'91), for a photograph and biography, see p. 426 of the March 1996 issue of this TRANSACTIONS.

James C. Bezdek (M'80–SM'90–F'92) received the Ph.D. degree from Cornell University, Ithaca, NY, in 1973.

He is a Professor of Computer Science at the University of West Florida, Pensacola. His interests include pattern recognition, fishing, computational neural networks, skiing, image processing, blues music, medical computing, and motorcycles.

Dr. Bezdek is Editor-in-Chief of the IEEE TRANSACTIONS ON FUZZY SYSTEMS.

Nikhil R. Pal (M'91) received the M.Tech. and Ph.D. degrees in computer science from the Indian Statistical Institute, Calcutta, in 1984 and 1991, respectively.

He is an Associate Professor in the Machine Intelligence Unit of Physics. He was with the Hindusthan Motors Ltd. W. B., from 1984 to 1985 and with the Dunlop India Ltd., W. B., from 1985 to 1987. In 1987 he joined the Computer Science Unit of the Indian Statistical Institute, Calcutta. During August 1991 to February 1993 and July 1994 to December 1994 he visited the University of West Florida, Pensacola. He was a Guest Lecturer at the University of Calcutta. His research interests include image processing, pattern recognition, fuzzy sets and systems, uncertainty measures, genetic algorithms, and neural networks.

Dr. Pal is an Associate Editor of the IEEE TRANSACTIONS ON FUZZY SYSTEMS and the *International Journal of Approximate Reasoning*.

Richard J. Hathaway (M'93) received the B.S. degree in applied mathematics from the University of Georgia, Athens, in 1979 and the Ph.D. degree in mathematical sciences from Rice University, Houston, TX, in 1983.

He is a Professor in the Mathematics and Computer Science Department of Georgia Southern University, Statesboro. His research interests include pattern recognition, statistical computing, and nonlinear optimization.

Pin-I Pai, for a photograph and biography, see this issue, p. 1211.