
4 Finite-Sample Convergence Properties of the LVQ1 algorithm, the Batch LVQ1 Algorithm and the Generalised LVQ1 algorithm

Abstract- This chapter address the convergence of Kohonen's LVQ1 algorithm when the number of training samples are finite with an analysis that uses the dynamical systems and optimization theories. It establishes the sufficient conditions to ensure the convergence of LVQ1 near a minimum of its cost function for constant step sizes and cyclic sampling. It is also proposed a batch version of LVQ1 based on the very fast Newton optimisation method that cancels the dependence of the on-line version on the order of supplied training samples and a straightforward generalization of LVQ1 that increases the probability of reaching a minimum point of the training error.

Index Terms- LVQ1 algorithm, Asymptotic convergence, Online gradient descent, Finite-sample properties, BLVQ1 algorithm, GLVQ1 algorithm, Newton optimisation.

1. Introduction

Kohonen's LVQ1 algorithm (Kohonen, 1996) is a powerful on-line gradient learning algorithms for non-parametric statistical pattern recognition based on nearest neighbour (NN) classification. Asymptotic convergence of LVQ1 has only been studied when the number of training samples (N) tends to ∞ (e.g. (LaVigna, 1990)(Zhu et al., 1995)) using the stochastic approximation theory (Bottou, 1998)(Benveniste et al., 1990). When $N < \infty$, the use of LVQ1 was only supported by empirical evidence since there was no guarantee of convergence from a theoretical point of view. However, in this chapter, we establish the sufficient conditions that ensure the convergence of LVQ1 near a minimum of its cost function E_{LVQ1} . Our analysis is based on the use of the discrete-time dynamical systems and optimization theories (Devaney, 1989) for the simplest case: training data are cyclically sampled and the step sizes are constant.

In the next section, LVQ1 is introduced as an on-line gradient algorithm and the properties of its cost function E_{LVQ1} are analysed. Section 3 presents the study of the finite-sample convergence. Since LVQ1 is sensitive to the order of training samples, we introduce in section 4 a batch version of LVQ1 (BLVQ1) based on the Newton optimization method that cancels this dependence and speeds up the convergence rate. Section 5 presents a straightforward generalization of the LVQ1 algorithm that increases the probability of reaching a minimum point of the training error. In section 6, some examples are presented to illustrate several aspects about LVQ1 and BLVQ1. Finally, some discussion and conclusions are given.

2. LVQ1 as an On-line Gradient Descent algorithm

Suppose we have N observations pairs $\mathbf{D}_N = \{(\mathbf{x}_i, \text{cl}(\mathbf{x}_i)), i = 1, \dots, N\}$ where $\mathbf{x}_i \in \mathfrak{R}^p$ is a random sample that belongs to one of the c classes and $\text{cl}(\mathbf{x}_i)$ is the class label associated to pattern \mathbf{x}_i . Our goal in the learning phase is to design a set of labelled prototypes (or codebook) $\mathbf{C} = [\mathbf{m}_1^T \mathbf{m}_2^T \dots \mathbf{m}_K^T]^T$ for an Euclidean Nearest Neighbor (NN) classifier using \mathbf{D}_N . There are K Voronoi regions $R_j = \left\{ \mathbf{x} \mid \|\mathbf{x} - \mathbf{m}_j\| = \min_{i=1, \dots, K} \|\mathbf{x} - \mathbf{m}_i\| \right\} j=1, \dots, K$ where the classifier maps any input pattern that falls in it to the class which its codevector $\mathbf{m}_j \in \mathfrak{R}^p$ belongs. The LVQ1 algorithm applies to compute \mathbf{C} the *pattern-based version* of gradient descent over the following cost function:

$$E_{LVQ1}(\mathbf{C}) = \frac{1}{2} \sum_{i=0}^{N-1} \sum_{j=1}^K \mathbf{1}(\mathbf{x}_i \in R_j) \mathbf{1}(\text{cl}(\mathbf{x}_i) = \text{cl}(\mathbf{m}_j)) \|\mathbf{x}_i - \mathbf{m}_j\|^2 - \frac{1}{2} \sum_{i=0}^{N-1} \sum_{j=1}^K \mathbf{1}(\mathbf{x}_i \in R_j) \mathbf{1}(\text{cl}(\mathbf{x}_i) \neq \text{cl}(\mathbf{m}_j)) \|\mathbf{x}_i - \mathbf{m}_j\|^2 \quad (1)$$

where the $\text{cl}(\mathbf{x})$ function returns the class label of \mathbf{x} and $\mathbf{1}(\text{condition})$ is the indicator function which is 1 if condition is true and 0 otherwise. The interest in the minimisation of E_{LVQ1} for classification can be justified since the minimum points of E_{LVQ1} ensure that the decisions of the resulting LVQ1-based NN classifier agree with a majority vote among the labelled training data that fall in each Voronoi cell (table 1). Although the minimisation of the classification error is not well addressed by this procedure (see example 2 in section 5), its convergence to the Bayes error as $K, N \rightarrow \infty$ could be investigated (section 6).

2.1. Critical Points of E_{LVQ1}

The critical points of E_{LVQ1} include those \mathbf{C} in which the learning algorithm has fixed points ($\nabla E_{LVQ1}(\mathbf{C}) = 0$) and a change in its dynamics ($\nabla E_{LVQ1}(\mathbf{C})$ does not exist). Solving the equation $\nabla E_{LVQ1}(\mathbf{C}) = 0$ yields:

$$\mathbf{m}_j^* = \frac{\sum_{i=1}^N \mathbf{1}(\mathbf{x}_i \in R_j) (\mathbf{1}(\text{cl}(\mathbf{x}_i) = \text{cl}(\mathbf{m}_j)) - \mathbf{1}(\text{cl}(\mathbf{x}_i) \neq \text{cl}(\mathbf{m}_j))) \mathbf{x}_i}{\sum_{i=1}^N \mathbf{1}(\mathbf{x}_i \in R_j) (\mathbf{1}(\text{cl}(\mathbf{x}_i) = \text{cl}(\mathbf{m}_j)) - \mathbf{1}(\text{cl}(\mathbf{x}_i) \neq \text{cl}(\mathbf{m}_j)))} \quad j = 1 \dots K \quad (2)$$

We must then compute the Hessian matrix $\mathbf{H} = \nabla^2 E_{LVQ1}(\mathbf{C})$ at $\mathbf{C}^* = [\mathbf{m}_1^{*T} \mathbf{m}_2^{*T} \dots \mathbf{m}_K^{*T}]^T$ to know when these fixed points behave as minimum, maximum or saddle points of E_{LVQ1} . \mathbf{H} can be decomposed in blocks according to each pair of codevectors

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 E_{LVQ1}}{\partial^2 \mathbf{m}_1} & \dots & \frac{\partial^2 E_{LVQ1}}{\partial \mathbf{m}_1 \partial \mathbf{m}_K} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 E_{LVQ1}}{\partial \mathbf{m}_K \partial \mathbf{m}_1} & \dots & \frac{\partial^2 E_{LVQ1}}{\partial^2 \mathbf{m}_K} \end{bmatrix} \quad (3)$$

The first and second partial derivatives of E_{LVQ1} respect to codevectors give

$$\frac{\partial E_{LVQ1}[VQ]}{\partial \mathbf{m}_j} = - \sum_{i=1}^N 1(\mathbf{x}_i \in R_j) (1(\text{cl}(\mathbf{x}_i) = \text{cl}(\mathbf{m}_j)) - 1(\text{cl}(\mathbf{x}_i) \neq \text{cl}(\mathbf{m}_j))) (\mathbf{x}_i - \mathbf{m}_j) \quad (4)$$

$$\frac{\partial^2 E_{LVQ1}[VQ]}{\partial \mathbf{m}_i \partial \mathbf{m}_j} = \begin{cases} \mathbf{0} & \text{if } i \neq j \\ \left(2N_{m_j} - N_j \right) \mathbf{I} & \text{if } i = j \end{cases}$$

where \mathbf{I} is the $p \times p$ identity matrix, N_j is the number of training samples that fall in the Voronoi region R_j and N_{mj} is the number of training points that fall in R_j belonging to the same class than \mathbf{m}_j . Since \mathbf{H} is a constant diagonal matrix, its pK eigenvalues λ_i are $\{\lambda_{p(j-1)+m} = 2N_{m_j} - N_j, m = 1, \dots, p, j = 1, \dots, K\}$. Thus, if we apply well known results of differential calculus (e.g. (Hiriart-Urruty, 1993)), we can obtain when \mathbf{C}^* will behave as an attractor, a repulsor or a bifurcation point of a gradient descent learning algorithm (see table 1). For any possible dichotomy of the training samples, the learning system could have a fixed point \mathbf{C}^* . However the number of these fixed points are reduced since \mathbf{C}^* will exist only if it implements the dichotomy that induces it (see example 2).

The value of E_{LVQ1} in these points can be either positive or negative depending on the problem. E.g. $E_{LVQ1} < 0$ in a minimum point would indicate that codevectors are (in average) nearer samples of the same class than samples belonging to other classes in each R_j (example 2). On the other hand, we could have a minimum with positive cost when Voronoi cells have much more data belonging to the same class than data of other classes assuming that distances to codevectors are (more or less) the same for all training data in each Voronoi cell. Therefore, the sign of the cost function at the minimum points is irrelevant here. The key point is that minimum points ensures that $N_{mj} > N_j/2$ for all $j=1, \dots, K$.

Eigenvalues of $\mathbf{H}(\mathbf{C}^*) = \nabla^2 E_{LVQ1}(\mathbf{C}^*)$	In \mathbf{C}^* , E_{LVQ1} has a	The learning system will see \mathbf{C}^* as
$\lambda_i > 0 \ i=1 \dots K \times p \Leftrightarrow 2N_{m_j} > N_j \text{ for all } j=1 \dots K$	<i>Minimum</i>	<i>Attractor</i>
$\lambda_i < 0 \ i=1 \dots K \times p \Leftrightarrow 2N_{m_j} < N_j \text{ for all } j=1 \dots K$	<i>Maximum</i>	<i>Repulsor</i>
Some $\lambda_i \geq 0$ and $\lambda_j \leq 0$	<i>Saddle point</i>	<i>Bifurcation point</i>

Table 1. Fixed points of E_{LVQ1}

Finally, $\nabla E_{LVQ1}(\mathbf{C})$ does not exist for those \mathbf{C} in which any training sample falls in a *Voronoi border* (see examples). In these points, the dynamics of the learning system change due to the

discontinuity of E_{LVQ1} . This conduct can induce, in some cases, spurious periodic points since the learning system can be trapped between different dynamics. In some cases, they could be useful (e.g. oscillation between attraction basins; see example 4) but they also could cause the learning system to reach unexpected (and undesired) solutions (e.g. oscillation between repulsion basins).

2.2. The LVQ1 algorithm

Since LVQ1 uses on-line gradient descent over equation 1, we update each codevector \mathbf{m}_j with

$$\mathbf{m}_j[n+1] = \begin{cases} \mathbf{m}_j[n] + \alpha[n](\mathbf{x}[n+1] - \mathbf{m}_j[n]) & \text{if } \mathbf{x}[n+1] \in R_j[n] \text{ and } \mathbf{x}[n+1], \mathbf{m}_j \in \text{same class} \\ \mathbf{m}_j[n] - \alpha[n](\mathbf{x}[n+1] - \mathbf{m}_j[n]) & \text{if } \mathbf{x}[n+1] \in R_j[n] \text{ and } \mathbf{x}[n+1], \mathbf{m}_j \notin \text{same class} \end{cases} \quad j=1 \dots K \quad (5)$$

where $\alpha[n]$ is the step size function that typically belongs to the interval (0,1). The geometric interpretation of this update equation clarifies the utility of LVQ1 for classification. Codevectors are moved away from samples of other classes but they come close to samples belonging to the same class. Note that in the cyclic version, LVQ1 samples as follows: $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{N-1}, \mathbf{x}_0, \mathbf{x}_1, \dots$

3. Finite-Sample Asymptotic Convergence of LVQ1

We will study here the case in which the training data is sampled cyclically and the step sizes $\{\alpha_j\}$ are constant since the convergence equations are simple enough to be analysed. Since LVQ1 performs on-line gradient descent over E_{LVQ1} , codevectors might minimise locally E_{LVQ1} . However, as we will show, LVQ1 only follows the gradient path, and hence it can converge to a minimum point of a particular cost function from any initial point, when α are smaller than the so-called convergence bounds $\{U_j^{\text{conv}} \quad j=1, \dots, K\}$. Besides, the step sizes must be below the optimization error bounds $\{U_j^{\text{opt error}} \quad j=1, \dots, K\}$ to ensure the convergence of LVQ1 near a local minimum of E_{LVQ1} .

3.1. Case $K=1$

If $K=1$ the learning equation is non-linear since there is a change of sign that depends on the class label of the training patterns. To get some insight about how to derive $\mathbf{m}[n]$, suppose the following example: \mathbf{m} belongs to A and sees this data sequence: \mathbf{x}_1 (A), \mathbf{x}_2 (B), \mathbf{x}_3 (B),... According to the update equation (5): $\mathbf{m}[1] = \mathbf{m}[0](1-\alpha) + \alpha\mathbf{x}[1]$, $\mathbf{m}[2] = \mathbf{m}[0](1-\alpha)(1+\alpha) + \alpha(1+\alpha)\mathbf{x}[1] - \alpha\mathbf{x}[2]$ and $\mathbf{m}[3] = \mathbf{m}[0](1-\alpha)(1+\alpha)^2 + \alpha(1+\alpha)^2\mathbf{x}[1] - \alpha(1+\alpha)\mathbf{x}[2] + \alpha\mathbf{x}[3]$. As we can see, $\mathbf{m}[n]$ has two different terms that depends on initial conditions ($\mathbf{m}[0]$) and data. In the

first term, $\mathbf{m}[0]$ is multiplied by $(1-\alpha)^{p[n]}$ where $p[n]$ is the number of samples in the sequence $\{\mathbf{x}[k], 0 \leq k < n\}$ that agree with \mathbf{m} 's label, and by $(1+\alpha)^{n-p[n]}$. The second term is a sum of weighted data. Each data $\mathbf{x}[k]$ ($0 < k \leq n$) is weighted by the multiplier factors $(1-\alpha)$ or $(1+\alpha)$ according to the class label of the sequence $\{\mathbf{x}[q], k < q < n\}$ using the same criterion than before. Finally, each training data is multiplied by α or $-\alpha$ depending on its class label. If we generalise this simple case, we will be able to compute $\mathbf{m}[n]$ when we just have passed one epoch:

$$\begin{aligned} \mathbf{m}[N] &= g(\alpha, 0)\mathbf{m}[0] + \mathbf{B}; \\ \mathbf{B} &= \sum_{i=1}^N \alpha (1(\text{cl}(\mathbf{x}[i]) = \text{cl}(\mathbf{m})) - 1(\text{cl}(\mathbf{x}[i]) \neq \text{cl}(\mathbf{m})))g(\alpha, i)\mathbf{x}[i] \end{aligned} \quad (6)$$

The weighted function $g(\alpha, i)$ depends on α and the position i in the array memory of training samples in the following way

$$g(\alpha, i) = (1 - \alpha)^{N_m[i]} (1 + \alpha)^{N[i] - N_m[i]} \quad (7)$$

where $N_m[i]$ is the number of training samples in the memory array after position i that belong to the same class than \mathbf{m} and $N[i]$ is the number of the remaining training samples in the memory array after position i . Clearly, $N[i] = N - i$ and $N_m[0] = N_m$ where N_m is the number of training samples that belong to the same class than \mathbf{m} .

Since $\mathbf{x}[n]$ is a periodic signal of period N , the algorithm repeats the same computations of the first epoch in the second one where now $\mathbf{m}[N]$ is the initial value, giving account of the past history. Consequently, we can write down that $\mathbf{m}[2N] = A\mathbf{m}[N] + \mathbf{B} = A^2\mathbf{m}[0] + (A + 1)\mathbf{B}$ where $A = g(\alpha, 0)$. Then we compute $\mathbf{m}[cN]$ by induction,

$$\mathbf{m}[cN] = A\mathbf{m}[(c-1)N] + \mathbf{B} = A^c\mathbf{m}[0] + \sum_{i=0}^{c-1} A^i\mathbf{B} \quad (8)$$

Let us introduce an equivalent discrete dynamical system of equation (8), $m_i[c] = f(m_i[c-1])$ $i = 1 \dots p$; $f(m_i) = m_i A + B_i$ with $n = cN$. Thus we can apply discrete-time dynamical systems theory that deals with 1-D systems of the form $m[c] = f(m[c-1])$ (Devaney, 1989). According to (Devaney, 1989)§1.4 the behaviour of the fixed points $m_i^* = f(m_i^*)$ only depends on $|A|$. E.g. these fixed points are attractors if $|A| < 1$ and then the

convergence rate depends on the value of A. In the two following points we will analyse the cases in which the algorithm converges very slowly $|A| \rightarrow 1$ ($\alpha \rightarrow 0$) and very fast $|A| \rightarrow 0$ ($\alpha \gg 0$).

3.1.1. LVQ1 as a gradient descent algorithm

When $\alpha \rightarrow 0$, the function $A(\alpha)$ can be approximated by its first order Taylor series expansion. Then we will use the approximation to derive the necessary conditions of convergence of the discrete sequence $\{\mathbf{m}[cN], c>0\}$. The Taylor series expansion of $A(\alpha)$ around $\alpha=0$ is

$$\begin{aligned} A(\alpha) &= A(0) + A'(0)\alpha + A''(0)\frac{\alpha^2}{2} + A'''(0)\frac{\alpha^3}{6} + O(\alpha^4) \\ A(0) &= S \\ A'(0) &= S^2 - N \\ A''(0) &= N(S+2) - 3N_m + 4N_m N(S+1) - N^2(S+2) - 4N_m^2(S+1) \end{aligned} \quad (9)$$

where $S=2N_m-N$. Then we can approximate A by $A \approx 1 - S\alpha$ when

$$\begin{aligned} \text{i) } |A'(0)\alpha| \gg |A''(0)\alpha^2/2| &\Rightarrow \alpha \ll U_1^{\text{conv}} \text{ where } U_1^{\text{conv}} = 2/|S - N/S| \text{ for } S^2 \neq N \\ \text{ii) } |A'(0)\alpha| \gg |A'''(0)\alpha^3/6| &\Rightarrow \alpha \ll U_2^{\text{conv}} \text{ where } U_2^{\text{conv}} = 6|A'(0)|/|A'''(0)| \text{ for } S^2 = N \end{aligned} \quad (10)$$

Practically, we must ensure that $\alpha < U_1^{\text{conv}}/\kappa$ for $S^2 \neq N$ and $\alpha < U_2^{\text{conv}}/\kappa$ for $S^2 = N$ where κ is a constant (e.g. $\kappa=10$). Figure 1 shows the absolute error between A and its first order Taylor series expansion ($|A(\alpha) - 1 + S\alpha|$) when $N=500$, $N_m=470$ ($S^2 \neq N$) and $N=324$ and $N_m=171$ ($S^2 = N$). We can observe that the error is quite low for alpha's that are ten times lower than the corresponding upper bounds U_1^{conv} (0.0046) and U_2^{conv} (0.0320).

If alpha is lower than U^{conv}/κ , the LVQ1 algorithm can follow a gradient path since equation (8) with $A \approx 1 - S\alpha$ can be derived applying the batch version of the gradient descent method over the following error function:

$$E_{\text{LVQ1}}^{\text{on-line}}(\mathbf{m}) = \frac{1}{2} \sum_{i=0}^{N-1} 1(\mathbf{x}_i \in R_j) (1(\text{cl}(\mathbf{x}_i) = \text{cl}(\mathbf{m})) - 1(\text{cl}(\mathbf{x}_i) \neq \text{cl}(\mathbf{m}))) \|g(\alpha, i)\mathbf{x}_i - \mathbf{m}\|^2 \quad (11)$$

Hence, one epoch (or N steps) of LVQ1 for $K=1$ is equivalent to one iteration of a batch gradient algorithm over equation (11). In correspondence with (Devaney, 1989), the

sufficient condition to ensure that LVQ1 converges is $0 < S < 2/\alpha$ and $\alpha < U^{\text{conv}}/\kappa$ ($\kappa > 1$) where U^{conv} is computed using equation (10) (see table 2). In other words, if $\alpha < 2/S$ and $\alpha < U^{\text{conv}}/\kappa$ then LVQ1 follows the gradient path and it converges near a local minimum of E_{LVQ1} , that is a fixed point of E_{LVQ1} with $S > 0$. Then $\mathbf{m}[\infty]$ gives

$$\lim_{c \rightarrow \infty} \mathbf{m}[cN] \approx \lim_{c \rightarrow \infty} \left((1 - \alpha S)^c \mathbf{m}[0] + \sum_{i=0}^{c-1} (1 - \alpha S)^i \mathbf{B} \right) = (1 - \alpha S)^\infty \mathbf{m}[0] + \sum_{i=0}^{\infty} (1 - \alpha S)^i \mathbf{B} = \frac{\mathbf{B}}{(1 - (1 - \alpha S))} = \frac{\mathbf{B}}{\alpha S} \quad (12)$$

$$= \frac{1}{S} \sum_{i=1}^N (1(\text{cl}(\mathbf{x}[i]) = \text{cl}(\mathbf{m})) - 1(\text{cl}(\mathbf{x}[i]) \neq \text{cl}(\mathbf{m}))) g(\alpha, i) \mathbf{x}[i]$$

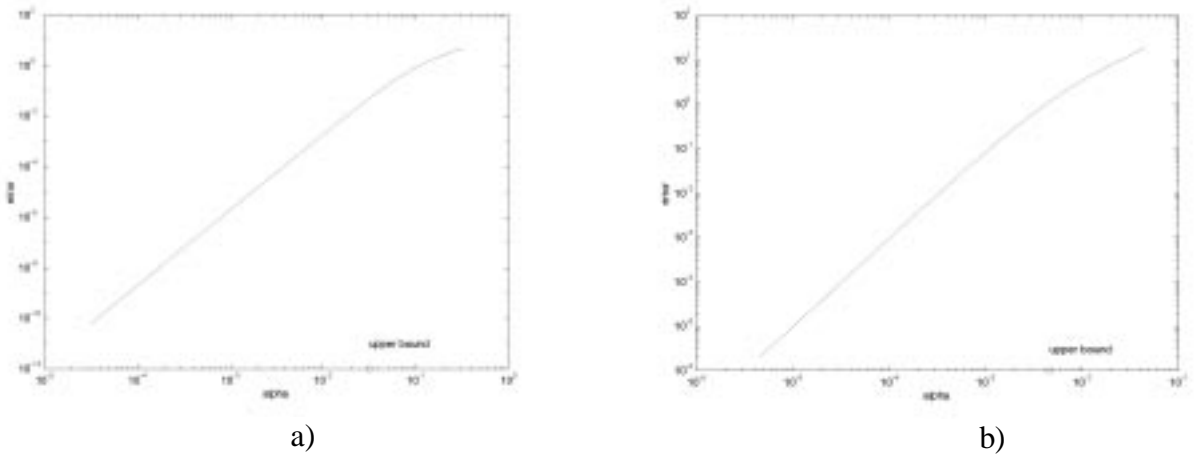


Fig.1. The error function $|A(\alpha) - 1 + S\alpha|$ for a) $N=500$ and $N_m=470$, b) $N=324$ and $N_m=171$.

Clearly, the equilibrium point of LVQ1 is not near a minimum point of E_{LVQ1} since there is an undesired weighted function $g(\alpha, i)$ that modifies the contribution of each training sample $\mathbf{x}[i]$. In other words, LVQ1 converges to a local minimum of $E_{\text{LVQ1}}^{\text{on-line}}$ instead of a local minimum of E_{LVQ1} . To alleviate this optimization error, $g(\alpha, i)$ might tend to the unity to ensure that $E_{\text{LVQ1}}^{\text{on-line}} \rightarrow E_{\text{LVQ1}}$ but then α would have to arbitrary decreased to zero. However, a small optimization error can be achieved when $g(\alpha, i)$ can be approximated by $1 - S[i]\alpha$. This can be ensured for all the possible cases only when $\alpha < 2/\kappa N_m$ (see annex). If this condition is met then equation 12 yields

$$\mathbf{m}[\infty] \approx \frac{1}{2N_m - N} \sum_{i=0}^{N-1} (1(\text{cl}(\mathbf{x}[i]) = \text{cl}(\mathbf{m})) - 1(\text{cl}(\mathbf{x}[i]) \neq \text{cl}(\mathbf{m}))) (1 - \alpha(N - 2n_i^m - i)) \mathbf{x}[i] \quad (13)$$

As equation 13 shows, the attractor of the learning system is now near a minimum point of E_{LVQ1} . However, it gives more relevance to those samples that are at the end of the memory array according to a weighted function. Besides the rate of convergence to it depends on S (table 2). On the other hand, the learning system is structurally stable ((Devaney, 1989) §1.9). Therefore, it exists a persistence of the learning dynamics under small perturbations in the step size α (e.g. $\mathbf{m}_\alpha[\infty] \approx \mathbf{m}_{\alpha+\varepsilon}[\infty]$, $\varepsilon \ll \alpha$).

$ \partial f(m_i^*)/\partial m_i = A $; $m_i^* = f(m_i^*)$ (Devaney, 1989)	m_i^* is a	Sufficient conditions
<1 and $0 < A < 1$	Exponential Attractor with exponent $1-S\alpha$	$S > 0$ and $\alpha < 1/S$
<1 and $-1 < A \leq 0$	Slow Attractor	$S > 0$ and $1/S \leq \alpha < 2/S$
>1	Repulsor	$S < 0$ or $\alpha > 2/S$
$=1$	Bifurcation point	$\alpha = 2/S$

Table 2. Fixed points of the LVQ1 algorithm for $K=1$ $\alpha < U$ where $U = U^{\text{conv}} / \kappa$. Note that LVQ1 can be expelled from a minimum of $E_{LVQ1}^{\text{on-line}}$ if alpha is not bounded enough.

3.1.2. LVQ1 as a quasi-gradient descent algorithm

If $S > 0$ and $\alpha \in (0,1)$, the function $A(\alpha)$ belongs to the interval $[0,1]$ with $A(0)=1$ and $A(1)=0$. Besides $A(\alpha)$ tends exponentially quick to zero as $\alpha \rightarrow 1$. Therefore, if $\alpha \gg 0$ $A(\alpha)$ cannot be approximated by $A \approx 1 - S\alpha$. Accordingly, equation 8 can be rewritten as

$$\begin{aligned}
 \mathbf{m}[cN] &= \mathbf{m}[(c-1)N] \left(1 - S\alpha + \sum_{i=2}^{\infty} A^{(i)}(0) \frac{\alpha^i}{i!} \right) + \mathbf{B} = \\
 &= \mathbf{m}[(c-1)N] \left(1 + \sum_{i=2}^{\infty} A^{(i)}(0) \frac{\alpha^i}{i!} \right) - \alpha \frac{\partial E_{LVQ1}^{\text{on-line}}[(c-1)N]}{\partial \mathbf{m}[(c-1)N]}
 \end{aligned} \tag{14}$$

Hence LVQ1 does not follows the gradient path, although it converges more quickly to the point $\mathbf{B}/(1-A)$. However, the optimization error of LVQ1 is greater: the equilibrium point is now the minimum of $E_{LVQ1}^{\text{on-line}}$ multiplied by the factor $S\alpha/(1-A) = 1 / \left(1 - \sum_{i=2}^{\infty} A^{(i)}(0) \alpha^{i-1} / i! \right)$.

3.2. Case $K>1$

When the number of codevectors $K>1$, the discrete-time learning system is highly non-linear due to the winner-takes-all process that is performed in each iteration of the algorithm (equation 5). Once we start the algorithm and one epoch (N iterations) has been computed, each codevector \mathbf{m}_j is updated N_j times where N_j is the number of the training samples assigned to the codevector \mathbf{m}_j during the epoch. Therefore, $\sum_{j=1}^K N_j = N$. Then, following the results of the above section, we can compute the exact value of each codevector at the end of the epoch:

$$\begin{aligned} \mathbf{m}_j[N] &= g_j(\alpha, 0)\mathbf{m}_j[0] + \mathbf{B}_j \quad j=1, \dots, K \\ g_j(\alpha, i) &= (1-\alpha)^{N_{m_j}[i]} (1+\alpha)^{N_j[i]-N_{m_j}[i]} \\ \mathbf{B}_j &= \sum_{i=1}^{N_j} \alpha \left(1(\text{cl}(\mathbf{x}_j[i]) = \text{cl}(\mathbf{m}_j)) - 1(\text{cl}(\mathbf{x}_j[i]) \neq \text{cl}(\mathbf{m}_j)) \right) g_j(\alpha, i) \mathbf{x}_j[i] \end{aligned} \quad (15)$$

where the sequence of training samples $\{\mathbf{x}_j[i], i=1, \dots, N_j\}$ is filled in the order in which LVQ1 assigns the training patterns $\mathbf{x}[n]$ to the codevector $\mathbf{m}_j[n]$. E.g. if LVQ1 assigns $\mathbf{x}[0]$, $\mathbf{x}[5]$ and $\mathbf{x}[15]$ to $\mathbf{m}_j[0]$, $\mathbf{m}_j[5]$ and $\mathbf{m}_j[15]$ respectively during the epoch, then the sequence $\{\mathbf{x}_j[i]\}$ is $\{\mathbf{x}_j[1]=\mathbf{x}[0], \mathbf{x}_j[2]=\mathbf{x}[5], \mathbf{x}_j[3]=\mathbf{x}[15]\}$. Besides $N_{m_j}[i]$ is the number of training samples that have been assigned on-line during the first N iterations of the algorithm to codevector \mathbf{m}_j after i^{th} position that belong to the same class than \mathbf{m}_j , $N_j[i]=N_j-i$ and $N_{m_j}[0]=N_{m_j}$ is the number of the training samples assigned to codevector \mathbf{m}_j during the epoch that belong to the same class than \mathbf{m}_j .

If α is small, the weighted function $g_j(\alpha, 0)$ can be approximated by $1-S_j\alpha$ (where $S_j = 2N_{m_j} - N_j$). To ensure that this approximation is valid, the step size α must be bounded in the following way

$$\begin{aligned} \text{i) } \alpha &\ll U_1^{\text{conv}}(j) \text{ where } U_1^{\text{conv}}(j) = 2/|S_j - N_j/S_j| \quad \text{for } S_j^2 \neq N_j \\ \text{ii) } \alpha &\ll U_2^{\text{conv}}(j) \text{ where } U_2^{\text{conv}}(j) = 6|g'_j(0,0)|/|g''_j(0,0)| \quad \text{for } S_j^2 = N_j \end{aligned} \quad (16)$$

Then equation (15) gives

$$\mathbf{m}_j[N] = (1-S_j\alpha)\mathbf{m}_j[0] + \mathbf{B}_j \quad j=1, \dots, K \quad (17)$$

In the following epoch, the algorithm departs from $\{\mathbf{m}_j[N], j=1,\dots,K\}$ and applies again the whole training set over its update equation since data sampling is cyclic. Thus, $\mathbf{m}_j[2N] = (1 - S_j \alpha) \mathbf{m}_j[N] + \mathbf{B}_j$, $j=1,\dots,K$. Although now S_j and \mathbf{B}_j can differ from the previous values since their value depend on the assignation of data to codevectors that can vary from epoch to epoch. We could repeat the same reasoning to derive the value of codevectors when LVQ1 has passed through training data a fixed number of epochs. In consequence, LVQ1 for $K > 1$ with $\alpha \ll U^{\text{conv}}(j)$ is a *line search method* (Dennis & Schnabel, 1989) of the following form:

$$\begin{aligned} \mathbf{m}_j[(c+1)N] &= \mathbf{m}_j[cN] + \alpha \mathbf{d}[c] = \mathbf{m}_j[cN] + (\mathbf{B}_j - S_j \alpha \mathbf{m}_j[cN]) = \\ &= \mathbf{m}_j[cN] + \alpha \sum_{i=1}^{N_j} (1(\text{cl}(\mathbf{x}_j[i]) = \text{cl}(\mathbf{m}_j)) - 1(\text{cl}(\mathbf{x}_j[i]) \neq \text{cl}(\mathbf{m}_j))) (\mathbf{g}_j(\alpha, i) \mathbf{x}_j[i] - \mathbf{m}_j[cN]) \end{aligned} \quad (18)$$

$j = 1, \dots, K, c \geq 0$

where the sequence $\{\mathbf{x}_j[i], i=1\dots N_j\}$ is defined as above. According to (Dennis & Schnabel, 1989)§4.2, LVQ1 as a line search method will be globally convergent if it satisfies three simple conditions:

$$\begin{aligned} \text{i) } & E_{\text{LVQ1}}(\mathbf{C}[(c+1)N]) < E_{\text{LVQ1}}(\mathbf{C}[cN]) + \gamma \alpha \mathbf{d}[c]^T \nabla_{\mathbf{C}} E_{\text{LVQ1}}(\mathbf{C}[cN]) \\ \text{ii) } & \mathbf{d}[c]^T \nabla_{\mathbf{C}} E_{\text{LVQ1}}(\mathbf{C}[(c+1)N]) \geq \beta \mathbf{d}[c]^T \nabla_{\mathbf{C}} E_{\text{LVQ1}}(\mathbf{C}[cN]) \\ \text{iii) } & \mathbf{d}[c]^T \nabla_{\mathbf{C}} E_{\text{LVQ1}}(\mathbf{C}[cN]) < -\sigma \|\mathbf{d}[c]\| \|\nabla_{\mathbf{C}} E_{\text{LVQ1}}(\mathbf{C}[cN])\| \end{aligned} \quad (19)$$

where T denotes transpose, $0 < \gamma < \beta < 1$ and $\sigma > 0$. The first condition ensures that the decrease in E_{LVQ1} is enough in relation to the step $\alpha \mathbf{d}[c]$. The second is that the step is not too small. Finally, the third condition simply says that the step direction must be a descent direction. If this three conditions are fulfilled then either $\nabla_{\mathbf{C}} E_{\text{LVQ1}}(\mathbf{C}[k]) = 0$ for some k or $\lim_{k \rightarrow \infty} \nabla_{\mathbf{C}} E_{\text{LVQ1}}(\mathbf{C}[k]) = 0$.

However there is no need of checking these three conditions at all points $\mathbf{C}[(c+1)N]$ and $\mathbf{C}[cN]$ for $c > 0$ due to the structure of the codebook space in which the optimization algorithm searches a solution. Each value of \mathbf{C} implements a possible dichotomy of training data. Furthermore, the codebook space is divided in a finite number of regions in which the NN classifier implements one of the possible dichotomies, that is regions where the classification accuracy has a fixed value (see examples on section 5). Hence, the dynamics of any line

search algorithm is characterised by a search within regions that implement a fixed dichotomy and a (finite) number of transitions between these regions.

If LVQ1 is inside one of these regions during an epoch then data assignation is not changed. Then equation (18) when $g(\alpha, i) \rightarrow 1$ (e.g. $\alpha < 2/\kappa N_{m_j}$) coincides with the update equation of a batch gradient descent algorithm over equation (1) since on-line assignation of training samples to codevectors coincides with batch assignation. Hence, LVQ1 will fulfil the three conditions of equation (19) if it simply could converge for the fixed dichotomy of the region. This link can be established since the above section shows that if α is bounded enough then LVQ1 can converge in the presence of a minimum of the cost function so this implies that LVQ1 then follows accurately the gradient path of the cost function. Since the above section works with the case in which a codevector always sees the same training data, we can apply here the results of table 2 where $U = \min_{j=1\dots K} U_j^{\text{conv}}/\kappa$ and U_j^{conv} is computed using the equation (16). Accordingly, the sufficient condition to ensure a good behaviour of LVQ1 inside regions with fixed dichotomies is $\alpha < U$ and $\alpha < \min_{j=1\dots K} 2/S_j$.

Nevertheless, equation (18) does not agree with the batch update equation if LVQ1 goes from one region of constant classification accuracy to another during an epoch. Then we might check if LVQ1 accomplishes these three conditions, although it would be enough to ensure that LVQ1 finally will transit from the origin region to another at the end of the epoch (e.g. condition iii was met). However if LVQ1 follows the gradient path inside regions of fixed dichotomies, it will be difficult to find situations in which transitions between these regions cause problems. Therefore, if LVQ1 works well inside regions of fixed dichotomy, the algorithm will be globally convergent from a practical point of view.

According to the above analysis, if α is effectively controlled, that is $\alpha < \min_{j=1\dots K} U_j^{\text{conv}}/\kappa$ and $\alpha < 2/S_j$ for all epochs, we will observe two different phases in the dynamics:

- 1) A phase of formation of the Voronoi Regions in which LVQ1 moves the codebook in different regions of constant classification accuracy and
- 2) A phase of fine adjustment of the Voronoi regions in which LVQ1 converges to an attractor near a local minimum of E_{LVQ1} .

Once LVQ1 reach an attraction basin, a fixed dichotomy is implemented. Therefore, each sequence $\{\mathbf{x}_j[i], i=1, \dots, N_j\}$ will be N_j periodic (see example 2), where N_j is a fixed number of training samples that are assigned to codevector \mathbf{m}_j in the attraction basin. Since then there are K independent non-linear dynamical systems with a fixed and periodic sequence $\{\mathbf{x}_j[i]\}$,

we can compute the K fixed points $\{\mathbf{m}_j[\infty], j=1,\dots,K\}$ using the results of section 3.1. If LVQ1 follows the gradient path ($\alpha < \min_{j=1,\dots,K} U_j^{\text{conv}}/\kappa$ and $\alpha < 2/S_j$) and the optimization error is small ($\alpha < \min_{j=1,\dots,K} 2/\kappa N_{m_j}$) then

$$\mathbf{m}_j[\infty] = \frac{1}{2N_{m_j} - N_j} \sum_{i=1}^{N_j} (1(\text{cl}(\mathbf{x}_j[i]) = \text{cl}(\mathbf{m}_j)) - 1(\text{cl}(\mathbf{x}_j[i]) \neq \text{cl}(\mathbf{m}_j))) (1 - \alpha(N_{m_j}[i] - N_j + i)) \mathbf{x}_j[i] \quad j = 1, \dots, K \quad (20)$$

where N_{m_j} is the number of training points that fall in R_j belonging to the same class than \mathbf{m}_j and $N_{m_j}[i]$ is the number of training samples in the sequence $\{\mathbf{x}_j[u]\}$ after $\mathbf{x}_j[i]$ that belong to the same class than \mathbf{m}_j . Again, the attractor of LVQ1 (equation 19) is near a minimum point of E_{LVQ1} . Thus, LVQ1 has an optimization error that depends on the position of data in memory. Although this error will be arbitrary small only when alpha was decreased to zero, we find in practice that it is small enough for $\kappa=10$ (see section 5).

4. The Batch LVQ1 algorithm

As we have just seen, LVQ1 is sensitive to the position of data in memory. This section proposes a batch version (BLVQ1) to cancel this dependence. Besides, it makes use of Newton optimization over E_{LVQ1} . BLVQ1 is defined as

$$\mathbf{C}[n+1] = [\mathbf{m}_1^T[n+1] \dots \mathbf{m}_K^T[n+1]]^T = \mathbf{C}[n] - \mathbf{H}[n]^{-1} \frac{\partial E_{LVQ1}[n]}{\partial \mathbf{C}[n]}; \quad (21)$$

where \mathbf{H} is defined as in section 2.1. Hence, the learning equation is

$$\mathbf{m}_j[n+1] = \frac{1}{2N_{m_j}[n] - N_j[n]} \sum_{i=1}^N 1(\mathbf{x}_i \in R_j[n]) (1(\text{cl}(\mathbf{x}_i) = \text{cl}(\mathbf{m}_j)) - 1(\text{cl}(\mathbf{x}_i) \neq \text{cl}(\mathbf{m}_j))) \mathbf{x}_i \quad j = 1, \dots, K \quad (22)$$

where N_{m_j} and N_j are defined as before. Note that \mathbf{H} can be negative when $2N_{m_j} < N_j$. In these cases, the use of this algorithm must be avoided (see section 6). Another problem arises when $N_{m_j}[n] = N_j[n]$. Then we can retain the last value: $\mathbf{m}_j[n+1] = \mathbf{m}_j[n]$. If this strategy is insufficient and BLVQ1 sticks at these points, we might restart the algorithm. On the other hand, the convergence of the algorithm is very fast due to the Newton effect (Hestenes, 1980).

5. The Generalised LVQ1 algorithm

LVQ1 performs a clustering process over a probability density function that is zero at Bayes borders for a two-class problem (see e.g. LaVigna, 1990). Consequently, the resulting nearest-neighbour classifier estimates Bayes classifier. However, what happens for problems with a greater number of classes? Regions where the density of points of the majority class is smaller than the sum of the other densities cause a maximum of the cost function E_{LVQ1} since there $2N_{mi} < N_i$ for all $i=1\dots K$. Hence, LVQ1 gets away from these (repelling) points. However, Bayes borders are in these regions of negative density (from the LVQ1's point of view) and consequently a correct placement of prototypes could be impossible.

A simple solution to the problem of negative density leads to the so-called **generalised LVQ1 (GLVQ1) algorithm** that is presented in on-line and batch versions. In the on-line version, a scaling factor is simply introduced in the amount of change of the winning prototype for the case in which its label does not agree with the input pattern's label:

$$\mathbf{m}_i[n+1] = \begin{cases} \mathbf{m}_i[n] + \alpha[n](\mathbf{x}[n+1] - \mathbf{m}_i[n]) & \text{if } \mathbf{x}[n+1] \in R_i[n] \text{ and } \mathbf{x}[n+1], \mathbf{m}_i \in \text{same class} \\ \mathbf{m}_i[n] - \lambda\alpha[n](\mathbf{x}[n+1] - \mathbf{m}_i[n]) & \text{if } \mathbf{x}[n+1] \in R_i[n] \text{ and } \mathbf{x}[n+1], \mathbf{m}_i \notin \text{same class} \end{cases} \quad i = 1, \dots, K \quad (23)$$

where $\lambda > 0$. The batch version has the following update equation:

$$\mathbf{m}_i[n+1] = \frac{1}{(1+\lambda)N_{mi}[n] - \lambda N_i[n]} \sum_{j=1}^N \mathbf{1}(\mathbf{x}_j \in R_i[n]) (1 - \lambda \mathbf{1}(\text{cl}(\mathbf{x}_j) \neq \text{cl}(\mathbf{m}_i))) \mathbf{x}_j \quad i = 1, \dots, K \quad (24)$$

where N_{mi} and N_i are defined as before. Note that GLVQ1 gives the K-means algorithm performed separately for each class when $\lambda=0$ and gives LVQ1 and batch LVQ1 (BLVQ1) when $\lambda=1$. In fact, GLVQ1 minimises the following cost function:

$$E_{GLVQ1}(\mathbf{C}, \lambda) = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^K \mathbf{1}(\mathbf{x}_i \in R_j) \mathbf{1}(\text{cl}(\mathbf{x}_i) = \text{cl}(\mathbf{m}_j)) \|\mathbf{x}_i - \mathbf{m}_j\|^2 - \frac{\lambda}{2} \sum_{i=1}^N \sum_{j=1}^K \mathbf{1}(\mathbf{x}_i \in R_j) \mathbf{1}(\text{cl}(\mathbf{x}_i) \neq \text{cl}(\mathbf{m}_j)) \|\mathbf{x}_i - \mathbf{m}_j\|^2 \quad (25)$$

where the $\text{cl}(\mathbf{x})$ function returns the class label of \mathbf{x} and $\mathbf{1}(\text{condition})$ is the indicator function. On-line GLVQ1 performs the *pattern-based version* of gradient descent over equation 25 and batch GLVQ1 employs Newton optimization. While on-line version is globally convergent (for λ belonging to certain interval), batch GLVQ1 is locally convergent and must be avoided when the algorithm is not

near a minimum point. Besides, the algorithm must be restarted when the dividing term of equation 24 is zero or takes a negative value.

For a large set of prototypes, It is easy to show that GLVQ1 can perform a clustering process over a density function that could be zero at Bayes borders so the resulting nearest-neighbour classifier could estimate Bayes. However, an effective clustering process can be performed when training sets are very large and this is not the common situation in real-world problems.

Hence, for small data sets, the equilibrium points of GLVQ1 can serve as an indication of how classification accuracy could be improved in comparison with LVQ1. The minimization of E_{GLVQ1} (equation 25) ensures a training error $\leq 100/(1+\lambda)\%$ since the minimum points accomplish that $N_{mi} > \lambda(N_i - N_{mi})$ for all $i=1, \dots, K$ and consequently the number of correct classification $\sum_{i=1}^K N_{mi} > (\lambda/1 + \lambda)N$. In this way, we could push the learning system towards a minimum of the training classification error for very large values of λ ($\lambda \gg 1$). However the cost function E_{GLVQ1} has more chances of being ill conditioned as λ augments since the number of feasible solutions (minimum points E_{GLVQ1}) is reduced as λ is increased. Moreover, numerical instability of the learning algorithm may appear for large values of λ .

6. Case Studies

This section includes several examples, which allows us to illustrate several aspects about LVQ1 and BLVQ1.

6.1. Example 1: Attractors and repulsors of E_{LVQ1} .

We have 2 classes (A and B), 2 examples \mathbf{a} and \mathbf{b} (one in each class) and 2 prototypes \mathbf{w}_A and \mathbf{w}_B (one for each class). $E_{LVQ1}(\mathbf{w}_A, \mathbf{w}_B)$ has two zeros on points $\mathbf{p}_1 = (\mathbf{w}_A = \mathbf{a}, \mathbf{w}_B = \mathbf{b})$, $\mathbf{p}_2 = (\mathbf{w}_A = \mathbf{b}, \mathbf{w}_B = \mathbf{a})$. According to section 2.1, the first zero (classification accuracy=100%) is a minimum and an attractor of LVQ1 since $\mathbf{H}(\mathbf{p}_1) = \mathbf{I}$. The second zero (accuracy=0%) is a maximum and a repulsor of LVQ1 due to $\mathbf{H}(\mathbf{p}_2) = -\mathbf{I}$. If we use BLVQ1 with a initial value of the prototypes $\mathbf{p}[0]$ that produce wrong classification, $\mathbf{H}(\mathbf{p}[0]) = -\mathbf{I}$ and the algorithm then stops at the maximum \mathbf{p}_2 . Figures 2a and 2b show the normalised E_{LVQ1} function ($2E_{LVQ1}/N$) and its contour levels for $a=2$ and $b=6$. We denote $F = (w_A + w_B)/2$ as the frontier point between classes. BLVQ1 only converges to \mathbf{p}_1 when $2 < F < 6$ ($w_A < w_B$). Instead, LVQ1 only does not converge to \mathbf{p}_1 when $w_B \leq a$ and $w_A \geq b$.

6.2. Example 2: Dynamics of LVQ1.

We have again two classes (A and B), 2 prototypes w_A and w_B (one for each class) and the following data: -.6(B), -.6(B), -.6(B), -.5(A), 1(A), 1.5(A), 2(A), 2.5(A), 3(B), 4(A), 6(B).

E_{LVQ1} (fig. 3) is discontinuous for all F that coincides with each sample and when $w_A=w_B$. These points make the dynamics of a gradient descent algorithm to change. There are 20 possible dichotomies but only four \mathbf{C} with $\nabla E_{LVQ1}(\mathbf{C})=0$ implement those dichotomies that induce them. Two out of these four codebooks are minimum points: $\mathbf{p}_1=(0.6, -0.65)$, $\mathbf{p}_2=(4.6, 6)$. They can be computed using equation (2) with the dichotomy $-0.5<F<1$ ($w_A>w_B$) and $4<F<6$ ($w_A<w_B$) respectively. Both points make $E_{LVQ1}<0$ since the distance between codevectors and data of the same class is much smaller than the distance between codevectors and data that belong to other classes. The minimum classification error is $2/11$ but \mathbf{p}_1 and \mathbf{p}_2 only gives $3/11$ and $4/11$ respectively. Besides $E_{LVQ1}(\mathbf{p}_2)<E_{LVQ1}(\mathbf{p}_1)$. Hence the best solution for LVQ1 is \mathbf{p}_2 . Unlike example 1, the minimum points of E_{LVQ1} do not minimise the classification error.

In figure 4 we can see the dynamics of LVQ1 with constant alpha and *cyclic sampling* for several values of α (0.5,0.2, 0.02 and 0.002) and different initial values. We can observe that LVQ1 follows the gradient path as α is decreased. When α is small enough, LVQ1 then converges near \mathbf{p}_1 or \mathbf{p}_2 depending on the initial value. Table 4 shows, according to section 3, the upper bounds of α for w_A and w_B (α_A and α_B) that ensure 1) the convergence of the algorithm in the attraction basin and 2) small optimization error (that is, a convergence near \mathbf{p}_1 or \mathbf{p}_2). These bounds help us to understand better the resulting dynamics. Since codevector has a different bound of α and we use only one global α , we must ensure that the step size was below the smallest bound. Figures 4a, 4b, 4f show the convergence to \mathbf{p}_1 . The step size α must be smaller than $0.061<0.141$ to ensure the convergence. Hence, LVQ1 only converges well when $\alpha=0.02$ and 0.002 . LVQ1 also converges when $\alpha=0.2$ in figure 4a but the optimization error is large since $0.2>0.03$. However, LVQ1 diverges when it is initialized near \mathbf{p}_1 for $\alpha=0.2$ (figure 4b). We notice that in all cases in which LVQ1 converges to \mathbf{p}_1 , the optimization error of w_A is bigger than w_B 's due to the bound to ensure a small optimization error is much smaller in the case of w_A . The same happens with the optimization error of \mathbf{p}_2 (figures 4c, 4d, 4e) since the upper bound of w_A for a small error is smaller than w_B 's. Unlike the previous case, alpha=0.2 and 0.5 can also ensure the convergence to \mathbf{p}_2 because of $0.2<0.6$. This fact allows the convergence to the attractor \mathbf{p}_2 when LVQ1 might converge to \mathbf{p}_1 (figures f and b).

In figure 5 we can see the dynamics of LVQ1 with constant alpha and *random sampling* for $\alpha=0.02, 0.002$ and the same initial values as before (we do not show the dynamics for $\alpha=0.5$ and 0.2 since LVQ1 violently diverges). As expected, the dynamics for the random

case are much noisier. However, if LVQ1 gets to an attraction basin, the noise is then extremely high until LVQ1 finally converges. This noise in the dynamics causes that the random version of LVQ1 converges much slower than cyclic version. Besides, the optimization error is greater since the equilibrium points are farther from the minimum points of the cost function. Finally, we can observe that if we start to run random LVQ1 near a fixed point (figures 5b and 5f), the algorithm converges to it. E.g. Figure 5f shows the convergence of random LVQ near a maximum of the cost function.

p1			p2		
Convergence		Small optimization error	Convergence		Small optimization error
Exponential attractor	Slow attractor	$\alpha_A < 0.03$ $\alpha_B < 0.2$	Exponential attractor	Slow attractor	$\alpha_A < 0.02$ $\alpha_B < 0.066$
$\alpha_A < 0.061$ $\alpha_B < 0.141$	-		$\alpha_A < 0.33$ $\alpha_B < 0.5$	$0.33 \leq \alpha_A < 0.6$ $0.5 \leq \alpha_B < 1$	

Table 4. Upper bounds of α for w_A and w_B for ensuring the convergence of LVQ1 and a small optimization error for $\kappa=10$.

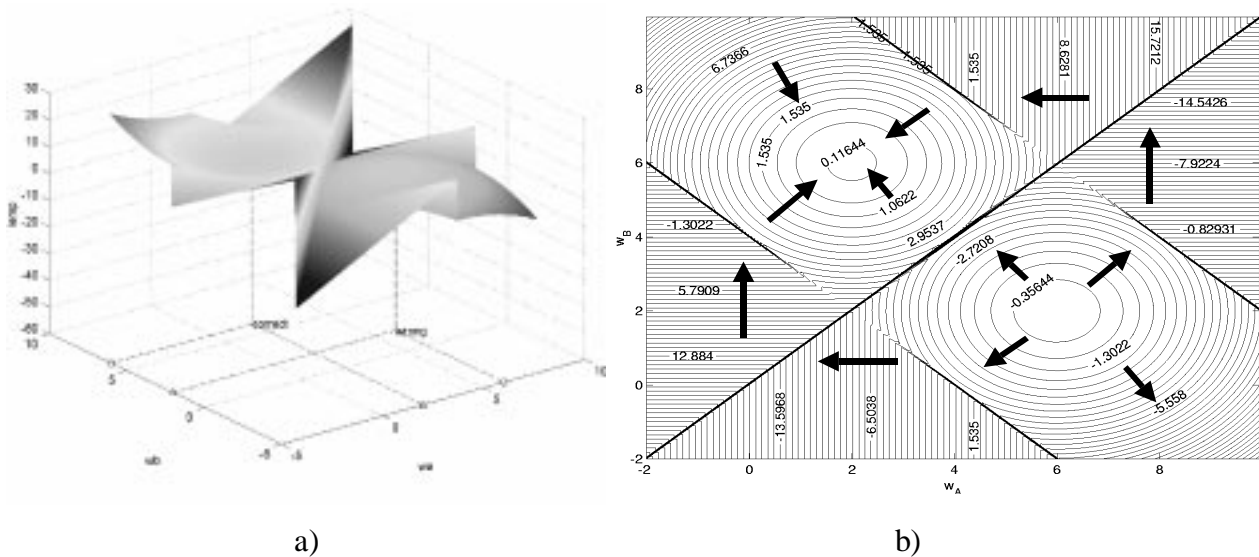


Fig.2. $2E_{LVQ1}/N$ of example 1: a) 3-D display. Symbols Δ and O denote the examples of class A and B respectively. The critical point of the cost function when both training samples coincides with the codevectors of the same class ($w_A=\Delta$, $w_B=O$) corresponds to a minimum. On the other hand, there is a maximum when codevectors are assigned to samples of the other class ($w_B=\Delta$, $w_A=O$); b) Contour levels. Arrows show the trajectory of a gradient descent algorithm

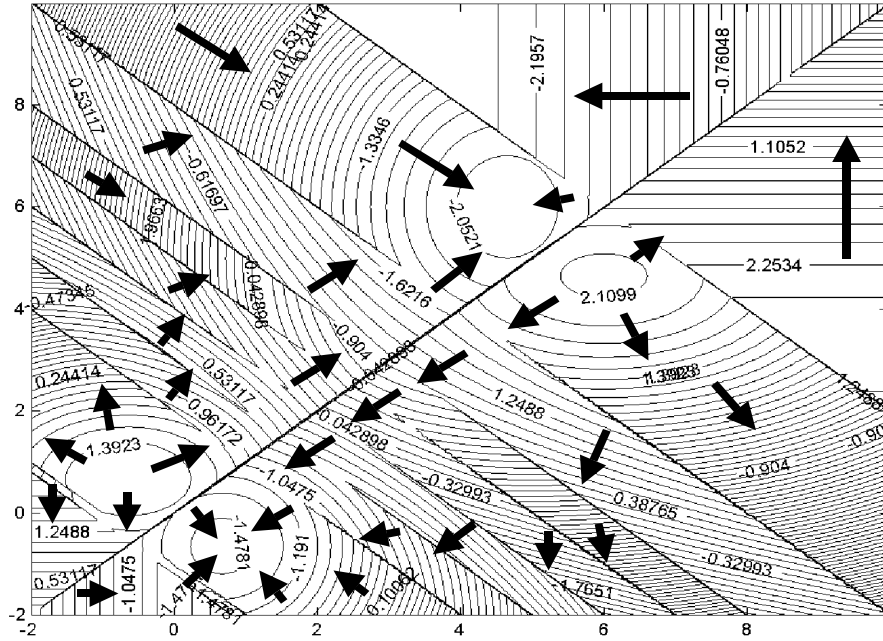
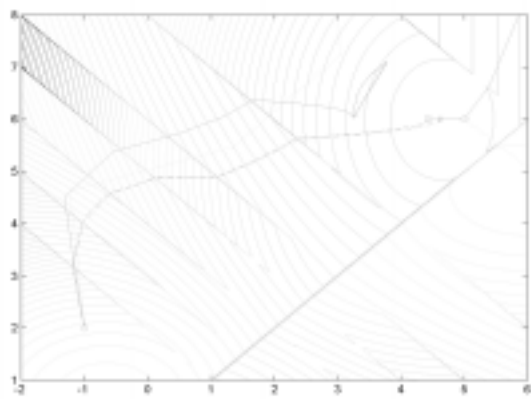


Fig.3. Contour levels of E_{LVQ1}/N of example 2. Arrows show the trajectory of a gradient descent algorithm again.

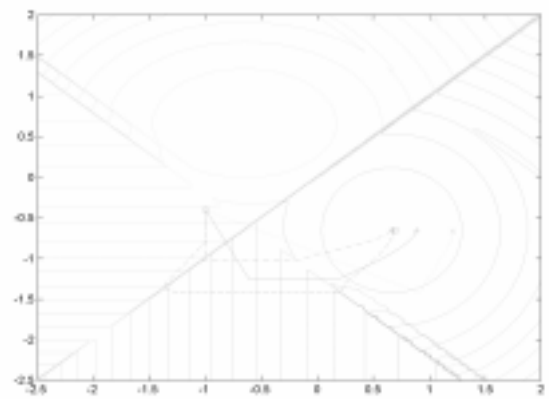
6.3. Example 3: Convergence of LVQ1 near attraction basins.

Suppose two equally probable gaussian classes A and B with the same covariance matrix $\mathbf{K} = \sigma^2 \mathbf{I}$ and centres \mathbf{m}_A and \mathbf{m}_B . For this problem, the optimal set of prototypes of a nearest neighbour classifier is formed by two prototypes $\mathbf{w}_A = \mathbf{m}_A$ and $\mathbf{w}_B = \mathbf{m}_B$. The probability of error of this classifier coincides with Bayes error. For a finite training set, the sub-optimal prototypes are simply the class means. As $N \rightarrow \infty$, these estimators converge to class expectations so the classifier tends to Bayes'. If we use LVQ1 these estimators are not computed since the critical points of E_{LVQ1} does not coincide with class means. However, due to the symmetry of the problem, the LVQ1 prototypes induce a frontier region, which tend to coincide with class means's. Consider this classification problem when $\mathbf{m}_A=2$, $\mathbf{m}_B=7$ and $\sigma=2.5$. Since both classes have the same priors the probability of misclassification is

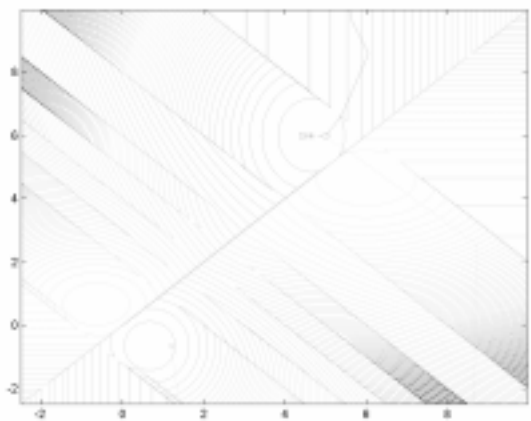
$$P(E) = \frac{1}{2} (P(E|C_A) + P(E|C_B)) = P(E|C_B) = \text{erfc}\left(\frac{m_B - F}{\sigma}\right) = \text{erfc}\left(\frac{7 - 4.5}{2.5}\right) = 15,729921\%$$



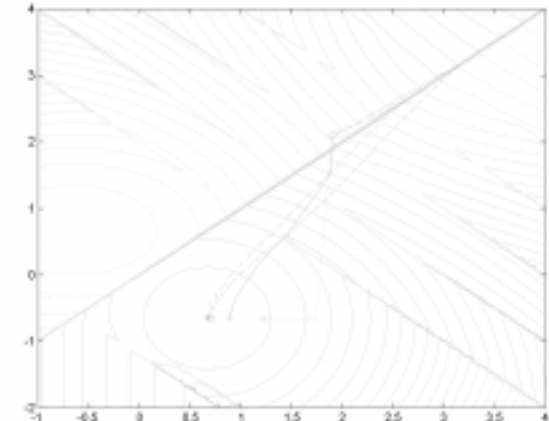
a)



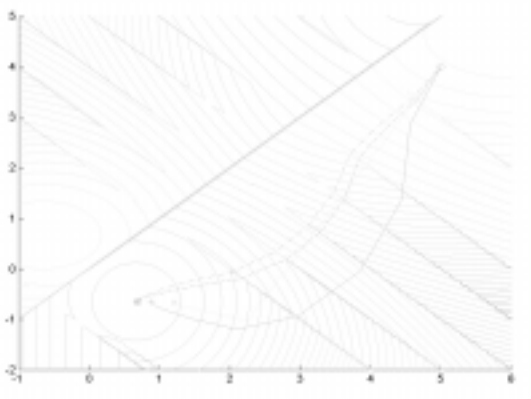
d)



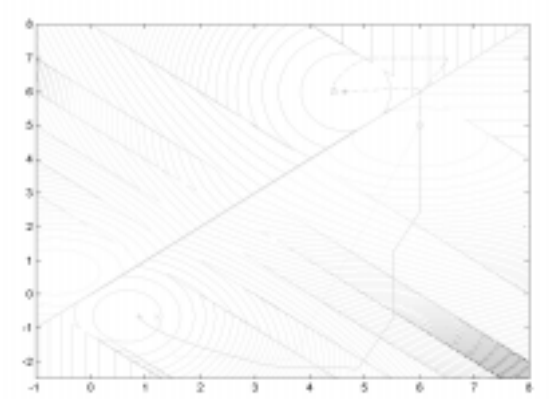
b)



e)

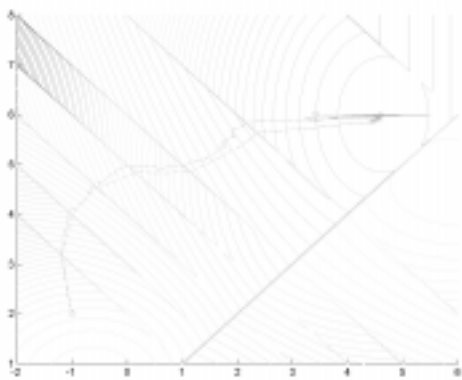


c)

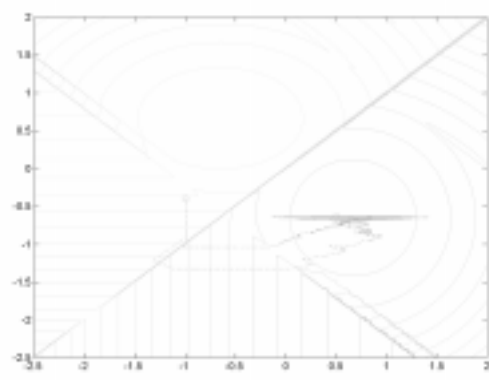


f)

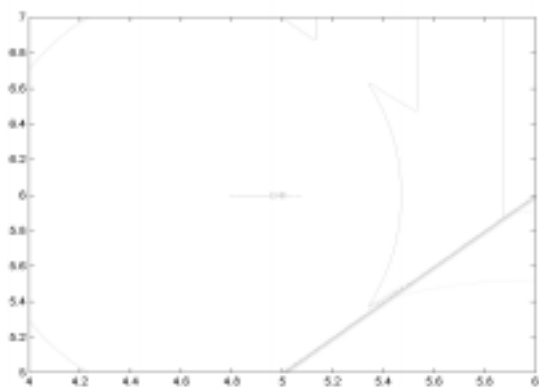
Figure 4. Dynamics of LVQ1 with constant alpha and cyclic sampling in example 2 for different initial conditions. Initial point is marked with O, while the reached attractors for $\alpha=0.5, 0.2, 0.02$ and 0.002 are marked with X, +, and * respectively.



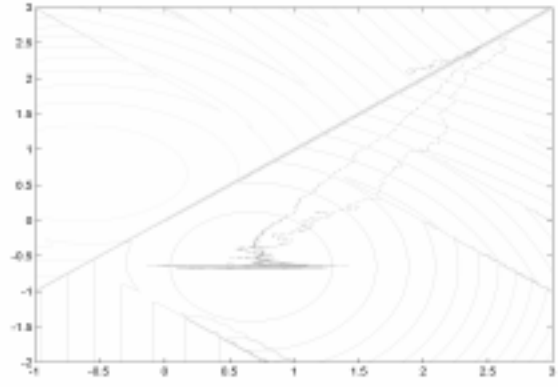
a)



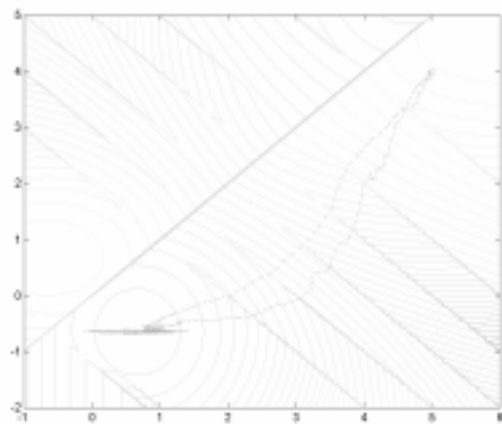
d)



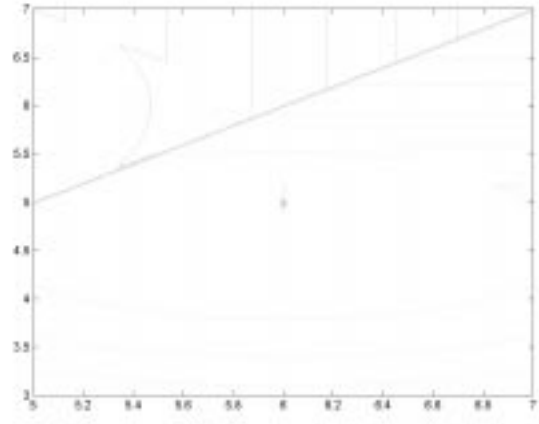
b)



e)



c)



f)

Figure 5. Dynamics of LVQ1 with constant alpha and random sampling in example 2 for different initial conditions. Initial point is marked with O, while the reached attractors for alpha=0.02 and 0.002 are marked with and * respectively.

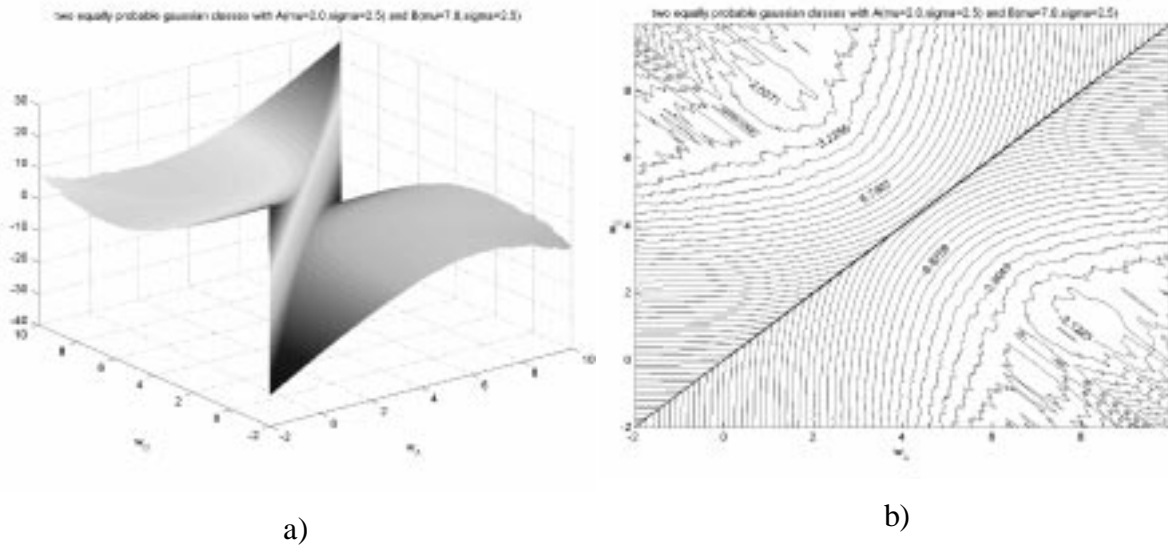


Fig. 6. $2E_{LVQ1}/N$ of example 3: a) 3-D Display; b) Contour levels.

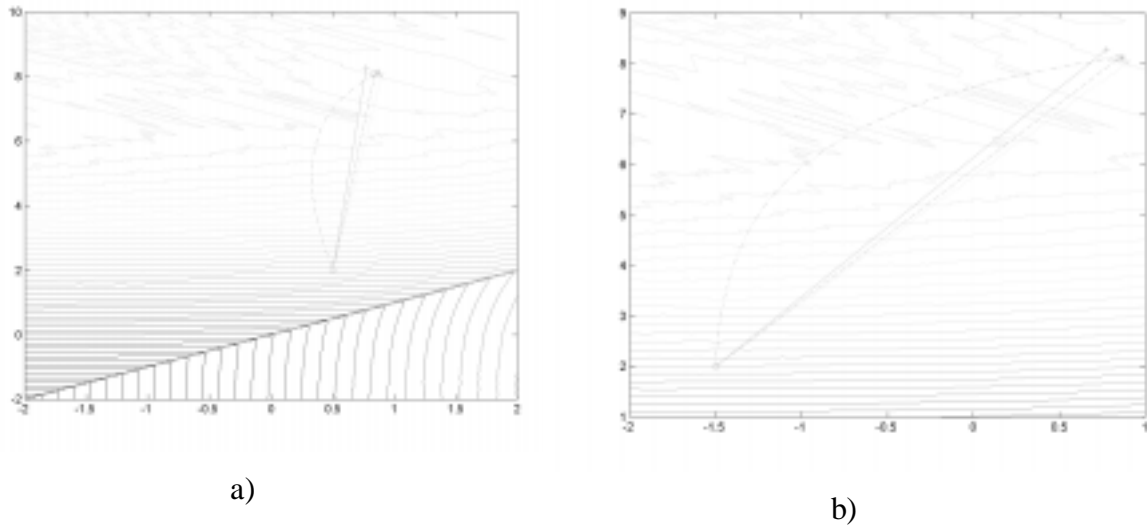


Figure 7. Dynamics of LVQ1 with constant α and cyclic sampling in example 3 for different initial conditions. Initial point is marked with a circle, while the reached attractors for $\alpha=0.5, 0.01, 0.001$ and 0.00001 are marked with X, +, a square and * respectively.

where F is the frontier point $(m_A+m_B)/2$ and erfc is the complementary error function. Let us compute the prototypes w_A and w_B using a training set of 20000 samples (10000 samples/class). As figures 6a and 6b show, the only minimum of E_{LVQ1} for this database is located near the point $(1,8)$ so $F \approx 4.5$. We notice that the dynamics of LVQ1 near the attraction or repulsion basins must be characterised by continuous transitions between regions of constant classification accuracy since training data is very dense there. If we use the class

means (2.002644 and 7.017978) as prototypes, $F=4.51031$ and the resulting NN classifier has the following probability of error

$$P(E) = \frac{1}{2} \left(\operatorname{erfc} \left(\frac{m_B - F}{\sigma} \right) + \operatorname{erfc} \left(\frac{F - m_A}{\sigma} \right) \right) = 15,730627\%$$

Now, we compute w_A and w_B with LVQ1. We use different values of the constant step size α (0.5, 0.01, 0.001 and 0.00001). According to section 3, α must be much smaller than 0.00066 ($2/(4000-1000)$) to ensure that LVQ1 near the attraction basin follows the gradient path. In figure 7, we show the dynamics of LVQ1 with cyclic sampling for two different initial points using one fixed configuration of the training data in memory. As expected, LVQ1 follows the gradient path for $\alpha=0.00001$. Besides, we can observe a striking behaviour of LVQ1 for the other values of the step size: LVQ1 converges very quickly and follows a similar trajectory than a Newton algorithm like BLVQ1. However the attractors of LVQ1 depend on the order of training data in memory. As figure 8 shows, if we use different ordering of training data in memory, LVQ1 converges to different points that form a ball around a center point. As alpha is increased, the radius of this ball is greater. This dependence causes that the variance in the classification accuracy increases when α is greater as the following experiment confirms. We have computed the attractors of LVQ1 for 100 different ordering of the training data in memory when α is 0.01 and 0.001. Then the classification error over the 100 runs was estimated with a test set of 100000 samples. The average error is 15.93 % and 15.90% with a variance of 0.03 and 0.008 for $\alpha=0.01$ and 0.001 respectively.

6.4. Periodic points in E_{LVQ1} .

Consider the following classification problem: -0.5 (A), 1.0 (A), 1.5 (A), 2.0 (A), 2.5 (A), 3.0 (B), 4.0 (A), 6.0 (B), 6.0 (B). We have again two classes (A and B), 2 prototypes w_A and w_B (one for each class). Figure 5 shows the contour levels of the normalised E_{LVQ1} . Once a gradient descent algorithm gets to an attraction basins, the algorithm will oscillate between $4 < F < 6$ with $w_A < w_B$ (the solution with minimum classification error 8/9) and $3 < F < 4$ with $w_A < w_B$ (error=7/9).

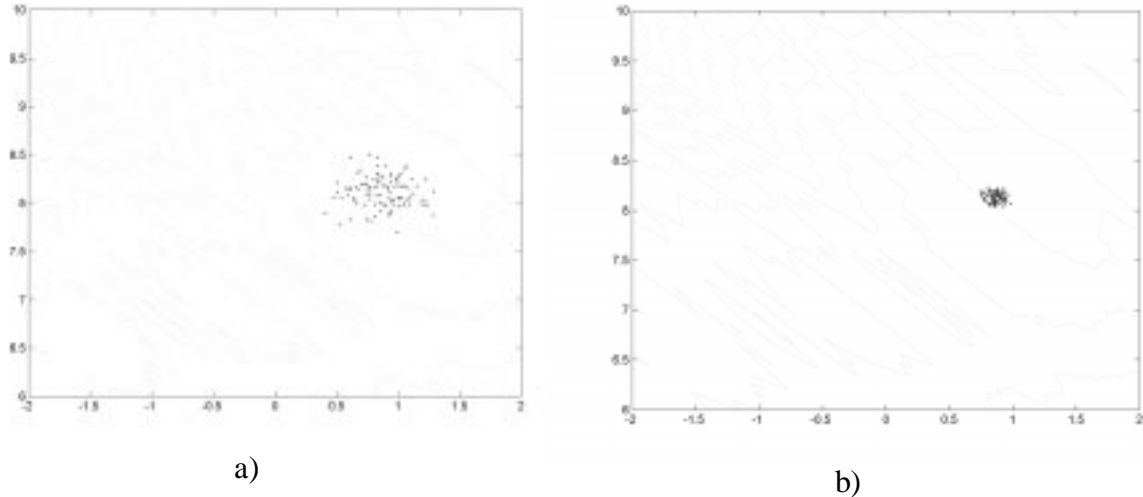


Fig.8. Distribution of attractors of LVQ1 in example 3 when $\alpha=0.01$ (a) and 0.001 (b) for different ordering of training data in memory.

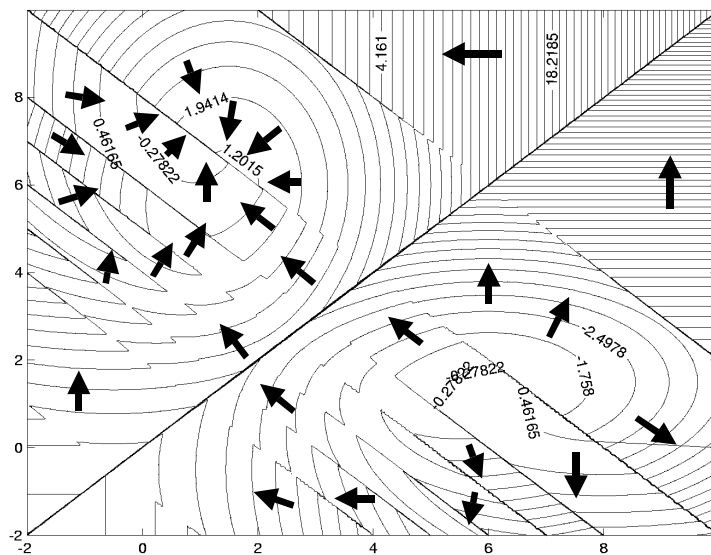


Fig.9. Contour levels of $2E_{LVQ1}/N$ of example 4.

6.5. LVQ1 vs. BLVQ1

We introduce the Finish speech database (Kohonen et al., 1995) that contains 3964 cepstral-coefficient vectors picked up from continuous Finnish speech, from the same speaker. Each vector has a dimension of 20 and has been labelled to represent one of the 20 possible phonemes. We use BLVQ1 and LVQ1 with constant α and cyclic sampling. Codebooks have been initialised with K-means and codevectors labels are computed by a majority rule of training data in each Voronoi region. The optimal values of the step size α and the training time have been estimated using a validation set. Table 3 shows results over 100 runs in a Pentium II 433

MHz. In average, BLVQ1 achieves a slightly better classification rate in 5 out of 7 cases. Besides, BLVQ1 is 0.3-1.5% faster.

	Codebook size	50	100	150	200	300	500
Blvq1	Test error (%)	12.5 (1.065)*	11.7 (0.71)	11.22 (0.58)	11.01 (0.69)	10.80 (0.57)	10.65 (0.53)
	Learning time (sec.)	1.49 (0.50)	2.48 (0.72)	3.14 (0.82)	4.87 (1.33)	6 (0.78)	9.78 (1.64)
Lvq1	Test error (%)	12.23 (0.82)	11.78 (0.78)	11.46 (0.61)	11.15 (0.68)	11.05 (0.53)	10.65 (0.62)
	Learning time (sec.)	3.72 (1.2)	4.74 (1.38)	5.66 (1.56)	6.58 (1.49)	8.44 (1.5)	12.75 (2.25)

Table 3. Experimental results for Finish speech database. [*Mean (variance)]

6.6. GLVQ1 vs. LVQ1

We use again the Finish speech database (Kohonen et al., 1995). Now, we compare GLVQ1 and LVQ1 with constant α and cyclic sampling. Codebooks have been initialised with the eveninit program (Kohonen et al., 1995). The optimal values of the step size α and the training time have been estimated using a validation set. Table 4 shows results over 100 runs. In average, GLVQ1 achieves a slightly better classification rate in all cases for $\lambda > 1$.

λ	Codebook Size			
	200	400	800	1600
1.0	10.939 (0.8221)*	10.828 (0.653)	10.706 (0.556)	10.086 (0.304)
2.0	10.7049 (0.8135)	10.812 (0.7594)	10.55 (0.57)	9.9855 (0.33)
2.5	10.608 (0.7863)	10.773 (0.7145)	10.5361 (0.5938)	9.939 (0.33)
3.5	10.601 (0.7199)	10.7292 (0.66)	10.47 (0.4826)	9.895 (0.375)

Table 4. Experimental results of GLVQ1 for Finish speech database . [*Mean (variance) of the test error]

7. Discussion

7.1 Finite-sample convergence of LVQ1.

As we show in section 3, the cyclic version of the LVQ1 algorithm is a line search method (Dennis & Schnabel, 1989) once the constant step size α is adequately bounded ($\alpha < \min_{j=1..K} U^{\text{conv}}(j)/\kappa$). According to (Dennis & Schnabel, 1989), LVQ1 will be then globally convergent (that is, it converges near a local minimum of E_{LVQ1}) when it follows approximately the gradient path of E_{LVQ1} . LVQ1 as an optimization algorithm will search a

solution in the codebook space. This space is divided in regions where the NN classifier implements one of the possible dichotomies. Hence, the dynamics of LVQ1 is characterised by a search in regions that implement a fixed dichotomy and a number of transitions between these regions. Inside these regions, LVQ1 is equivalent to a batch gradient descent algorithm over E_{LVQ1} . But LVQ1 will follow the gradient path when $\alpha < \min_{j=1,\dots,K} 2/S_j$. (This is in fact the typical condition of gradient algorithms that limits the step size with the largest eigenvalue of the Hessian matrix of the cost function; see (Bishop, 1995) §7.5.1 for further details.) On the other hand, transitions between these regions can cause LVQ1 do not converge to a local minimum. Although equation (19)-iii helps checking that LVQ1 ensure that the LVQ1 tracks the gradient path during transitions, one can expect LVQ1 work well there once it follows the gradient path inside regions of fixed dichotomy.

The conditions for ensuring that LVQ1 is globally convergent make α small and therefore the convergence rate is then slow. However we have shown empirically that, if LVQ1 is near an attraction basin, the convergence rate can be accelerated without affecting the stability of the algorithm when we increase alpha (example 3). This happens since LVQ1 can converge near an attraction basin, or LVQ1 is locally convergent, when $|g_j(\alpha, 0)| < 1$ according to section 3.1. In other words, α meets the necessary condition that enables the convergence once LVQ1 gets to a region of constant classification accuracy. Consequently, the algorithm can converge locally (that is near an attraction basin).

7.2. Cyclic vs. random LVQ1.

Once the constant step size is bounded enough, cyclic LVQ1 is globally (or locally) convergent. However, the attractors of cyclic LVQ1 depend on the order in which training data are stored in memory (eq. 19). These makes cyclic LVQ1 converges around balls near attractors for different combinations of stored data in memory. These balls will have smaller radius as alpha is decreased. This dependence increases the variance of the codevectors and, consequently, the variance of the classification accuracy. This result could confirm that random sampling was usually preferred by researchers using LVQ algorithm since cyclic sampling was expected to make the training process sensitive to the order of supplied samples. However, random LVQ1 is not the cure for the problem. The dynamics of random LVQ1 are really unstable. As we have shown using a simple experiment, random LVQ1 is more sensitive to the step size. Besides it converges much slower and the dynamics are extremely noisy near attraction basins. Furthermore, the optimization errors of random LVQ1

are greater than cyclic LVQ1's for the same values of α . Finally, if the algorithm is initialized near a maximum of the cost function, the algorithm stops. Thus, cyclic LVQ1 seems clearly superior to random LVQ1.

7.3. LVQ1 vs. BLVQ1.

Due to LVQ1 is sensitive to samples sequence, a batch version was proposed to eliminate this dependence. Besides BLVQ1 converges faster due to Newton's effect. In fact, Newton's convergence is superlinear and many times quadratical (Hestenes, 1980). However its deficiencies must be taken into account (Hiriart-Urruty, 1993). It diverges violently if the algorithm is not near a solution. Besides, \mathbf{H} must be positive definite, otherwise its solution tends to approximate a maximum or a saddle point (see example 1). If there is no guarantee of being near a solution, one can start with LVQ1 and then switch to BLVQ1.

7.4. GLVQ1 vs. LVQ1.

GLVQ1 is a straightforward generalisation of LVQ1 since a scaling factor λ is introduced in the negative term of the cost function of LVQ1. GLVQ1 includes K-means and LVQ1 as a special cases gives when $\lambda=0$ and $\lambda=1$ respectively. Moreover, the minimization of E_{GLVQ1} (equation 25) ensures a training error $\leq 100/(1+\lambda)\%$ so we could push the learning system towards a minimum of the training classification error for very large values of λ ($\lambda \gg 1$). However the cost function E_{GLVQ1} has more chances of being ill conditioned as λ augments since the number of feasible solutions (minimum points E_{GLVQ1}) is reduced as λ is increased. Moreover, numerical instability of the learning algorithm may appear for large values of λ .

7.5. Initialisation Procedures.

If $\mathbf{C}[0]$ was near a local minimum of E_{LVQ1} , convergence to spurious attractors in LVQ1 or to saddle or maximum points in BLVQ1 could be avoided. $\mathbf{C}[0]$ might be placed inside Bayesian class borders since the condition of existence of attractors is met there: $2N_{mj} > N_j$ for any Voronoi cell in Bayes regions. The LVQ_PAK's eveninit program (Kohonen et al., 1995) could be used since it places codevectors inside an estimation of Bayes borders. The use of K-means followed by label assignation can also be considered (as in example 3) since each resulting Voronoi cell j also meet the condition $2N_{mj} > N_j$.

7.6. LVQ1-based nearest neighbour (NN) classifier as a data-dependent partitioning classifier.

A data-dependent partitioning method (Devroye et al., 1996) uses the training data \mathbf{D}_N to produce a partition $P_N = \Pi_N(\mathbf{D}_N)$ of the input space according to a prescribed rule Π_N (e.g. a

learning algorithm). Then a classification rule g_N is created based on a majority vote among the labelled training data that falls in the cells of P_N . If LVQ1 (our Π_N) reaches a local minimum of E_{LVQ1} , we can ensure that $2N_{mj} > N_j$ $j=1\dots K$. Then the NN classifier that uses C^* assigns new patterns to one of the existing classes according to a majority rule of D_N in each Voronoi cell. Thus the LVQ1-based NN classifier is a data-dependent partitioning classifier. Therefore, the study of the necessary conditions for the LVQ1-based NN classifier converges to the Bayes classifier as $K, N \rightarrow \infty$ can be analysed with the theory developed in (Devroye et al., 1996). According to (Devroye et al., 1996), the rule $\{g_N\}$ is strongly consistent if the *class of partitions P_N is rich enough* (as the Voronoi partitions are if K grows with N at a right rate) and *the rule is local enough* which is related to Π_N . If Π_N was the K-means algorithm then the resulting classifier would be strongly consistent. Hence, one could use the mathematical tools developed for the K-means case ((Devroye et al., 1996) §21.5) to investigate the conditions that ensure the consistency for LVQ1 since LVQ1 and K-means are in fact related (e.g. if Bayes error is 0, LVQ1 and K-means will converge to the same solution as $K, N \rightarrow \infty$).

8. Conclusion

LVQ1 performs on-line gradient descent over the cost function E_{LVQ1} . The minimum points of E_{LVQ1} ensures that the LVQ1-based NN discriminates according to a majority vote among the labelled training data that fall in each Voronoi cell. Although these points do not guarantee the minimisation of the classification error, the convergence of the classification error to the Bayes error as $K, N \rightarrow \infty$ could be investigated using the mathematical tools developed in (Devroye et al., 1996).

For constant step sizes (α) and cyclic sampling, we have found the necessary conditions for LVQ1 will be globally convergent. The first is that LVQ1 must be a line search algorithm. This is achieved when $\alpha < \min_{j=1,\dots,K} U^{\text{conv}}(j)/K$ for all the epochs. Furthermore, LVQ1 must follow the gradient path. This second condition can be fulfilled when $\alpha < \min_{j=1,\dots,K} 2/S_j$ for all the epochs. These two simple bounds on the step size can help to ensure a good global convergence of LVQ1, although in practice LVQ1 can be locally convergent (that is it can converge near an attraction basin) for bigger values of the step size. If $\alpha < \min_{j=1,\dots,K} 2/\kappa N_{mj}$ for all the epochs, the optimization error of LVQ1 will be small.

However, the attractors of LVQ1 will also depend on the order in which training data are stored in memory. Since random sampling make worse the dependence of attractors on the order of supplied data (and also the dynamics), a batch version of LVQ1 (BLVQ1) was proposed to

cancel this problem. BLVQ1 employs the Newton optimisation method so it converges very fast. Nevertheless, negative Hessian can make it to search e.g. maximum instead of a minimum. Therefore, the use of BLVQ1 must be avoided if the algorithm is not near a minimum. LVQ1 could be used first and we then would switch to BLVQ1. However, BLVQ1 seems to work well alone using a proper initialisation procedure: it achieves a similar classification rate than LVQ1 in the Finnish speech database (Kohonen et al., 1995) with a speedup in the training time (LVQ1 time/BLVQ1 time) that goes from 130% to 250%.

Finally, a straightforward generalization of LVQ1 (GLVQ1) has been presented. GLVQ1 includes K-means and LVQ1 as a special cases gives. The minimization of GLVQ1's cost function ensures a training error $\leq 100/(1+\lambda)\%$ where λ is a parameter determined by the user. Hence, we could push the learning system towards a minimum of the training classification error for very large values of λ ($\lambda \gg 1$). Experimental results show that this simple generalization allows a better classification accuracy of the nearest-neighbour classifier.

References

- Benveniste, A., Métivier, M., & Priouret, P. (1990). Adaptive Algorithms and Stochastic Approximations, Berlin: Springer-Verlag.
- Bishop, C. M. (1995). Neural Networks and Pattern Recognition, Oxford: Oxford University Press.
- Bottou, L. (1998). Online Learning and Stochastic Approximation. David Saal (Ed.). Online Learning and Neural Networks. Cambridge, UK: Cambridge University Press.
- Dennis Jr., J.E. & Schnabel, R.B. (1989). A View of Unconstrained Optimization in Nemhauser, G.L., Rinnooy Kan, A.H.G. & Todd, M.J. (Eds.) Optimization, Amsterdam: North-Holland.
- Devaney, R. L. (1989). An introduction to Chaotic Dynamical Systems, 2nd. Boston, MA: Reading, Addison-Wesley.
- Devroye, L., Györfi, L. & Lugosi, G. (1996). A Probabilistic Theory of Pattern Recognition, Berlin: Springer-Verlag
- Hestenes, M. (1980). Conjugate Direction Methods in Optimization, Berlin, New York: Springer-Verlag.
- Hiriart-Urruty, J.B. & Lemaréchal, C. (1993). Convex Analysis and Minimization Algorithms, Berlin: Springer-Verlag.
- Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., & Torkkola, K. (1995). LVQ_PAK. The Learning Vector Quantization Program Package. Version 3.1, Helsinki: Helsinki University of Technology, Laboratory of Computer and Information Science.
- Kohonen, T. (1996). Self-organizing Maps, 2nd Edition, Berlin: Springer-Verlag.
- Lavigna, A. (1990). Nonparametric classification using learning vector quantization. Ph. D. Dissertation, University of Maryland.
- Zhu, C. et al. (1995). Analysis of LVQ algorithms for Pattern Classification, Proceedings of ICASSP 95, 5, 3471-3474.

Annex: Worst-case analysis for the weighted functions $g(\alpha, i)$

The weighted function $g(\alpha, i)$ (equation (7)) will be able to be approximated by $1 - S[i]\alpha$ when the second order term of the Taylor expansion of $g(\alpha, i)$ is negligible respect to $S[i]\alpha$. (We assume that $S^2[i] \neq N[i]$). Hence,

$$\alpha \ll \min_{i=0..N} \frac{2}{\left| S[i] - \frac{N[i]}{S[i]} \right|} \quad \text{or} \quad \alpha \ll \frac{2}{\max_{i=0..N} \left| S[i] - \frac{N[i]}{S[i]} \right|} \quad (\text{A-1})$$

The upper bound of alpha must be computed for all the possible cases. So we must compute the case in which the sequences $S[i]$ and $N[i]=N-i$ produce the worst situation, that is the maximum of $|S[i] - N[i]/S[i]|$ is the greatest value of the maximums of the other possible sequences $S[i]$ and $N[i]$. This happens when the sequence $S[i]$ has the greatest maximum of all the possible ones. Since,

$$S[i+1] = \begin{cases} S[i]+1 & \text{if } N_m[i+1] = N_m[i] \\ S[i]-1 & \text{if } N_m[i+1] = N_m[i]-1 \end{cases} \quad (\text{A-2})$$

$S[i]$ has the greatest maximum when all the training data that belong to the same class than the codevector are at the end of the array. Then,

$$S[i] = \begin{cases} S[i]+1 & \text{if } i < N - N_m \\ S[i]-1 & \text{if } i \geq N - N_m \end{cases} \quad (\text{A-3})$$

and the maximum value N_m+1 is reached for $i=N-N_m-1$. Consequently,

$$\alpha_{\text{worst-case}} \ll \frac{2}{\left| N_m + 1 - \frac{N - (N - N_m - 1)}{N_m + 1} \right|} = \frac{2}{N_m} \quad (\text{A-4})$$