IMPROVED VERSIONS OF LEARNING VECTOR QUANTIZATION

Teuvo Kohonen

Helsinki University of Technology
Laboratory of Computer and Information Science
Rakentajanaukio 2 C, SF-02150 Espoo, Finland

Abstract. The Vector Quantization (VQ) methods are closely related to certain paradigms of self-organizing Neural Networks, and among adaptive VQ methods those based on supervised learning have yielded very high pattern recognition accuracies. The present paper introduces a new variant of the (supervised) Learning Vector Quantization, and discusses practical problems associated with the application of the algorithms. A comparative study of several methods applied to Speech Recognition is included.

1. Introduction

Categorization of signal patterns is one of the most usual Neural Network (NN) applications. It has been demonstrated with artificial as well as natural data [1-5] that the Learning Vector Quantization methods [4-7] constitute a very viable alternative to the more traditional approaches; their classification accuracy is at least as high as that of any other NN algorithm, while due to the very simple computations thereby applied, their speed in learning as well as in classification can be significantly higher. Moreover they are very easy to use.

While the classification accuracy of the LVQ algorithms has been demonstrated to be very close to the decision-theoretic Bayes limit even in difficult cases, nonetheless some minor problems have remained; one of them is optimal initialization of the codebook vectors. The present work aims at providing some helpful answers to such questions and other details. Some new developments are introduced here for the first time; they perform slightly better than the earlier versions.

2. Vector Quantization for the Description of the Probability Density Function

Several results (cf. [8]) exist about Vector Quantization (VQ) for the approximation of continuous functions. Consider a Euclidean space R^n in which a stochastic input variable $\boldsymbol{x} \in R^n$ has the probability density function $p(\boldsymbol{x})$. If a number of *codebook* or *reference vectors* $\boldsymbol{m}_i$, $i = 1,2,...,K$ are placed into R^n, and $\boldsymbol{x}$ is approximated by the closest one denoted $\boldsymbol{m}_c$,

$$\|\boldsymbol{x}-\boldsymbol{m}_c\| = \min_i \{\|\boldsymbol{x}-\boldsymbol{m}_i\|\} \quad , \tag{1}$$

where any norm may be used, then the average discretization error with respect to arbitrary power r is defined by the functional

$$E = \int \|\boldsymbol{x}-\boldsymbol{m}_{c(x)}\|^r \, p(\boldsymbol{x}) \, dV_x \tag{2}$$

where dV_x is a volume element in the $\boldsymbol{x}$ space. With $r = 2$, E defines the mean-square error. It has been shown in the scalar case by Max [8], and for vectors by Gersho [9] and Zador [10] that for the optimal placement of the $\boldsymbol{m}_i$ (when E is minimized) the point density of the $\boldsymbol{m}_i$ approximates the function

$[p(x)]^{n/(n+r)}$. In other words, for $n >> r$ as the case usually is in practice, *the optimal VQ approximates* $p(x)$.

3. Vector Quantization for Classification: a Counterexample

It is not at all obvious how the codebook vectors in Vector Quantization ought to be placed, if this method is directly used for *classification*, i.e., when the codebook vectors are assigned to different categories, and the one that is closest to the input vector defines the classification of the latter. Consider the artificially constructed example delineated in Fig. 1 where the probability density function of class C_1 is uniform in an n-dimensional hypercube and zero outside it, whereas the density function of class C_2 may have *any* form as long as its value inside the hybercube is smaller than that of C_1. Based on known results of the Bayes decision theory (cf., e.g., [7]) it is obvious that the number of misclassifications is minimized and the optimal decision surface defined with only one codebook vector located in the middle of C_2, surrounded by $2n$ codebook vectors of C_1. Two conclusions can thus be drawn: 1. Sometimes even one codebook vector per class may suffice to define the optimal border. 2. The optimal number and placement of the codebook vectors seems to have no direct correspondence to the density functions of each class taken separately.

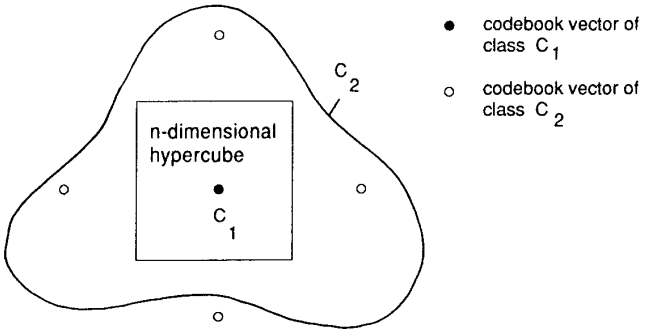

Fig. 1. Exemplification of optimal classification using minimal number of codebook vectors.

4. The First Version of Learning Vector Quantization (LVQ1)

Assume tentatively that the number of codebook vectors in each class and their initial approximate values have been selected by some "adequate" method (cf. Sec. 5), and a placement that is supposed to minimize the classification errors has to be determined. The first iterative algorithm LVQ1 corrects the $\boldsymbol{m}_i(t)$, $t = 0,1,2,...$ as follows: if $\boldsymbol{x} = \boldsymbol{x}(t)$ is a random input sample belonging to some known class, and the codebook vectors are similarly assigned to a number of different classes, then Eq. (1) is always used to define the closest codebook vector $\boldsymbol{m}_c$. After that, the $\boldsymbol{m}_i = \boldsymbol{m}_i(t)$ are updated according to the rule:

$$\boldsymbol{m}_c(t+1) = \boldsymbol{m}_c(t) + \alpha(t)[\boldsymbol{x}(t)-\boldsymbol{m}_c(t)]$$
if $\boldsymbol{x}$ and the closest codebook vector belong to the same class ,

$$\boldsymbol{m}_c(t+1) = \boldsymbol{m}_c(t) - \alpha(t)[\boldsymbol{x}(t)-\boldsymbol{m}_c(t)]$$
if $\boldsymbol{x}$ and the closest codebook vector belong to different classes ,

$$\boldsymbol{m}_i(t+1) = \boldsymbol{m}_i(t) \quad \text{for } i \neq c \,, \tag{3}$$

where $0 < \alpha(t) < 1$, and α is decreasing monotonically with time (e.g., linearly, starting from a small value like 0.01 or 0.02).

This algorithm tends to pull the codebook vectors away from the class borders, leaving a "depletion layer" approximately in those zones where the Bayesian decision borders are. This can be deduced in the the following way. If the steepest-descent gradient-step optimization method is applied to minimize E on the basis of successive stochastic samples $\boldsymbol{x}(t)$, and we consider $r = 2$, then the corrections read

$$\boldsymbol{m}_c(t+1) = \boldsymbol{m}_c(t) + \alpha(t)[\boldsymbol{x}(t) - \boldsymbol{m}_c(t)] \ ,$$

$$\boldsymbol{m}_i(t+1) = \boldsymbol{m}_i(t) \quad \text{for } i \neq c. \tag{4}$$

The $p(\boldsymbol{x})$ in Eq. (2) defines what samples $\boldsymbol{x}(t)$ and with what statistical frequency there occur in the sequence of Eq. (4). The minus sign in the second equation (3) may then be interpreted *to define corrections in the same direction as if Eq. (4) were used for the class to which* $\boldsymbol{m}_c$ *was assigned, but the probability density function of the samples of the neighboring (overlapping) class would be subtracted from that into which* $\boldsymbol{m}_c$ *belongs*. The difference of the probability density functions of the neighboring classes, by definition, falls to zero at the Bayes border, leaving the above "depletion layer" of the codebook vectors.

5. "Adequate" Initialization of the Codebook Vectors

If we would know the optimal number of codebook vectors to be assigned to each class, we might initialize the vectors of each class into values close to the class means. These numbers, however, are not obvious, as stated in Sec. 3. A problem also arises if the class distributions are disconnected, eventually even intermingled. At least it may be said that in general the optimal numbers are *not* proportional to the a priori probabilities of the classes.

One might be able to devise decision rules to create and/or to delete codebook vectors during the learning process, based on the global rate of errors. As such a mechanism might be complex and slow, we first recommend the following simpler means.

Some researchers have used the K-means clustering [8] to define the number and location of the initial codebook vectors. The placement of all codebook vectors is first determined without taking their classification into account, and then *calibrating* the codebook vectors by input vectors with known classification; i.e., a codebook vector is labeled according to the majority of classes of those calibration vectors to which it was closest.This choice, although rather good in the case of smooth and continuous distributions, does not guarantee any stable or unique distribution in difficult cases.

The Self-Organizing Map [7] has an additional advantage of placing the codebook vectors along an "elastic surface" in the input space, whereby it may be expected that their distribution would be more stable. After that, a similar calibration is performed. In most of our speech recognition experiments we used this type of initialization, whereafter "unreliable" codebook vectors were eliminated [2].

Recent experiments [12] have shown that if the topology of the Self-Organizing Map is defined dynamically along the Minimum Spanning Tree (MST) of the codebook vectors, the latter tend to quickly and stably find the form of the overall probability density function of input, however complex it is.

6. The LVQ2

This algorithm has already been used by many researchers. While in the LVQ1 only one codebook vector was updated at a time, this algorithm updates *two* vectors at each step, namely, the "winner" and the "runner-up". The purpose is to shift the midplane of these two vectors directly to the zone where the Bayes border is supposed to lie. An algorithm that can easily be seen to work in that direction is the

following. First define a "window" (cf. below and Fig. 2) around the midplane of neighboring codebook vectors $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$. Let $\boldsymbol{m}_i$ belong to class C_i and $\boldsymbol{m}_j$ to class C_j, respectively. The corrections are defined by

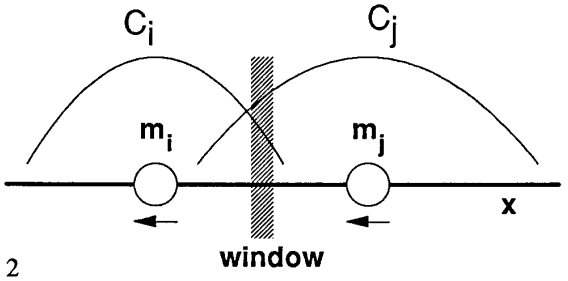

Fig. 2

$$\boldsymbol{m}_i(t+1) = \boldsymbol{m}_i(t) - \alpha(t)[\boldsymbol{x}(t)-\boldsymbol{m}_i(t)] \quad ,$$

$$\boldsymbol{m}_j(t+1) = \boldsymbol{m}_j(t) + \alpha(t)[\boldsymbol{x}(t)-\boldsymbol{m}_j(t)] \quad ,$$

if $\boldsymbol{m}_i$ is the nearest codebook vector to $\boldsymbol{x}$ and $\boldsymbol{m}_j$ the next-to-nearest one, respectively, and $\boldsymbol{x}$ belongs to C_j but not C_i; furthermore $\boldsymbol{x}$ must fall into the "window" (5)

One question usually concerns the optimal definition of the "window". If we are working in a high-dimensional signal space, it seems reasonable to define the "window" in terms of relative distances d_i and d_j measured from $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$, respectively. In this way the borders of the "window" are Apollonian hyperspheres. The vector $\boldsymbol{x}$ is defined to lie in the "window" if

$$\min (d_i/d_j \, , \, d_j/d_i) > 1 - \varepsilon \tag{6}$$

where the size of ε depends on the number of available training samples. Typically we have had ε around 0.35, roughly corresponding to a 25 per cent window. If the window is made too narrow, the training result will suffer from the low statistical accuracy due to the reduced number of corrections.

7. Elimination of instabilities from the LVQ2

It has been pointed out by some of our colleagues that although the corrections of $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$ are roughly in the same direction, the correction on $\boldsymbol{m}_j$ (correct class) has a larger magnitude than that on $\boldsymbol{m}_i$ (wrong class), thus resulting in monotonically decreasing distances $\|\boldsymbol{m}_i-\boldsymbol{m}_j\|$. Although this effect is not as important with high dimensionality as when, e.g., $n = 1$ or 2, nonetheless it would be an improvement if the corrections on $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$ had the same magnitude. This can be effected, e.g., by normalization of the corrections on the right-hand sides of Eq. (5).

8. The LVQ2.1

Since normalization of the corrections in Eq. (5) would make the learning computations significantly heavier and slower, an alternative is to compensate for the above effect, approximately at least, by other means.

The original assumption about making the corrections dependent on *errors* only seems unnecessary, as already indicated by the LVQ1 experiments. We can accept *all the training vectors falling into the "window"*, and the only condition is that *one* of $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$ must belong to the *correct* class, while *the other* to the *incorrect class*, respectively. This algorithm, properly dubbed "LVQ2.1" would then read

$$\boldsymbol{m}_i(t+1) = \boldsymbol{m}_i(t) - \alpha(t)[\boldsymbol{x}(t)-\boldsymbol{m}_i(t)] \, ,$$

$$\boldsymbol{m}_j(t+1) = \boldsymbol{m}_j(t) + \alpha(t)[\boldsymbol{x}(t)-\boldsymbol{m}_j(t)]$$

where $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$ are the two closest codebook vectors to $\boldsymbol{x}$, and the classes of $\boldsymbol{x}$ and $\boldsymbol{m}_j$ agree, whereas those of $\boldsymbol{x}$ and $\boldsymbol{m}_i$ disagree, respectively; furthermore $\boldsymbol{x}$ must fall into the "window". (7)

Notice that $\|m_i - m_j\|$ can now both increase and decrease with approximately equal probability. At the same time more than twice as many corrections are made as in the LVQ2, which increases statistical accuracy in learning.

9. Taking More "Runners-up" into Account

The decision border is determined by those codebook vectors that are adjacent to each other over the decision border. It then occurs quite commonly that an input vector has several neighboring codebook vectors that are almost equidistant from it. Therefore, in order to still increase the statistical accuracy of learning, the corrections might not only be restricted to the "winner" and the first "runner-up", but taking the second etc. "runners-up" also into account. An additional advantage from this is that whereas the updated "runners-up" are often also nearest neighbors in the correct class, their simultaneous correction tends to place them more smoothly.

10. Experiments with Speech Data

In a typical NN Speech Recognition experiment, the input pattern vectors are rather high dimensional and they may be formed, say, as a concatenation of several adjacent speech spectra (e.g., seven spectra from a 70-ms time window centered at the phoneme to be recognized [1,2]).As the purpose of the present study was *relative* comparison of algorithms, we only used *single, 15-frequency-channel spectra* of phonemes (from Finnish speech). The task was very much the same as in our real-time microprocessor-based "Phonetic Typewriter" [14]. As a matter of fact, that device was used to collect the data for this experiment. The number of statistically independent spectral vectors applied for training vs. testing (1500 + 1500) also corresponded to a typical task for which the LVQ was used, namely, quick enrollment of a new speaker after having collected about 150 to 200 words of his dictation. In Test 1, the first of the sets was used for training and the second for testing, respectively; in Test 2 the order was reversed. By and large the experimental setup corresponded to that used in our earlier reports [4,5]. The number of codebook vectors was 117. (We must emphasize, however, that for the practical design of speech recognizers, we avail about $2 \cdot 10^9$ bytes of digital speech data in our files.)

For comparison, Table 1 also contains the recognition accuracies [4,5] for the same data using the Parametric Bayes Classifier (with assumed multivariate normal distributions), the kNN classification results (identical with k = 5 and 6 neighbors), and earlier results relating to the LVQ1 and LVQ2 classifiers, respectively. To these, the LVQ2.1 classifier experiments using one vs. two "runners-up" to the "winner" have now been added.

Table 1 **Speech recognition experiments. Error percentages for isolated phonemes.**

	Parametric Bayes Classifier	kNN Classifier	LVQ1	LVQ2	LVQ2.1 one runner-up	LVQ2.1 two runners-up
Test 1	12.1	12.0	10.2	9.8	9.3	9.3
Test 2	13.8	12.1	13.2	12.0	11.5	11.0

It can be observed that the acceptance of training vectors from the whole "window" in LVQ2.1 (and not only when classification was wrong), as well as inclusion of the second "runner-up" had a positive effect on accuracy; and although no radical improvements were discernible, these amendments do neither essentially increase the computing load. Moreover, especially in low-dimensional problems, LVQ2.1 has been found to be more stable and is thus preferable to LVQ2.

11. Conclusions

The LVQ algorithms are working explicitly in the input domain of the primary observation vectors, and their purpose is to approximate the theoretical Bayes decision borders using piecewise linear decision surfaces. This is done by purported optimal placement of the class codebook vectors in the signal space. As the classification decision is based on the nearest-neighbor selection among the codebook vectors, its computation is very fast. It has turned out that the differences between the presented methods in regard to the remaining discretization error are not significant, and thus the choice of the algorithm may be based on secondary arguments, such as stability in learning, in which respect the new version LVQ2.1 seems to be superior to the others.

References

[1] E. McDermott and S. Katagiri, "Shift-Invariant, Multi-Category Phoneme Recognition Using Kohonen's LVQ2," *Proc. Int. Conf. Acoust., Speech, Signal Processing*, Glasgow, U.K., 1989, pp. 81-84.

[2] H. Iwamida, S. Katagiri, E. McDermott, and Y. Tohkura, "A Hybrid Speech Recognition System Using HMMs with an LVQ-trained Codebook," ATR Technical Report TR-A-0061, August 25, 1989, ATR Auditory and Visual Perception Research Laboratories, Japan.

[3] J. Orlando, R. Mann, and S. Haykin, "Radar Classification of Sea-Ice Using Traditional and Neural Classifiers," *Proc. IJCNN-90-WASH DC, Int. Joint Conf. on Neural Networks*, Washington, D.C., USA, 1990, pp. II - 263-266.

[4] T. Kohonen, "Learning Vector Quantization," Neural Networks 1, Supplement 1, p. 303, 1988.

[5] T. Kohonen, G. Barna, and R. Chrisley, "Statistical Pattern Recognition with Neural Networks: Benchmarking Studies," Proc. IEEE Int. Conf. on Neural Networks, San Diego, Cal., USA, July 24-27, 1988, pp. I - 61-68.

[6] T. Kohonen, "An Introduction to Neural Computing," Neural Networks, vol. 1, pp. 3-16, 1988.

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

[8] J. Makhoul, S. Roucos, and H. Gish, "Vector Quantization in Speech Coding," Proc. IEEE, vol. 73, no. 11, pp. 1551-1588, 1985.

[9] J. Max, "Quantizing for Minimum distortion," IRE Trans. Inform. Theory, vol. IT-6, no. 2, pp. 7-12, Mar. 1960.

[10] A. Gersho, "On the Structure of Vector Quantizers," IEEE Trans. Inform. Theory, vol. IT-28, pp. 157-166, Mar. 1982.

[11] P.L. Zador, "Asymptotic Quantization Error of Continuous Signals and the Quantization Dimension," IEEE Trans. Inform. Theory, vol. IT-28, no. 2, pp. 139-142, Mar. 1982.

[12] J. Kangas, T. Kohonen, J. Laaksonen, O. Simula, and O. Ventä, "Variants of Self-Organizing Maps," *Proc. IJCNN 89, Int. Joint Conf. on Neural Networks*, Washington, D.C., USA, 1989, pp. II - 517-522.

[13] T. Kohonen, K. Mäkisara, and T. Saramäki, "Phonotopic Maps - Insightful Representation of Phonological Features for Speech Recognition," *Proc. Seventh Int. Conf. on Pattern Recognition*, Montreal, Canada, July 30 - August 2, 1984, pp. 182-185.

[14] T. Kohonen, "The 'Neural' Phonetic Typewriter," Computer, vol. 21, pp. 11-22, Mar. 1988.