Premature clustering phenomenon and new training algorithms for LVQ

Mohhamad-Taghi Vakil-Baghmisheh*, Nikola Pavešić

Faculty of Electrical Engineering, Laboratory of Artificial Perception, Systems and Cybernetics, University of Ljubljana, Tražaška 25, Ljubljana, Slovenia

Received 4 March 2002; received in revised form 26 September 2002; accepted 26 September 2002

Abstract

Five existing LVQ algorithms are reviewed. The Premature Clustering Phenomenon, which downgrades the performance of LVQ is explained. By introducing and applying the “equalizing factor” as a remedy for the premature clustering phenomenon a breakthrough is achieved in improving the performance of the LVQ network, and its performance becomes competitive with that of the best known classifiers. For estimating the equalizing factor four different formulas are suggested, which result in four different versions of the LVQ4a algorithm. A new weight-updating formula for LVQ is presented, and the LVQ4b algorithm is presented as implementation of this new weight-updating formula in batch mode training. In addition, four variants of the LVQ4c algorithm are presented as the customized LVQ4b algorithm for pattern mode training.

A meticulous analysis of their performances and that of five early training algorithms has been carried out and they have been compared against each other, on 16 databases of the Farsi optical character recognition problem.

Keywords: Neural networks; LVQ; Pattern recognition; Farsi optical character recognition; Premature clustering phenomenon; Equalizing factor; LVQ4a; LVQ4b; LVQ4c

1. Introduction

Learning vector quantization (LVQ), originally was introduced by Linde et al. [1] and Gray [2] as a tool for image data compression, and later on was adapted by Kohonen [3] for pattern recognition.

Its main idea is to divide the input space $\mathbb{R}^n$ into a number of distinct regions, called decision regions (Voronoi cells), and for each region one codebook (Voronoi) vector is assigned. Classification is performed based on the vicinity of the input vector $x$ to the codebook vectors; $x$ will be classified as the label of its nearest neighbor among codebook vectors. During the training, the codebook vectors and consequently the borders of decision regions are adjusted through an iterative process. Already, there exist five versions of the training algorithm, LVQ1, LVQ2.1, LVQ3, OLVQ1 and CLVQ (Combined LVQ) [4–6]. The main drawbacks of the existing algorithms are their slow convergence and weak recognition rate, because of the premature clustering phenomenon.

In the following, first we will review the existing algorithms. In Section 3, the premature clustering phenomenon will be explained, and the equalizing factor—as our solution to the problem—will be introduced. A new weight-updating formula will be presented in Section 4. In Section 5 the final versions of new training algorithms will be presented. For estimating the equalizing factor, four different formulas are suggested, which result in four different versions of the LVQ4a algorithm. Then, the LVQ4b and LVQ4c algorithms are presented as implementations of the new weight-updating formula. In Section 6, experimental results
obtained on 16 databases of Farsi OCR application are presented. The article ends with a summary and conclusions presented in Section 7.

2. Early training algorithms for LVQ

Let \( \mathbf{x}^q \) be an input vector (from training set): \( \mathbf{x}^q \in \mathbb{R}^n \), \( q = 1, \ldots, Q \).

Let \( \mathbf{w}^m \) be the \( m \)th codebook vector: \( \mathbf{w}^m \in \mathbb{R}^n \), \( m = 1, \ldots, M \).

2.1. LVQ1

1. Find the best-matching codebook vector to \( \mathbf{x}^q \):
   \[
   c = \arg\min_m (\|\mathbf{x}^q - \mathbf{w}^m\|). 
   \]

2. Adjust \( \mathbf{w}^q \):
   \[
   \mathbf{w}^q(t + 1) = \mathbf{w}^q(t) + \varepsilon(t) \cdot [\mathbf{x}^q - \mathbf{w}^q(t)],
   \]
   in which:
   \[
   0 < \varepsilon(t) < 1,
   \]
   \[
   s(t) = \begin{cases} 
   +1 & \text{if the classification is correct,} \\
   -1 & \text{if the classification is wrong,}
   \end{cases}
   \]
   \( \varepsilon(t) \) is a monotonically decreasing function of time and \( \mathbf{w}^q(t) \) represents the sequential values of \( \mathbf{w}^q \) in the discrete-time domain, \( t = 0, 1, 2, \ldots \).

2.2. LVQ2.1

Here, the main idea is differential shifting of the decision borders, so it is suitable for fine-tuning of decision borders. In LVQ2.1, two best-matching codebook vectors \( \mathbf{w}^i \) and \( \mathbf{w}^j \) will be updated simultaneously, provided that:

1. One of them should belong to the correct class (as the label of \( \mathbf{x}^q \)) and the other one to a wrong class.
2. \( \mathbf{x}^q \) should fall in the window that is defined around the midplane of \( \mathbf{w}^i \) and \( \mathbf{w}^j \). (\( \mathbf{x}^q \) is defined to fall in the window if
   \[
   \frac{1 - w}{1 + w} < \frac{d_i}{d_j} < \frac{1 + w}{1 - w},
   \]
   where \( d_i \) and \( d_j \) are the Euclidean distance of \( \mathbf{x}^q \) from \( \mathbf{w}^i \) and \( \mathbf{w}^j \), respectively. A relative window width \( w \) in the interval \([0.2, 0.3]\) is recommended by Kohonen \(^1\), while its legitimate range is \( 0 < w < 1 \).

In the case of fulfilling the afore-mentioned conditions the update rule is:
   \[
   \begin{align*}
   \mathbf{w}^i(t + 1) &= \mathbf{w}^i(t) + \varepsilon(t) [\mathbf{x}^q - \mathbf{w}^i(t)], \\
   \mathbf{w}^j(t + 1) &= \mathbf{w}^j(t) - \varepsilon(t) [\mathbf{x}^q - \mathbf{w}^j(t)],
   \end{align*}
   \]
   where \( \mathbf{w}^i \) is supposed to be in the same class as \( \mathbf{x}^q \), and \( \mathbf{w}^j \) in a different class.

2.3. LVQ3

LVQ3 is the extension of LVQ2.1 in that the learning process of LVQ2 is extended for the case where all \( \mathbf{x}^q, \mathbf{w}^i, \mathbf{w}^j \) belong to the same class to enhance the long run behavior of the training. For this case the update rule will be:
   \[
   \mathbf{w}^k(t + 1) = \mathbf{w}^k(t) + \varepsilon(\mathbf{x}) [\mathbf{x}^q - \mathbf{w}^k(t)], \quad \text{for} \ k = i, j,
   \]
   where \( 0.1 < \varepsilon < 0.5 \) is a stabilizing constant factor. The optimal value of \( \varepsilon \) depends on the size of the window, being smaller for narrower windows.

2.4. The optimized learning-rate LVQ1 (OLVQ1)

Kohonen \(^6\) shows that in LVQ1 algorithm, the scaling factors of different input vectors, when referring to the end of the learning phase, are different, and considers this the main drawback of LVQ1. He assigns an individual learning-rate factor \( \varepsilon(t) \) to every codebook vector \( \mathbf{w}^q \). He sets:
   \[
   \varepsilon(t) = \frac{\varepsilon(t - 1)}{1 + s(t)\varepsilon(t - 1)}
   \]
   and says that if this condition is made to hold for all \( t \), the traces collected up to time \( t \), of all earlier inputs will be scaled down by an equal amount at the end.

Finally he makes the claim that relation (7) gives the optimal values for learning rate and the fastest convergence would be achieved.

This is absolutely an unfounded claim. First of all, he does not provide any proof as to why the \( \varepsilon(t) \) calculated in this way should be considered optimal. Secondly, he accepts that this condition cannot be held for all \( t \), because of instability induced by that and states “since \( \varepsilon(t) \) can also increase, it is especially important that it shall not rise above value 1”, and as a precautionary act suggests limiting it not to go above its initial value, and recommends an initial value of 0.3, and finally, he suggests that training be continued by LVQ1, LVQ2, or LVQ3 to improve accuracy. The questions that arise from this suggestion are that:

If the learning rate of OLVQ1 is really optimized why should it cause instability? And if OLVQ1 is really optimized why should it be followed by another unoptimized algorithm? Although, the theory and formulation of OLVQ1 are groundless, we simulated this algorithm to affirm our discussion with practical results.

\(^1\) In our experiments a value in this range caused instability.
2.5. CLVQ

Kohonen recommends that the learning process be started with OLVQ1, and if necessary continued by LVQ1, LVQ2.1, or LVQ3 with a low initial learning rate value. Henceforth, we assign the name CLVQ (combined LVQ) for any combination of LVQ algorithms.

3. The premature clustering phenomenon and its solution

Actually, the formation of Voronoi cells and codebook vectors in LVQ is a clustering process. Towards the end of training, we will have a small number of wrong patterns in every cluster, but a large majority of patterns will be in the correct clusters. These few wrong patterns will not be effective enough to drive the cell center away from its current location, and their impact on the weight updating formula will be neutralized by the impact of many correct patterns. As the result, cluster centers and borders freeze and the continuation of training does not change them, and a large learning rate will result only in oscillation around these prematurely formed centers and borders. We will call this phenomenon the “Premature Clustering Phenomenon”.

The solution for premature clustering, is to increase the number of wrong patterns in every cluster to drive the cell center away. Since we cannot have extra patterns, we should increase their impact by considering different weights for correct and wrong patterns to equalize their total impact on shifting cluster centers. This idea can be implemented by assigning different step sizes for correct and incorrect predictions to have equal effective numbers of correct and wrong training patterns, for every codebook vector and in every epoch. Although, superficially it may seem that we are discriminating among prototype vectors, in fact we give different weights to correct and wrong patterns, which construct the prototype vectors.

For example, when one misclassified vector has remained out of 50 input vectors, which make the decision region and its prototype vector, we will weigh the correctly classified pattern 49 times less than the misclassified pattern. So one misclassified pattern will have an impact equal to 49 correctly classified patterns in shifting decision borders and this will enormously increase system’s sensitivity to misclassified patterns.

By implementing the equalizing factor, at the same time we will realize discrimination theory of learning. This theory inspired by observing human activities in learning has been proposed by the authors, and is stated for the ease of reference, in the following:

_Discrimination theory of learning:_ Considering that some pattern associations are learned more easily than others, humans exercise different levels of effort in learning different pattern associations. Further, this effort is different for a specific pattern association at different stages of learning.

4. Preliminary LVQ4 algorithm

Let \( \mathbf{x}^q \) be an input vector (from training set): \( \mathbf{x}^q \in \mathbb{R}^q \), \( q=1,\ldots,Q \) and let \( \mathbf{w}_m \) be the \( m \)th codebook vector: \( \mathbf{w}_m \in \mathbb{R}^q \), \( m=1,\ldots,M \).

1. Find the best-matching codebook vector to \( \mathbf{x}^q \):
   \[
   c = \arg \min_m (\| \mathbf{x}^q - \mathbf{w}_m \|).
   \] (8)

2. Adjust \( \mathbf{w}_c \):
   \[
   \mathbf{w}_c(t+1) = \mathbf{w}_c(t) + \alpha(n) \mathbf{s}(n) (\mathbf{x}^q - \mathbf{w}_c(t))
   \] (9)
   in which:
   \[
   0 < \alpha(n) < 1,
   \]
   \[
   \mathbf{s}(n) = \begin{cases} \mathbf{d}_c(n) & \text{if the classification is correct,} \\ -1 & \text{if the classification is wrong,} \end{cases}
   \] (10)

\( \alpha(n) \) is a monotonically decreasing function of epoch number \( n \), \( \mathbf{w}_c(t) \) represents sequential values of \( \mathbf{w}_c \) in discrete-time domain, \( (t = 0, 1, 2, \ldots) \), and \( \mathbf{d}_c(n) \) is the “equalizing factor”. In addition, \( \alpha(n) \) and \( \mathbf{d}_c(n) \) must be kept constant during a training epoch.

\( \mathbf{d}_c(n) \) is defined as:
\[
\mathbf{d}_c(n) = \frac{E_c(n)}{P_c(n)},
\] (11)
where \( E_c(n) \) and \( P_c(n) \), respectively, are the numbers of wrong and correctly classified patterns by codebook vector \( c \) in the current epoch. Considering that till the end of current epoch \( E_c(n) \) and \( P_c(n) \) will not be known, we will estimate the value of \( \mathbf{d}_c(n) \) using the results of preceding epochs.

In the sequel, we will present four methods for estimating \( \mathbf{d}_c(n) \), but previous to that in the following theorem, we show analytically that by defining \( \mathbf{d}_c(n) \) as in Eq. (11), the effective numbers of correct and wrong training patterns, for every codebook vector and in every epoch, will be equal. In addition, this theorem will lead us to a new weight updating formula.

_Theorem._ Setting the value of \( \mathbf{d}_c(n) \) equal to \( E_c(n)/P_c(n) \), where \( E_c(n) \) and \( P_c(n) \) respectively, are the numbers of wrong and correctly classified patterns by codebook vector \( c \) in the current epoch—will equalize the effective numbers of correct and wrong patterns that affect the codebook vector \( c \) in the current epoch.

_Proof._ Without losing generality, we will consider just one codebook vector. Therefore, we will eliminate the subscript \( c \) from formulas for simplicity of notation. In addition, \( \beta(n) \) and \( \gamma(n) \) are kept constant during a training epoch. Subsequently, we will use these equations for training:

\[
\mathbf{w}(t+1) = \mathbf{w}(t) + \beta \mathbf{x}^q - \mathbf{w}(t)) = \beta \mathbf{x}^q + (1 - \beta) \mathbf{w}(t)
\]
if classification is correct, (12)
\( w(t + 1) = w(t) + \gamma [x^d - w(t)] = \gamma x^d + (1 - \gamma)w(t) \)

if classification is wrong. \( \text{(13)} \)

**State 1:** Let \( x^1 \) be the first input vector and be classified correctly

\( w(1) = \beta x^1 + (1 - \beta)w(0) \) \( \text{(14)} \)

and \( x^2 \) be the second input vector and be misclassified

\( w(2) = \gamma x^2 + (1 - \gamma)w(1) \) \( \text{(15)} \)

by inserting from Eq. (14) into Eq. (15) we obtain

By following the same method, that we used previously, we obtained correctly classified and the \( l \)th correctly classified and the \( r \) th correctly classified and the \( (l + r) \)th classified wrongly:

\( w(l + r) \cong \gamma \{x^{l+r} + x^1 + \ldots + x^l\} + w(0) \) \( \text{(23)} \)

which obviously shows equal effective numbers of correctly and wrongly classified patterns.

**State 4:** Now we consider \( m = l + r \) patterns, the first \( l \) correctly classified and the last \( r \) patterns as wrong ones.

By following the same method, that we used previously, we obtain

\( w(l + r) \cong \gamma \{x^{l+r} + x^1 + \ldots + x^l\} + \beta [x^l + \ldots + x^1] + [1 - \gamma r - l\beta]w(0). \)

**State 5:** If we consider \( m = l + r \) patterns, \( l \) patterns correctly classified and \( r \) patterns as misclassified ones, regardless of their order of presentation to the network, by following the preceding procedure we get the same results, and this ends the proof. \( \square \)

**Considerations:**

1. If in Eq. (10), instead of considering the equalizing factor in the weight updating formula of correctly classified patterns, we apply it to the weight-updating formula of misclassified patterns, i.e. we define \( s(n) \) as follows:

\[
 s(n) = \begin{cases} 
 1 & \text{if the classification is correct}, \\
 -d'_c(n) & \text{if the classification is wrong}, 
\end{cases} 
\]

\( \text{(26)} \)

the definition of \( d'_c(n) \) should be

\[
 d'_c(n) = \frac{1}{d_c(n)}. 
\]

Since through time, the number of misclassified patterns is decreasing and the number of correctly classified patterns is increasing, \( d'_c(n) \) will increase through time, and this will destabilize the system.

2. Formula (25) offers a new weight-updating formula for LVQ, which will be detailed in Section 5.2.

3. We will have 9 variants of LVQ4 algorithm, which will be presented in three groups: LVQ4a, LVQ4b and LVQ4c.

5. LVQ4 Algorithms

5.1. LVQ4a Algorithms

In this method weight updating is done in pattern mode and is based on the original formula (Eq. (10)). Considering that in this method the equalizing factor will not be known
till the end of epoch, we will estimate its value using the results of previous epochs.

**Step 1:** Find the best-matching codebook vector to input vector $x^n$:

$$c = \arg \min_n \left( \|x^n - w_n\| \right). \quad (28)$$

**Step 2:** Adjust $w_c$:

$$w_c(t + 1) = w_c(t) + \alpha(n)s(n)[x^n - w_c(t)], \quad (29)$$

in which:

$$0 < s(n) < 1,$$

$$s(n) = \begin{cases} d_c(n) & \text{if the classification is correct}, \\ -1 & \text{if the classification is wrong}, \end{cases} \quad (30)$$

$n$ is the epoch number and $s(n)$ is a monotonically decreasing function of $n$, for instance:

$$\alpha(n) = k_1 \exp(-n/\tau_1) + k_2 \exp(-n/\tau_2) + k_3 \exp(-n/\tau_3), \quad (31)$$

or $\alpha(n) = \begin{cases} k_1 & \text{if } n \leq n_0, \\ k_2 \exp\left(-\frac{n - n_0}{\tau}\right) & \text{if } n > n_0. \end{cases} \quad (32)$

### 5.1.1. Methods for estimating $d_c(n)$

For estimating $d_c(n)$ four methods are offered:

1. $\hat{d}_c(n) = d_{total}(n - 1), \quad (33)$
2. $\hat{d}_c(n) = d_c(n - 1), \quad (34)$
3. $\hat{d}_c(n) = d_c(n - 1) \frac{d_{total}(n - 1)}{d_{total}(n - 2)}, \quad (35)$
4. $\hat{d}_c(n) = d_c(n - 1) \frac{d_{total}(n - 1)}{d_{total}(n - 2)}. \quad (36)$

Considering the irregularities as $n = 1$ and 2 and the cases of zero terms in the denominator, more detailed formulas would be:

1. $\hat{d}_c(n) = \begin{cases} \varepsilon & \text{if } n = 1, \\ d_{total}(n - 1) & \text{if } n \geq 2. \end{cases} \quad (37)$
2. $\hat{d}_c(n) = \begin{cases} \varepsilon & \text{if } n = 1, \\ d_c(n - 1) & \text{if } n \geq 2 \text{ and } P_c(n - 1) \neq 0, \\ \lambda & \text{if } n \geq 2 \text{ and } P_c(n - 1) = 0. \end{cases} \quad (38)$
3. $\hat{d}_c(n) = \begin{cases} \varepsilon & \text{if } n = 1,2, \\ d_c(n - 1) \frac{d_{total}(n - 1)}{d_{total}(n - 2)} & \text{if } n \geq 3 \text{ and } P_c(n - 1) \neq 0, \\ P_c(n - 1) \neq 0, \quad (39) \\ \lambda & \text{if } n \geq 3 \text{ and } P_c(n - 1) = 0. \end{cases}$

and in the fourth method, $d_c(n - 1)$ is calculated as follows:

$$d_c(n - 1) = \begin{cases} E_c(n - 1) & \text{if } P_c(n - 1) \neq 0, \\ \lambda & \text{if } P_c(n - 1) = 0, \end{cases} \quad (41)$$

where $\varepsilon$ and $\lambda$ are positive numbers ($\varepsilon < 1$ and $\lambda \geq 1$), and $d_{total}(n)$ is the ratio of total numbers of wrongly and correctly classified patterns in the $n$th epoch. Based on our simulation results, suggested values for $\varepsilon$ and $\lambda$ are 0.3 and 3, respectively, although, the network performance is not too sensitive to the value of $\lambda$.

The variants of LVQ4a, created by using estimation methods 1–4 (i.e., Eqs. (37)–(41)), will be called LVQ4a1 to LVQ4a4, respectively.

### 5.2. LVQ4b algorithm

This method is based on exact implementation of formula (25), in batch mode training. Weight-updating will be performed after presentation of all training patterns that constitute an epoch, and the equalizing factor will be calculated precisely.

By grouping correctly classified and misclassified patterns by codebook vector $c$, we have

$$w_c(n) = \gamma(x_c^{1/\gamma} + \cdots + x_c^{1/\gamma}) + \beta(x_c^{1/\beta} + \cdots + x_c^{1/\beta}) + w_c(n - 1). \quad (42)$$

If we set

$$\gamma(n) = -\alpha(n) \quad (43)$$

$\beta(n)$ will be

$$\beta(n) = -\alpha(n)\gamma(n) = \alpha(n)d_c(n). \quad (44)$$

By substituting for $\gamma$ and $\beta$

$$w_c(n) = -\alpha(n)V_c + \alpha(n)d_c(n)U_c + w_c(n - 1). \quad (45)$$

And by regrouping, we get

$$w_c(n) = \alpha(n)[d_c(n)U_c - V_c] + w_c(n - 1). \quad (46)$$

**Algorithm**

**Step 0:** $q = 1$ and $n = 1.$
Step 1: Find the best-matching codebook vector to input vector $x^q$:
$$c = \arg \min_m (\|x^q - w_m\|).$$

Step 2: If classification is correct, then:
$$\begin{cases} 
P_c(n) = P_c(n) + 1, \\
U_c = U_c + x^q
\end{cases}$$
if classification is wrong, then:
$$\begin{cases} 
E_c(n) = E_c(n) + 1, \\
V_c = V_c - x^q
\end{cases}$$
if $q = Q$, i.e., one epoch is complete go to Step 3, else $q = q + 1$, and go to Step 1.

Step 3: For every codebook vector:
if $P_c(n) \neq 0$, then:
$$d_c(n) = E_c(n)/P_c(n)$$
else $d_c(n) = \lambda$.

Step 4: For every codebook vector:
$$w_c(t + 1) = w_c(t) + \alpha(n)[d_c(n)U_c - V_c],$$
where $n$ is the epoch number and $\alpha(n)$ is a monotonically decreasing function of $n$, for instance as defined in Eq. (31) or Eq. (32).

Step 5: Reset $P_c(n)$, $d_c(n)$, $U_c$ and $V_c$; increase $n$, and go to Step 1.

5.3. LVQ4c algorithms

It is possible to implement the main equation of LVQ4b (i.e. Eq. (45)) in pattern mode. To this end, the weight-updating formula should be changed as follows:
$$w_c(t + 1) = \begin{cases} 
\frac{w_c(t) + \alpha(n)d_c(n)x^q}{P_c(n)} & \text{if classification is correct,} \\
\frac{w_c(t) - \alpha(n)x^q}{P_c(n)} & \text{if classification is wrong.}
\end{cases}$$
(47)

In this algorithm $d_c(n)$ must be estimated through one of the methods used in LVQ4a algorithm, i.e. Eq. (37)–(41).

Remark. The only difference of this algorithm with LVQ4a is in the weight-updating formula. Based on our simulation results—despite its superiority over earlier versions of LVQ—its variants show as being inferior to the variants of LVQ4a and LVQ4b in terms of recognition rate, monotony and speed of convergence.

6. Experiments

6.1. Databases

18 databases composed of feature vectors of 32 isolated characters of the Farsi alphabet, sorted in three groups, were created through various feature extraction methods, including: PCA, vertical, horizontal and diagonal projections, zoning, pixel change coding and some combinations of them, with the number of features varying from 4 to 78 per character according to Table 1.
Fig. 1. A sample set of Machine-Printed Farsi Characters (isolated case), font size = 28 pt.

Fig. 2. Variations of one sample character among different fonts, font size = 26 pt.

For creating these databases, 34 Farsi fonts, which are used in publishing online newspapers and web sites, were downloaded from the Internet. Fig. 1 demonstrates a whole set of isolated characters of one sample font printed as text. Fig. 2 shows variations of one sample character among different fonts. Then, 32 isolated characters of these 34 Farsi fonts were printed in an image file. Eleven sets of these fonts were boldface and one set was italic. In the next step, by printing the image file and scanning it with various brightness and contrast levels, two additional image files were obtained. Then, using a 65 × 65 pixel size window the character images were separated into images of isolated characters. After applying a shift-radius invariant image normalization, and by reducing the sizes of character images to 24 × 24 pixel, the features vectors were extracted as detailed in the appendix.

6.2. Simulation results

In all simulations, 2/3 of the feature vectors, obtained from original image and the first scanned image were assigned for the training set, and 1/3 of them obtained from the second scanned image were assigned for the test set. Therefore, 68 samples per character were used for training and 34 samples per character for testing. Thus, the total number of samples in training and test sets are 2176 and 1088, respectively.

Many functions were tried for learning rate, and after experiments the best learning rate functions were found to be:

1. \( \alpha(n) = k (4 \exp(-n/30) + 20 \exp(-n/10) + 40 \exp(-n)) \) (48)
2. \( \alpha(n) = \begin{cases} k & \text{if } n \leq 10, \\ k \exp(-(n-10)/20) & \text{if } n > 10. \end{cases} \) (49)
3. \( \alpha(n) = k \left( 1 + \frac{25}{n - 0.6} \right) \) (50)

and \( k \) must be readjusted for every simulation.

For the early versions of LVQ, if applicable, formulas (49) and (50) were more suitable, and for all other algorithms using formulas (48) and (49) produced better results. Training was carried out for 70 epochs for all the cases, except CLVQ, which is explained in the following:

Algorithms LVQ2.1 and LVQ3 failed to converge. The conclusion is simple: they can be useful only for fine adjustment of the borders. After testing all combinations for CLVQ, the LVQ1 followed by LVQ2.1 was chosen as the best combination, with 10 and 60 epochs of training, respectively.

Table 2 shows the recognition errors obtained on databases of group A, where \( P \) is the number of prototype patterns per class. These results must be considered sub-optimal and slightly better performance could be achieved by a better fitting learning rate function. Based on our experiments, in the case of optimal adjustment, maximum decrease in the recognition errors could be 25%, although very unlikely.

6.2.1. Analysis

The first evident point in Table 2 is the outstanding performance of new training algorithms (excluding LVQ4c2 which is volatile) and the weak performance of early algorithms i.e. LVQ1, OLVQ1 and CLVQ on both training and test sets of all databases. Then the weak performance of all algorithms on test sets of databases db1 and db2 becomes apparent, although our algorithms have a relatively better performance. This weak performance on db1 and db2 should be attributed to inappropriate implementation of the feature extraction method.

The next visible point is the impact of data normalization. We see that it has a major impact on the performances of early algorithms, and a trivial impact on those of LVQ4a2 and LVQ4a3 on every database. If we exclude db1 and db2, the performances of LVQ4a2, LVQ4a3, LVQ4a4 and LVQ4b could be considered almost invariant by data normalization, while its impact on the performance of LVQ4c2 is negative on every database. Data normalization on db5 has a positive impact on the performance of LVQ4a1.
Table 2
Recognition errors of LVQ algorithms on databases of group A and B, with \( P = 3 \)

<table>
<thead>
<tr>
<th>Training algorithm</th>
<th>Database</th>
<th>LVQ1</th>
<th>OLVQ1</th>
<th>CLVQ</th>
<th>LVQ4a1</th>
<th>LVQ4a2</th>
<th>LVQ4a3</th>
<th>LVQ4a4</th>
<th>LVQ4b</th>
<th>LVQ4c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>db1</td>
<td>122</td>
<td>140</td>
<td>67</td>
<td>11</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>db2</td>
<td>142</td>
<td>150</td>
<td>61</td>
<td>12</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>db3</td>
<td>116</td>
<td>152</td>
<td>43</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>db4</td>
<td>121</td>
<td>122</td>
<td>39</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>db5</td>
<td>140</td>
<td>147</td>
<td>64</td>
<td>27</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>Test</td>
<td>db1</td>
<td>82</td>
<td>86</td>
<td>53</td>
<td>48</td>
<td>27</td>
<td>32</td>
<td>32</td>
<td>38</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>db2</td>
<td>88</td>
<td>91</td>
<td>56</td>
<td>48</td>
<td>39</td>
<td>38</td>
<td>36</td>
<td>41</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>db3</td>
<td>61</td>
<td>73</td>
<td>21</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>db4</td>
<td>56</td>
<td>58</td>
<td>20</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>db5</td>
<td>68</td>
<td>71</td>
<td>33</td>
<td>15</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Train</td>
<td>dbn1</td>
<td>125</td>
<td>135</td>
<td>55</td>
<td>21</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>dbn2</td>
<td>156</td>
<td>150</td>
<td>67</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>14</td>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>dbn3</td>
<td>109</td>
<td>98</td>
<td>40</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>dbn4</td>
<td>99</td>
<td>99</td>
<td>49</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>10</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>dbn5</td>
<td>111</td>
<td>113</td>
<td>53</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td>Test</td>
<td>dbn1</td>
<td>79</td>
<td>75</td>
<td>49</td>
<td>43</td>
<td>38</td>
<td>30</td>
<td>29</td>
<td>33</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>dbn2</td>
<td>97</td>
<td>97</td>
<td>56</td>
<td>50</td>
<td>51</td>
<td>47</td>
<td>45</td>
<td>34</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>dbn3</td>
<td>61</td>
<td>51</td>
<td>21</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>dbn4</td>
<td>46</td>
<td>49</td>
<td>24</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>dbn5</td>
<td>55</td>
<td>56</td>
<td>27</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>24</td>
</tr>
</tbody>
</table>

We also see that all algorithms have achieved their best recognition rate on the test set of db4, excluding LVQ4a1, which has slightly better recognition rate on the test set of db3. The best overall results have been obtained by LVQ4a3 on both db4 and dbn4, although the results are so close to those of LVQ4a2, LVQ4a4 and LVQ4b that the differences could be attributed to unoptimal learning rate functions. Later we will say more about the cause of these differences. Among new algorithms the weakest results on normalized data have been obtained by LVQ4c2. To make a conclusive judgment, performing further simulations is necessary.

We made the conditions more difficult by decreasing the number of prototype patterns from 3 to 2 and 1, and the obtained results are demonstrated in Table 3. In this table, the results obtained by another member of LVQ4c class, i.e. LVQ4c3 has been presented for the first time. This one has achieved better results than its classmate LVQ4c2, especially on normalized data, and again the weak performance of LVQ4c2 on normalized data is evident.

Since LVQ4a2 has gained better results with both \( P = 1 \) and \( P = 2 \), it could be concluded that with a small number of prototype vectors LVQ4a2 shows a better performance. Later we will see that these small differences in recognition errors could be attributed to the impact of initial values of prototype vectors. In addition, this experiment reveals the positive impact of normalization on their performance. Therefore the impact of data normalization depends on the database, training algorithm and the size of network, notwithstanding the performances of LVQ4a and LVQ4b algorithms are less volatile.

The optimum value for window width in LVQ2.1 was found to be \( w = 0.1 \) for all our databases. By going far from this value, recognition error will increase, regardless of the training-time length. In other words, continuation of the learning process for a longer time will not improve the performance. This phenomenon happens with all other parameters of all LVQ algorithms. In another case study on the Circle-in-the-square problem [7], the optimum value for \( w \) was found to be slightly smaller than 0.1.

As we discussed in Section 2, the claimed optimized attribute for OLVQ1, is unattractive. The obtained results are enough proof for this; its performance is the worst among all variants of LVQ algorithms. The only advantage that can be attributed to OLVQ1 is its faster convergence over LVQ1 at the beginning of training due to its large learning rate, but it is not able to decrease the error sufficiently. In other words, in all the cases excluding dbn3, its final recognition error is not less than that of LVQ1. Fig. 3 compares its convergence against that of LVQ1 on dbn4, on which it shows its best results equivalent to that of LVQ1. In this figure the
Table 3
Recognition errors of LVQ algorithms on db4 and dbn4, with \( P = 1 \) and 2

<table>
<thead>
<tr>
<th>Training algorithms</th>
<th>Database</th>
<th>LVQ4a1</th>
<th>LVQ4a2</th>
<th>LVQ4a3</th>
<th>LVQ4a4</th>
<th>LVQ4b</th>
<th>LVQ4c2</th>
<th>LVQ4c3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train db4</td>
<td>18</td>
<td>7</td>
<td>9</td>
<td>15</td>
<td>14</td>
<td>23</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>dbn4</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>39</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Test db4</td>
<td>9</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>7</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>dbn4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>( P = 1 )</td>
<td>Train db4</td>
<td>95</td>
<td>33</td>
<td>57</td>
<td>68</td>
<td>62</td>
<td>72</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>dbn4</td>
<td>60</td>
<td>19</td>
<td>29</td>
<td>41</td>
<td>44</td>
<td>84</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>Test db4</td>
<td>45</td>
<td>16</td>
<td>28</td>
<td>36</td>
<td>30</td>
<td>32</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>dbn4</td>
<td>30</td>
<td>12</td>
<td>16</td>
<td>23</td>
<td>22</td>
<td>40</td>
<td>27</td>
</tr>
</tbody>
</table>

Fig. 3. Convergence diagrams of OLVQ1 and LVQ1 obtained on db4, \( P = 3 \).

Table 4
Recognition errors of LVQ algorithms on databases of group C, with \( P = 3 \)

<table>
<thead>
<tr>
<th>Training algorithm</th>
<th>Database</th>
<th>LVQ1</th>
<th>CLVQ</th>
<th>LVQ4a1</th>
<th>LVQ4a2</th>
<th>LVQ4a3</th>
<th>LVQ4a4</th>
<th>LVQ4b</th>
<th>LVQ4c2</th>
<th>LVQ4c3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train db7</td>
<td>121</td>
<td>62</td>
<td>23</td>
<td>4</td>
<td>9</td>
<td>16</td>
<td>16</td>
<td>48</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>db8</td>
<td>219</td>
<td>94</td>
<td>41</td>
<td>17</td>
<td>18</td>
<td>42</td>
<td>20</td>
<td>107</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>db10</td>
<td>185</td>
<td>61</td>
<td>18</td>
<td>11</td>
<td>11</td>
<td>15</td>
<td>16</td>
<td>66</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>db11</td>
<td>99</td>
<td>38</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>27</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>db12</td>
<td>209</td>
<td>84</td>
<td>33</td>
<td>24</td>
<td>19</td>
<td>39</td>
<td>22</td>
<td>81</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>db13</td>
<td>143</td>
<td>70</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>8</td>
<td>37</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Test db7</td>
<td>53</td>
<td>29</td>
<td>11</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>8</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>db8</td>
<td>114</td>
<td>49</td>
<td>19</td>
<td>8</td>
<td>7</td>
<td>24</td>
<td>8</td>
<td>57</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>db10</td>
<td>92</td>
<td>33</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>38</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>db11</td>
<td>43</td>
<td>18</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>db12</td>
<td>103</td>
<td>44</td>
<td>16</td>
<td>10</td>
<td>10</td>
<td>21</td>
<td>10</td>
<td>46</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>db13</td>
<td>65</td>
<td>34</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>20</td>
<td>4</td>
</tr>
</tbody>
</table>

dynamic recognition error\(^2\) has been depicted versus the epoch number.

Table 4 shows the recognition errors obtained on databases of group C. Apparent in this table is the weak performance of LVQ1 and CLVQ, followed by LVQc2. Among databases the worst results belong to db8 and db12 and the best to db11, on which almost all the algorithms have achieved their best performance; db13 should be considered to rank the second best among databases, but considering the number of its features the speed of any algorithm on it will be much less than that on db11. The best overall results belong to LVQ4a2, LVQ4a3, and LVQ4b.

\(^2\)The dynamic recognition error is obtained while the network is under training, and after presenting any pattern one prototype pattern will change, so it is different with the recognition error, which is obtained after training. The difference between these two could be sizable, especially at the beginning of training, during which the network goes through a lot of changes in every epoch.
In addition, LVQ4c3 shows better performance than that of its classmate LVQ4c2.

The last point about algorithms and databases is the speed of convergence. As a sample, Fig. 4 compares two convergence diagrams of CLVQ1 and LVQ4a2 obtained on db4, and from this figure we see that even the convergence speeds of new algorithms are better than those of early algorithms.

Although very good recognition results have been obtained on db11, quite competitive with those obtained on db4, but the convergence of all algorithms on db11 are slower than those on the databases of group A and B. Fig. 5 compares the convergence diagrams obtained by LVQ4a2 on db4, db11 and dbn4. Another factor, which in all algorithms could affect the speed of convergence, is the method of initialization. If, instead of the first samples, we choose the cluster centers obtained by k-means algorithm as initial prototype vectors, it can improve the performance very positively in terms of final recognition error, monotony and speed of convergence. This initializing method is especially recommended for databases with a huge number of samples, or with a very slow rate of convergence. To present an example we have chosen db12, our worst database from group C. Table 5 compares the results obtained on db12 with two modes of initialization. Without exception, all algorithms have responded positively to this method. Although, in the cases of LVQ1 and LVQ4c3 its impact is insignificant comparing to that of other algorithms. In addition, this method of initialization, eases the process of parameter tuning by decreasing sensitivity to it.

**7. Conclusions**

1. We have discovered and explained the occurrence of the Premature Clustering Phenomenon in the LVQ network, which is responsible for its poor performance.
2. In Ref. [7] we have compared the performances of LVQ4a1 and LVQ4a2 against those of LVQ1, CLVQ,

<table>
<thead>
<tr>
<th>Table 5</th>
<th>Comparing recognition errors of LVQ algorithms on db12, with two mode of initialization and $P = 3$. Mode a: starts from first samples, Mode b: starts from the cluster centers created by $k$-means algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training algorithm</td>
<td>LVQ1</td>
</tr>
<tr>
<td>Train Mode a</td>
<td>209</td>
</tr>
<tr>
<td>Mode b</td>
<td>189</td>
</tr>
<tr>
<td>Test Mode a</td>
<td>103</td>
</tr>
<tr>
<td>Mode b</td>
<td>97</td>
</tr>
</tbody>
</table>
10. The learning rate function was divided into two parts: 

Thus, the learning rate function, \( \eta(n) = \alpha(n)s(n) \) [see Eqs. (9) and (29)]. If we consider \( s(n) \) as the relative learning rate, \( \alpha(n) \) will be the envelope of the learning rate function. LVQ algorithms are extremely sensitive to both relative learning rate and envelope tuning, which is an exhausting task, and necessitates developing an automated approach. The learning rate function, can affect both convergence speed and final recognition error. By defining \( s(n) \) using the equalizing factor we managed to solve part of the automation problem. If we could offer a method for calculating \( s(n) \), with regard to the discrimination theory, we believe that the sensitivity to learning rate function will be solved, and LVQ algorithm will be entirely automated.

11. A drawback of the LVQ algorithms, i.e. sensitivity to the envelope of learning rate function, remains unsolved.

12. Based on the obtained results, LVQ4a2, LVQ4a3 and LVQ4b are the best among all variations, in terms of speed and monotonity of convergence, final recognition error, and unwavering excellent performance with different numbers of prototype vectors on various databases.

**Acknowledgements**

This work has been partially supported within the framework of the Slovenian–Iranian Bilateral Scientific Cooperation Treaty. The authors would like to thank Alan McConnell Duff for linguistic revision.

**Appendix A. Feature extraction methods**

**A.1. Group A—db1 to db5**

These databases were created by extracting the principal components, extracted by a single layer feedforward linear neural network with Generalized Hebbian Training Algorithm (GHA) [8,9], as summarized in the following:

For extracting principal components, we used \( m-l \) single layer feedforward linear network with Generalized Hebbian Training Algorithm (GHA).

- **db1** To train the network: \( 8 \times 8 \) non-overlapping blocks of the image of every character were considered as an input vector. The image was scanned from top left to bottom right and \( l \) was set equal to 8. Therefore for every character 72 features were extracted. The training was performed with 34 samples per character and the learning rate was set to \( \eta = 7 \times 10^{-3} \). To give enough time for the network to learn the statistics of data, training procedure was repeated for three epochs.

- **db2** The same as db1, but \( l \) was set equal to 6, thus for every character 54 features were extracted.
The image matrix of every character was converted into a vector, by scanning vertically from top left to bottom right, then this vector was partitioned into 9 vectors which were inserted into the network as 9 input vectors. In this way, 72 features were extracted for every character.

db4 Similar to db1, but the dimension of the input blocks was considered to be $24 \times 3$, i.e. every three rows were considered as one input vector. In this way, for every character 64 features were extracted.

db5 Similar to db4, but $l$ was set equal to 6, thus for every character 48 features were extracted.

A.2. Group B—dbn1 to dbn5

These databases are normalized versions of db1 to db5. After creating every database, the feature vectors were normalized by mapping the $i$th component of all the vectors into the interval $[0, 1]$.

A.3. Group C—db6 to db13

db6 This database was created by the zoning method. Each character image was divided into four overlapping squares and the percentage of black pixels in each square was obtained. The size of overlap was set to two pixels in each edge, which yields the best recognition rate. The best recognition rate on this database does not exceed 53%, so its features were used only in combination with other features.

db7 Pixel change coding was used to extract the feature vectors of this database.

db8 The feature vectors of this database were extracted by vertical and horizontal projections.

db9 The feature vectors of this database were extracted by diagonal projection. Ten components from the beginning and seven components from the end were deleted, because their values were zero for all characters. The best recognition rate on this database does not reach 85%, so its features were used only in combination with other features.

db10 By concatenating the feature vectors of db8 and db9, feature vectors of this database were extracted.

db11 The feature vectors of this database were created by concatenating the feature vectors of db6 and db7.

db12 The feature vectors of this database were created by concatenating the feature vectors of db6 and db8.

db13 The feature vectors of this database were created by concatenating the feature vectors of db11 and some selected features from db8, that is 10 features from the middle of both vertical and horizontal projections.

References


About the Author—MOHAMMAD-TAGHI VAKIL-BAGHMISHEH was born in 1961 in Tabriz, Iran. He received his B.Sc. and M.Sc. degrees in Electronics, from Tehran University in 1987 and 1991. From 1991 to 1995 he was a member of the academic staff in Tabriz University. In 1995 he joined the Laboratory of Artificial Perception, Systems and Cybernetics, Faculty of Electrical Engineering, University of Ljubljana, Slovenia and has just completed his Ph.D. working on neural networks and pattern recognition.

About the Author—NIKOLA PAVESIĆ was born in 1946. He received his B.Sc. degree in Electronics, M.Sc. degree in Automatics, and Ph.D. degree in Electrical Engineering from the University of Ljubljana, Slovenia, in 1970, 1973 and 1976, respectively. Since 1970 he has been a staff member at the Faculty of Electrical Engineering in Ljubljana, where he is currently head of the Laboratory of Artificial Perception, Systems and Cybernetics. His research interests include pattern recognition, neural networks, image processing, speech processing, and information theory. He is the author and co-author of more than 100 papers and 3 books addressing several aspects of the above areas. Professor Nikola Pavesić is a member of IEEE, the Slovenian Association of Electrical Engineers and Technicians (Meritorious Member), the Slovenian Pattern Recognition Society, and the Slovenian Society for Medical and Biological Engineers. He is also a member of the editorial boards of several technical journals.