

LINEAR VECTOR CLASSIFICATION: AN IMPROVEMENT ON LVQ ALGORITHMS TO CREATE CLASSES OF PATTERNS

Michel Verleysen, Philippe Thissen, Jean-Didier Legat

Université Catholique de Louvain
Microelectronics Laboratory
3, pl. du Levant
B-1348 Louvain-la-Neuve
Belgium

Learning Vector Quantization algorithms (LVQ1 and LVQ2), proposed by Kohonen, are widely used for the quantization and the classification of vectors into clusters. These algorithms quantize each class of vectors in the space into a defined number of 'prototypes'. Despite an efficient quantization of the stimuli space, these algorithms are not well adapted to classification tasks where the distribution of prototypes inside a single class is not important, provided that the boundaries between classes are adequately approximated through the prototypes. We propose here an adaptation of the LVQ1 algorithm where the resulting prototypes will approximate the boundaries between classes; by this way, stimuli located as well near the border as in the center of a class will be correctly classified, even if they are not adequately quantified in the sense of 'Vector Quantization'.

1. Introduction

Learning Vector Quantization algorithms [1], are used in data processing to quantize and to classify a K-dimensional input space represented by N vectors (or 'stimuli'), into a fixed number P of prototypes, approximating the probability density of the N stimuli, and dissociating the different classes to which the stimuli (and the prototypes) belong. LVQ are iterative algorithms where all stimuli are sequentially (or randomly) presented; the basic principle is to raise the number of prototypes of the right class in regions where the stimuli density is important, and to move away prototypes leading to misclassification. The difference between the LVQ1 and LVQ2 algorithms resides in the criterion to satisfy in order to move prototypes, and in the way of how the final position of the boundaries between classes are determined.

2. Learning Vector Quantization

2.1. LVQ1

The LVQ1 algorithm can be described as follows.

Before the first iteration, P K-dimensional prototypes $p(i)$ ($i=1...P$) are randomly initialized. If a priori limits of the set of input vectors in the space are known, the prototypes will be chosen inside these limits, in order to accelerate the convergence of the algorithm. One possibility is to initialize the prototypes to any P of the N input vectors. At each iteration, a K-dimensional input vector $x(j)$ ($j=1...N$) is compared to all P prototypes. We then select the prototype $p(a)$ for which the standard Euclidean distance between $p(a)$ and $x(j)$ is minimum

$$\text{dist}(p(a), x(j)) \leq \text{dist}(p(k), x(j)) \quad \forall k \in \{1...P\} \setminus \{a\} \quad (1)$$

($1 \leq a \leq P$). If vectors $x(j)$ and $p(a)$ belong to the same class, $p(a)$ is moved in the direction of $x(j)$ following equation (2).

$$p(a) = p(a) + \alpha (x(j) - p(a)) \quad (2)$$

where α is the adaptation factor ($0 < \alpha < 1$). If the two vectors belong to different classes, $p(a)$ is moved in the opposite direction of $x(j)$:

$$p(a) = p(a) - \alpha (x(j) - p(a)) \quad (3)$$

The adaptation factor α must decrease with time to obtain a good convergence of the algorithm [2]. Usually, the same value of the adaptation factor α is kept for a whole 'epoch' (an epoch consists in presenting once the whole set of input patterns), and is decreased before the next one. Figures 1a) and 1b) respectively show the distribution of prototypes for an uniform two-class distribution (prototypes are marked '+' and 'x' for the two classes, and input vectors are marked '.'), and for a single class with 2-nd order decreasing (in the two directions) probability density of stimuli.

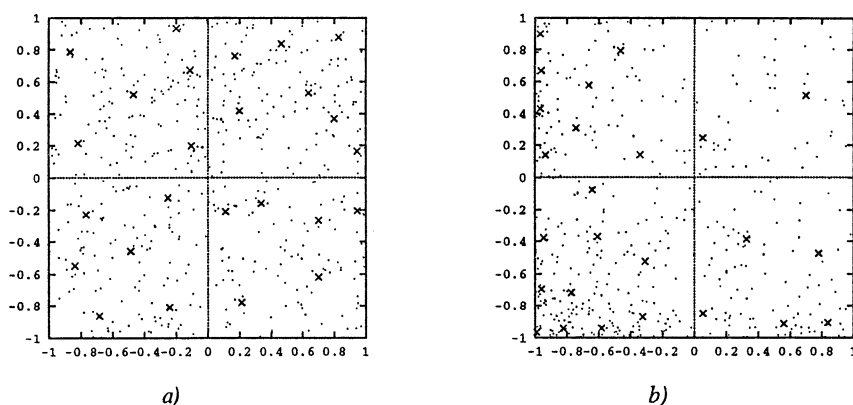


fig.1 examples of LVQ convergence on 2-D vectors (100 iterations)

2.2. LVQ2

LVQ2 is an extension of the LVQ1 rule, where several conditions must be verified before moving prototypes [3]. In this rule, the two prototypes $p(a)$ and $p(b)$ closest to input vector $x(j)$ are selected according to equation (1.1). The three conditions to satisfy in LVQ2 algorithm are then:

- stimulus $x(j)$ and nearest prototype $p(a)$ must belong to different classes
- stimulus $x(j)$ and second nearest prototype $p(b)$ must belong to the same class
- stimulus $x(j)$ must be inside a predefined symmetrical window centered on the midpoint of segment $p(a)-p(b)$

If these conditions are satisfied, prototypes $p(a)$ and $p(b)$ are moved according to:

$$p(a) = p(a) - \alpha (x(j) - p(a)) \quad (4)$$

$$p(b) = p(b) + \alpha (x(j) - p(b)) \quad (5)$$

The three mentioned conditions for moving the prototypes guarantee a better approximation of the Bayes decision boundary between prototypes $p(a)$ and $p(b)$; a justification of this can be found in [1]. This algorithm suffers from a huge amount of computations necessary to reach a quasi-equilibrium point.

3. Learning Vector Classification (LVC)

The above described LVQ algorithms are efficient to quantize the space into a defined number of regions. The influence regions defined by the prototypes are the result of a "Voronoi tessellation": the influence region of one defined prototype is the set of all points in the input space which are closer from this prototype than from all others. If the probability density of input vectors is constant, the influence regions will have similar volumes. On the other hand, if input vectors are not equiprobable, the volumes of the influence regions will be modulated in order to have the same probability to be selected at each presentation of a stimulus.

Nevertheless, if the distribution of prototypes in the whole input space is adequate for vector quantization, it would be more efficient for vector classification to move all prototypes near the boundaries between classes instead to have most of them inside the classes. Indeed, if the vectors near the boundaries between classes are correctly classified, the influence regions of the prototypes near these boundaries will automatically extend inside the classes, leading to a correct classification of vectors further from the limits of the class; this is the result of the Voronoi tessellation, where the regions can spectacularly extend if no other prototypes are in the vicinity.

3.1. LVC algorithm

We propose here an algorithm which will converge on a solution where all prototypes are situated in the neighborhood of the boundaries between classes. The principle of this algorithm is simple: it consists in selecting the stimuli, or input vectors, near the boundaries, and in applying the LVQ1 algorithm to these vectors only. By this way, the prototypes will represent the distribution of these selected vectors, and will thus be concentrated near the limits of the classes.

Of course, it is impossible to make a choice between the stimuli, and to select only those situated near the boundaries between classes, if these boundaries themselves are not known (which is the initial situation of the problem). The proposed algorithm is thus based on an iterative process: at each iteration, an approximation of the boundaries is computed with the prototypes resulting from the previous iteration. With this approximation, the stimuli situated within a defined distance D of the boundaries are used as input of a standard LVQ1 algorithm; this will converge on a solution where the prototypes better approximate the boundaries than before the current iteration, and the process can be repeated, decreasing the D parameter at each step.

To reduce the amount of computations, a stimulus x will be deemed to be within a distance D of the boundaries between classes if the Euclidean distance between x and the closest prototype from a different class is less than D . The region where stimuli are considered is thus approximated by the union of all hyperspheres centered on the prototypes and with radii equal to D , instead of being an hypersurface situated at a distance D of the boundary determined by the Voronoi tessellation. Since only an approximation of the limits between classes is needed at each iteration, this way to estimate the boundaries is valid and will not decrease the performances of the algorithm.

When decreasing parameter D between two iterations, it may happen that some prototypes fall outside the region where input patterns are considered, i.e. the above-mentioned estimated volume (union of hyperspheres centered on prototypes from other classes). In this case, only the prototypes located inside the region are kept for the next iteration. To keep the same number of prototypes for each iteration, the remaining ones are again randomly chosen in the list of input vectors, as at the beginning of the LVQ algorithm, but of course inside the region where stimuli will be considered at the next iteration. All prototypes could also be randomly chosen inside this region at each iteration; keeping the results of the previous step however turned out to be efficient for a reduced amount of computations.

More precisely, with the same notations as in section 2, the algorithm can be described as follows:

1. Initialize P prototypes to any of the N input vectors (stimuli). For example, choose

$$p(i) = x(i) \quad (1 \leq i \leq P) \quad (6)$$

2. Choose initial distance $D=D_{\text{init}}$ (input vectors are only considered within a distance D of any prototype from another class). Since it is important to consider all prototypes during the first iteration, distance D_{init} can be chosen as the maximum distance between patterns in the input distribution.

If N_{steps} is the number of epochs (see section 2.1), the following is repeated N_{steps} times:

3. Execute LVQ1 algorithm on the stimuli inside the considered region.
4. Decrease parameter D linearly:

$$D = D - \frac{D_{final} - D_{init}}{N_{steps}} \quad (7)$$

where D_{final} is the value of D after completion of the algorithm (D_{final} depends on the precision required on the boundaries).

5. Eliminate prototypes falling outside the region determined by the actual set of prototypes and the new distance D .
6. Replace missing prototypes by stimuli falling inside the region.

Step 3, the LVQ1 algorithm executed at each iteration, is in the reality not critical; keeping the prototypes found at the previous iteration, as explained above, allows in the reality to execute only one epoch of the LVQ1 algorithm (one presentation of the set of input patterns) at each cycle. By this way, the execution of the LVQ1 algorithm and the diminution of the regions where stimuli are considered are simultaneously performed, and the total number of cycles is similar to what would be necessary in a standard LVQ1 process; verifying if the stimuli and/or prototypes fall inside the regions determined by the estimated boundaries and by distance D leads however to supplementary computations.

3.2. Simulation results

The LVQ1 and LVC algorithms have been compared on the same sets of data, and the computations have been carried out during the same number of epochs (see last remark of section 3.1). Figures 2 a) and 2 b) respectively show the results of the LVQ1 and LVC algorithms after 20 epochs on uniform distributions. 1000 stimuli were considered, and 50 prototypes; stimuli above the sine curve are deemed to belong to class 1, and those below the curve to class 2.

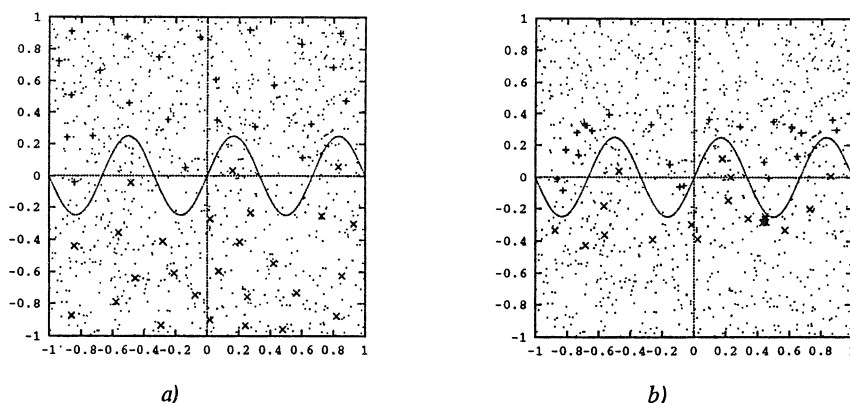


fig.2 LVQ1 and LVC algorithms on uniform distributions with 50 prototypes

Figure 2b) shows a sensible amelioration in the classification of input vectors. Indeed, it can be seen in figure 2a) that less than 20 prototypes are used to determine the approximation of the sine curve; the other prototypes are further from the boundary, and do not contribute to this approximation. In other words, if the Voronoi tessellation of the prototypes in figure 2a) is drawn, and if only the lines separating different classes are kept, more than 30 prototypes are useless. On the other hand, in figure 2b), almost all prototypes contribute to the boundary; as a result, a better approximation of the sine curve is achieved.

The two algorithms have been tested for the classification of the 1000 input vectors; 28 misclassifications came out of algorithm LVQ1, while only 17 were found with the LVC. This is however a weak result,

since the approximation of the boundary between classes was already quite good with the LVQ1 algorithm; it can indeed be intuitively viewed that less than 20 prototypes are necessary to obtain such a pretty good approximation, which leaves the remaining prototypes free to scan the whole input space.

However, if the number of prototypes is not sufficient to correctly approximate the boundaries between classes in the case of the LVQ1 algorithm, i.e. when too much of them are used to scan the input space and do not contribute to the useful part of the Voronoï tessellation, the LVC algorithm can greatly improve the performances of the classification, with regards to the LVQ1 algorithm. The example of figures 3a) and 3b) shows respectively the LVQ1 and LVC algorithms applied to the same input distribution as in figure 2, but now with only 10 prototypes instead of 50.

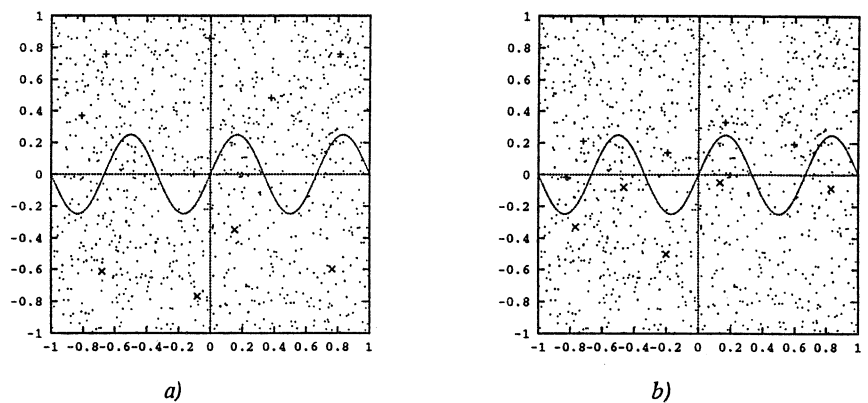


fig.3 LVQ1 and LVC algorithms on uniform distributions with 10 prototypes

Figure 3a) clearly shows that 10 prototypes are not sufficient to approximate adequately the boundary between the classes when the LVQ1 algorithm is used; on the contrary, in figure 3b) shows that most prototypes are brought near the boundary with the LVC algorithm, leading to a much better approximation of the sine function. The 1000 input vectors tested with the LVQ1 and LVC algorithms respectively lead to 97 and 26 misclassifications; such an amelioration (about 7%) is appreciable if we consider that in most applications the important patterns to classify are situated near the boundaries.

The two above examples show increased percentages of correct classification when using the LVC algorithm. In the reality, the performances increase can be much more than 7% (in the example of figure 3), if the input data are spread over very large volumes (or surfaces in our examples). We considered indeed only problems where the size of the boundary region is almost of the same order of magnitude as the size of the whole volume comprising all input vectors (in the examples of figures 2 and 3, the boundary region is roughly 40% of the total space). In other problems however, the boundary region can be much smaller (with respect to the whole volume), leading to bad classification performances if the LVQ1 algorithm is used. The amelioration brought by the LVC algorithm is then much appreciable.

Let us mention that it is possible to approximate more closely the boundaries by reducing D_{final} . However, if this parameter is chosen too small, it may happen that all input vectors near the boundaries are no more used during the last iterations, leading to a concentration of prototypes in small regions, and thus to a strong decrease of the performances. One way to avoid this phenomenon is to compute the percentage of correct classifications at each iteration, and to stop when this percentage decreases too firmly. With this method, the algorithm becomes independent from parameter D_{final} . This does not increase the number of computations, since all input patterns are already compared with all prototypes at each iteration.

A last example uses overlapping distributions of input patterns. In figure 4a) and 4b), if y is the y -coordinate of the vectors, the class of the stimuli is given by equation (8).

Again, maximizing the number of prototypes in the boundary regions leads to increased percentages of correct classifications: with 25 prototypes and 1000 input vectors, the number of misclassifications is respectively 67 and 45 for the LVQ1 and LVC algorithms.

$$\begin{cases} \text{class} = 1 & \text{if } y \geq 0.2 \\ p(\text{class} = 1) = (y + 0.2) / 0.4 & \text{if } 0.2 \geq y \geq -0.2 \\ \text{class} = 2 & \text{if } -0.2 \geq y \end{cases} \quad (8)$$

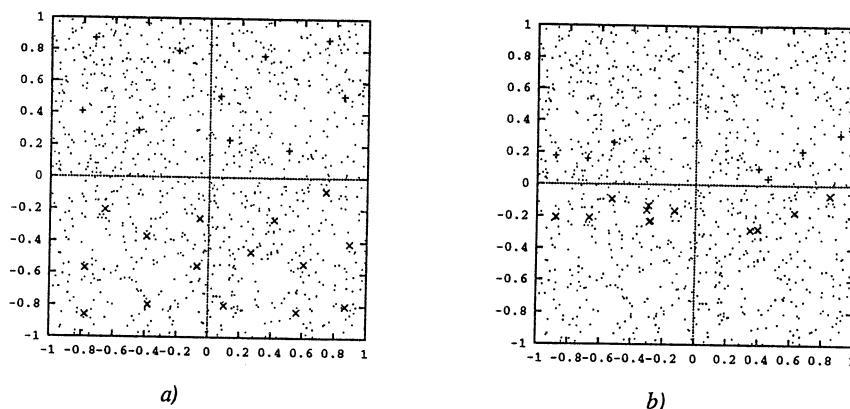


fig.4 LVQ1 and LVC algorithms on overlapping distributions (see text)

4. Conclusion

We described here an algorithm which can be used for the classification of patterns. It is based on Kohonen's LVQ1 algorithm, but has the particularity to bring all prototypes near the boundaries between classes, leading to a better approximation of these limits, and thus to increased classification performances. The prototypes found with the LVC algorithm however no more quantify adequately the input space; applications where such quantization is important should then avoid the use of this method.

We showed on several examples the well-founded of this algorithm. The increase of performances, with respect to the standard LVQ1 algorithm, is particularly sensible when the volume comprising all input vectors is very large with respect to the regions around boundaries between classes.

The way of how prototypes are generated at each iteration, while efficient, is however somewhat arbitrary. Other methods to bring the prototypes near the boundaries, like moving too distant ones in the direction of the nearest prototype from another class, could be considered. Such possibility is currently being evaluated.

5. References

- [1] Kohonen, T. (1988), "Self-organization and associative memory", 2nd edition, Springer-Verlag, Berlin.
- [2] Jutten, C., Guerin, A., Nguyen Thi, H.L. (1991), "Adaptive optimization of neural algorithms", in: A. Prieto ed., Artificial Neural Networks, Springer-Verlag Lecture Notes in Computer Sciences n°540, Berlin.
- [3] McDermott, E., Katagiri, S. (1991), "LVQ-based shift-tolerant phoneme recognition", IEEE Transactions on Signal Processing, vol.39, n.6, June 1991, pp.1398-1411.

Acknowledgments

Part of this work has been funded by ESPRIT-BRA project n°6891, ELENA-Nerves II, supported by the Commission of the European Communities (DG XIII).