ON COMPUTING THE FUZZIFIER IN FLVQ: A DATA DRIVEN APPROACH

L. B. ROMDHANE*, B. AYEB† and S. WANG‡

Department of Mathematics and Computer Sciences,
Faculty of Sciences, University of Sherbrooke, Que, Canada

*benromdh@DMI.usher.ca
†ayeb@DMI.usher.ca
‡wang@DMI.usher.ca

Received 2 February 1998
Revised 25 March 2002
Accepted 25 March 2002

Clustering is an important research area that has practical applications in many fields. Fuzzy clustering has shown advantages over crisp and probabilistic clustering, especially when there are significant overlaps between clusters. Most analytic fuzzy clustering approaches are derived from Bezdek’s fuzzy c-means algorithm. One major factor that influences the determination of appropriate clusters in these approaches is an exponent parameter, called the fuzzifier. To our knowledge, no theoretical reason leading to an optimal setting of this parameter is available. This paper presents the development of an heuristic scheme for determining the fuzzifier. This scheme creates close interactions between the fuzzifier and the data set to be clustered. Experimental results in clustering IRIS data and in codebook design required for image compression reveal a good performance of our proposal.

Keywords: Fuzzy clustering; fuzzy set membership; fuzzifier; simulated annealing.

1. Introduction

Clustering is an important research area that has practical applications in a variety of fields, including pattern recognition and image compression. Clustering algorithms attempt to partition the input data into groups, i.e. clusters, such that patterns within a cluster are more similar to each other than to patterns in other clusters. In fuzzy clustering, the affinity of an input vector to the considered cluster is modeled by a membership function whose range is [0, 1] rather than either/or (0/1). Fuzzy clustering algorithms have shown advantages over their crisp/probabilistic counterparts; especially when there is significant overlap between clusters. For a concise review of fuzzy clustering schemes and an experiment-based comparison with their crisp/probabilistic counterparts, we refer the reader to the literature; e.g. Refs. 4 and 11.

By adopting the concept of fuzzy membership, various well-known fuzzy clustering algorithms were proposed in the literature; e.g. fuzzy C-means (FCM, for short), Fuzzy Learning Vector Quantization (FLVQ, for short), Fuzzy Kohonen Clustering Networks. The performance of fuzzy clustering algorithms based on FCM is affected by an exponent parameter, called the fuzzifier. This parameter affects both the convergence rate and the cluster validity of the considered algorithm. To date, there are no reliable criteria for the selection of the optimal fuzzifier for a given set of training vectors. Instead, the best value (resp. a range of values) is fixed through experimentation; i.e. this choice is still heuristic. In the current work, we develop a new approach to computation of the fuzzifier that takes into account the degree of overlap between clusters. Our proposed scheme follows a “natural” simulated
annealing behavior to address the cluster validity and the convergence rate problems.

The remainder of this paper is organized as follows. In Sec. 2, we formulate the clustering problem and give the main notations adopted in this paper. In Sec. 3, we describe the Descending FLVQ (↓FLVQ, for short); and discuss the problems caused by an improper setting of the fuzzifier. Section 4 outlines our proposal and compares it to ↓FLVQ. In Sec. 5, numerical experiments are reported. The final section includes discussions and conclusions.

2. Notation and Problem Formulation

Let \( c \) be an integer \((1 < c < n)\) and let \( X = \{x_1, x_2, \ldots, x_n\} \) be a set of \( n \) feature vectors in \( \mathbb{R}^p \). \( X \) is a numerical object datum. \( x_j \) is a representation of the \( j \)th object in \( X \). \( x_{kj} \) is the \( k \)th feature value of the \( j \)th object. The number of clusters \( c \) can be specified \( a \ priori \) or computed from the input data set using criteria of optimality such as the fuzzy hypervolume and density. Clusters may also have centroids, i.e. class prototypes. Let \( V = \{\nu_1, \nu_2, \ldots, \nu_c\} \) be the set of centroids, \( \nu_i \in \mathbb{R}^p (1 \leq i \leq c) \).

FCM seeks optimal partitions of the input data by minimizing the weighted (within groups) sum of squared errors objective function \( J_m \) defined by Ref. 3:

\[
J_m(U, V; X) = \sum_{k=1}^{c} \sum_{i=1}^{n} (u_{ik})^m \|x_k - \nu_i\|_A^2
\]

(1)

subject to the constraints:

\[
0 \leq u_{ik} \leq 1 \quad \forall i, k
\]

\[
0 < \sum_{k=1}^{n} u_{ik} < n \quad \forall i
\]

\[
\sum_{i=1}^{c} u_{ik} = 1 \quad \forall k
\]

(2)

where \( u_{ik} \) is the membership degree of the \( k \)th feature vector in the \( i \)th cluster. \( U = [u_{ik}] (1 \leq i \leq c, 1 \leq k \leq n) \) is a fuzzy \( c \)-partition of the data set \( X \); and \( \|x_k - \nu_i\|_A = \sqrt{(x_k - \nu_i)^T A (x_k - \nu_i)} \) is an inner product norm where \( A \) is any positive definite weight matrix. The parameter \( m \) in (1) is called a fuzzifier. The following theorem states necessary conditions on \( U \) and \( V \) in order to minimize the objective function in (1).

**Theorem 1**

Assume \( \|x_k - \nu_j\|_A^2 > 0, \forall j, k. (U, V) \) may minimize \( J_m \) only if, for \( m > 1 \):

\[
u_i = \frac{\sum_{k=1}^{n} (u_{ik})^m x_k}{\sum_{k=1}^{n} (u_{ik})^m} \quad \forall i
\]

(3)

(4)

The following section describes ↓FLVQ and discusses the problems caused by an improper setting of the fuzzifier \( m \).

3. The Role of the Fuzzifier in FCM and ↓FLVQ

↓FLVQ\(^4\) depicted in Fig. 1 is run in three steps. In the first step, the required parameters are fixed and initial class prototypes are guessed. In the second step, a set of class prototypes \( V \) and a fuzzy \( c \)-partition \( U \) are computed. In the third step, \( V \) and/or \( U \) are used. In ↓FLVQ, the fuzzifier \( m \) is computed at each epoch, i.e. iteration counter \( t \), by:

\[
m_t = m_0 - t \Delta_m.
\]

(5)

We should notice the fact that ↓FLVQ is FCM when the fuzzifier \( m \) is fixed (i.e. \( \Delta_m = 0 \)); and is Hard C-Means\(^3\) when \( m \) is fixed to one and \( u_{ik} \in \{0, 1\} \forall i, k \). ↓FLVQ starts with a large value \( m_0 \) and decreases it gradually through iteration to a low terminating value. Tsao et al. in Ref. 7 adopted the same strategy for computing the fuzzifier in Fuzzy Kohonen Clustering Networks. The decreasing scheme in (5) may address the convergence rate and the cluster validity problems — for more details, we refer the reader to Ref. 4. However, as reported by Bezdek and Pal, ↓FLVQ is very sensitive to the initial and final values of the fuzzifier. In ↓FLVQ, at each epoch, the fuzzifier is the same for all clusters. This may be non-optimal since the amount of overlap is not the same for all fuzzy clusters. Chung and Lee reported in Ref. 6 that an appropriate specification of the fuzzifier requires knowledge of the characteristics of the
Step 1. Initialization:
1.1) Store unlabeled data $X = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^p$.
1.2) Pick $1 < c < n$; $N = \max$ iterations; $\| \cdot \|_A$ an inner product norm; $\epsilon, m_0 > 0$ and $\Delta_m$ real positive numbers.
1.3) Guess $V_0 = \{\nu_{1,0}, \nu_{2,0}, \ldots, \nu_{c,0}\} \in \mathbb{R}^p$.

Step 2. Computation of class prototypes:
For $t = 1, 2, \ldots, N$ Do:
2.1) Compute $m_t$ using (5).
2.2) For $k=1$ to $n$ Do:
2.2.1) Compute membership degrees of all data samples in cluster $k$ using (3).
2.2.2) Update prototype vector of class $k$ using (4).
2.3) If $E_t = \|V_t - V_{t-1}\|^2 = \sum_{i=1}^{c} \|\nu_{i,t} - \nu_{i,t-1}\|^2 < \epsilon$ Stop.
2.4) Else $V \leftarrow V_t; U \leftarrow U_t$; Next $t$

Step 3. Output $V$ and/or $U$

Fig. 1. The $\downarrow$FLVQ clustering algorithm.

---

data set at hand. Hence, close interactions between the fuzzifier and the data to be clustered should be created. It is this property that we investigate and use in our proposal, discussed subsequently.

4. Our Proposal: DD-$\downarrow$FLVQ

In real arbitrary data, clusters may or may not overlap. When boundaries between clusters are “clear”, the partition of input vectors is a nearly crisp decision-making process. Therefrom, low values of the fuzzifier should be adopted. However, when boundaries are “fuzzy”, we should slow down the decision process by adopting large values of the fuzzifier. In order to model this behavior, we investigate the use of the membership values in computing the fuzzifier. In fact, one can say that a large (resp. small) overlap between clusters results in similar (resp. distinct) memberships of a given pattern in all possible clusters. The more these memberships are away from each other, the more a vector is attracted toward a particular class. In a formal manner, we propose the following scheme:

$$m_k(t) = m_0 \gamma_c (1 - \beta_k(t)) + \delta$$

where $m_0$ is the initial fuzzifier as in Ref. 4, $t$ is iteration counter, $\delta$ is a small real number, $\gamma_c$ is a scaling factor defined by:

$$\gamma_c = \frac{c}{c - 1}$$

and $\beta_k(t)$ is the sum of squared memberships of pattern $k$ in all clusters; i.e.:

$$\beta_k(t) = \sum_{i=1}^{c} (u_{ik}(t))^2 \quad \forall k = 1, \ldots, n.$$

Using the constraints on the computed memberships in (2), and the schemes for computing $m_k(t)$ and $\beta_k(t)$ in (6) and (8) respectively, we can state that:

$$\frac{1}{c} \leq \beta_k(t) \leq 1 \quad \forall k = 1, \ldots, n$$

$$\delta \leq m_k(t) \leq m_0 + \delta$$

$$\forall k = 1, \ldots, n, t = 0, 1, \ldots$$

According to (10), $\delta$ is the smallest value the fuzzifier can take. Since $m_k(t)$ must be in $[1, \infty]$ (see Theorem in Sec. 3), we must have $\delta > 1$. But this parameter should remain small. Through iteration a pattern may be attracted toward a particular cluster, in which case its membership in that cluster will increase. Consequently, $\beta_k$ will increase, and the fuzzifier computed by (6) will decrease. This “natural” decrease meets the requirements of simulated annealing behavior. Figure 2 depicts $\beta_k(t)$ in the case of a two-cluster problem.
The fuzzifier in our scheme is computed for each pattern, formulas (3) and (4) will be changed respectively to:

$$u_{ik,t} = \left( \sum_{j=1}^{c} \left( \frac{\|x_k - \mu_i\|}{\|x_k - \mu_j\|} \right)^{\frac{1}{\gamma}} \right)^{-1}$$

\forall i, k, t = 0, 1, \ldots; \quad (11)

$$\nu_i,t = \frac{\sum_{k=1}^{n} (u_{ik})^{m_k(t)} x_k}{\sum_{k=1}^{n} (u_{ik})^{m_k(t)}} \quad \forall i, t = 0, 1, \ldots \quad (12)$$

where $m_k(t)$ is computed by (6) at each epoch $t$. We call our modification of the original $\downarrow$FLVQ,\textsuperscript{4} Data Dependent $\downarrow$FLVQ ($\downarrow$FLVQ, for short). The changes required to convert Fig. 1 to $\downarrow$FLVQ can be summarized as follows. First, step 2.1 is omitted. Second, in step 2.2.1 we use (11) instead of (3). Third, we compute the $c$ fuzzifiers $m_k(t)$ with (6), and then use (12) in step 2.2.2 instead of (4). Hereafter, we comment on the computational burden with these modifications of $\downarrow$FLVQ. In our proposal, the fuzzifier is pattern-dependent which makes it computationally more expensive than that adopted in $\downarrow$FLVQ. For a given pattern, computing the fuzzifier requires $(c + 1)$ basic operations at each epoch. This can be easily seen when rewriting (6) as:

$$m_k(t) = m_0 \gamma_c + \delta - m_0 \gamma_c \left\{ \sum_{i=1}^{c} (u_{ik}(t))^2 \right\}$$

$$\forall k, t = 1, 2, \ldots \quad (13)$$

Consequently, at each epoch, our scheme requires $n \times (c + 1)$ basic operations to compute all $c$ fuzzifiers for a data set $X$ of size $n$, whereas in $\downarrow$FLVQ, we need only only two basic operations to compute the fuzzifier at each epoch — see (5). But, since in real-world problems the number of clusters is relatively small, and regarding overall the clustering process, this difference can be negligible. Both approaches are compared in the next section.

5. Numerical Experiments

$\downarrow$FLVQ is tested and compared to $\downarrow$FLVQ with two sets of experiments. The first example uses the IRIS data of Anderson.\textsuperscript{3} In the second example, both schemes are used in codebook design required by image compression based on vector quantization.

5.1. Clustering the IRIS data

IRIS\textsuperscript{3} consists of three subspecies (Setosa, Versicolour, Virginica), each containing 50 samples. Each sample is described by four features: sepal length (SL), sepal width (SW), petal length (PL) and petal width (PW). IRIS has been extensively used by researchers to illustrate various clustering (unsupervised) and classifier (supervised) designs mainly because cluster boundaries are “fuzzy”. We have used the Euclidean norm for computation of $J_m$ in (1) and for the stopping criterion. For a fair comparison between $\downarrow$FLVQ and $\downarrow$FLVQ, we use the same value for the initial fuzzifier $m_0$ and the error threshold $\epsilon$; i.e. the shared parameters by both approaches. We based our comparison upon two criteria: the number of epochs required by the algorithm to terminate and the number of resubstitution misclassifications. The number of crisp clustering errors is evaluated by using the computed cluster centers $V$ as a basis for the one nearest prototype method\textsuperscript{4} (1-NP, for short). Initialization of class prototypes can be achieved using many techniques; e.g. unsupervised learning,\textsuperscript{8} a random process.\textsuperscript{7} In this example, we have adopted the same method as used in Ref. 4. From the input
Table 1. Initial prototype vectors of IRIS using (16).

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\nu^T_{1,0}$</td>
<td>4.30 2.00 1.00 1.00</td>
</tr>
<tr>
<td>$\nu^T_{2,0}$</td>
<td>6.10 3.20 3.95 1.30</td>
</tr>
<tr>
<td>$\nu^T_{3,0}$</td>
<td>7.90 4.40 6.90 2.50</td>
</tr>
</tbody>
</table>

For data set $X$, we compute the minimum and maximum feature values respectively by:

$$ m_j = \min_k \{x_{kj}\} \quad \forall j = 1, \ldots, p \tag{14} $$

$$ M_j = \max_k \{x_{kj}\} \quad \forall j = 1, \ldots, p \tag{15} $$

With these, we compute the $j$th component of the $i$th initial prototype as:

$$ v_{ij} = m_j + (i - 1) \left( \frac{M_j - m_j}{c - 1} \right) \quad \forall j = 1, \ldots, p \quad \forall i = 1, \ldots, c. \tag{16} $$

Initial prototypes are shown in Table 1.

In this first set of experiments, we have fixed the following parameters to arbitrary values: $\epsilon = 0.01$, $\Delta_m = 0.004$, $\delta = 1.1$ and $N(\max\text{ iter.}) = 100$. Using the initialization process (16), we have run $\dagger$FLVQ and DD--$\dagger$FLVQ for different values of the initial fuzzifier $m_0$ ranging over $[2.0, 7.0]$ (see Table 2). Indeed, these are useful limits to avoid numerical instability of $\dagger$FLVQ as reported by Bezdek and Pal in Ref. 4. In Table 2, the first column shows the value(s) of the initial fuzzifier $m_0$. The second column shows the final prototypes. The third column shows the iterations required.

Table 2. Sample runs of $\dagger$FLVQ and DD--$\dagger$FLVQ with different values of the initial fuzzifier $m_0$.

<table>
<thead>
<tr>
<th>$m_0$</th>
<th>$\dagger$FLVQ</th>
<th>DD--$\dagger$FLVQ</th>
<th>Iterations</th>
<th>Misc. Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\nu^T_1$</td>
<td>(5.004, 3.403, 1.485, 0.252)</td>
<td>(5.007, 3.412, 1.475, 0.249)</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>or $\nu^T_2$</td>
<td>(5.893, 2.762, 4.370, 1.401)</td>
<td>(5.873, 2.752, 4.326, 1.379)</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>or $\nu^T_3$</td>
<td>(6.783, 3.054, 5.657, 2.057)</td>
<td>(6.824, 3.067, 5.714, 2.076)</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>2.5</td>
<td>$\nu^T_1$</td>
<td>(5.001, 3.389, 1.494, 0.252)</td>
<td>(5.004, 3.411, 1.477, 0.247)</td>
<td>9</td>
</tr>
<tr>
<td>or $\nu^T_2$</td>
<td>(5.914, 2.792, 4.384, 1.399)</td>
<td>(5.837, 2.761, 4.285, 1.335)</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>or $\nu^T_3$</td>
<td>(6.700, 3.039, 5.558, 2.038)</td>
<td>(6.695, 3.053, 5.592, 2.073)</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>3.0</td>
<td>$\nu^T_1$</td>
<td>(5.002, 3.386, 1.496, 0.252)</td>
<td>(4.986, 3.399, 1.479, 0.246)</td>
<td>8</td>
</tr>
<tr>
<td>or $\nu^T_2$</td>
<td>(5.927, 2.804, 4.396, 1.404)</td>
<td>(5.813, 2.777, 4.260, 1.327)</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>or $\nu^T_3$</td>
<td>(6.674, 3.033, 5.524, 2.028)</td>
<td>(6.646, 3.052, 5.538, 2.088)</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>3.5</td>
<td>$\nu^T_1$</td>
<td>(5.005, 3.384, 1.499, 0.251)</td>
<td>(4.971, 3.365, 1.477, 0.242)</td>
<td>8</td>
</tr>
<tr>
<td>or $\nu^T_2$</td>
<td>(5.945, 2.819, 4.409, 1.411)</td>
<td>(5.783, 2.795, 4.213, 1.301)</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>or $\nu^T_3$</td>
<td>(6.633, 3.023, 5.469, 2.005)</td>
<td>(6.631, 3.051, 5.533, 2.105)</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>4.0</td>
<td>$\nu^T_1$</td>
<td>(5.007, 3.384, 1.500, 0.250)</td>
<td>(4.975, 3.353, 1.492, 0.244)</td>
<td>8</td>
</tr>
<tr>
<td>or $\nu^T_2$</td>
<td>(5.952, 2.825, 4.413, 1.414)</td>
<td>(5.765, 2.841, 4.249, 1.331)</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>or $\nu^T_3$</td>
<td>(6.617, 3.019, 5.448, 1.994)</td>
<td>(6.602, 3.024, 5.497, 2.037)</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>4.5</td>
<td>$\nu^T_1$</td>
<td>(5.009, 3.384, 1.501, 0.249)</td>
<td>(4.937, 3.309, 1.475, 0.242)</td>
<td>9</td>
</tr>
<tr>
<td>or $\nu^T_2$</td>
<td>(5.959, 2.830, 4.419, 1.418)</td>
<td>(5.744, 2.829, 4.218, 1.308)</td>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>or $\nu^T_3$</td>
<td>(6.600, 3.014, 5.426, 1.982)</td>
<td>(6.589, 3.017, 5.445, 1.994)</td>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>5.0</td>
<td>$\nu^T_1$</td>
<td>(5.012, 3.385, 1.502, 0.249)</td>
<td>(5.030, 3.423, 1.467, 0.241)</td>
<td>10</td>
</tr>
<tr>
<td>or $\nu^T_2$</td>
<td>(5.972, 2.838, 4.430, 1.424)</td>
<td>(5.877, 2.835, 4.326, 1.374)</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>or $\nu^T_3$</td>
<td>(6.575, 3.008, 5.392, 1.964)</td>
<td>(6.536, 3.006, 5.317, 1.982)</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>
number of epochs (iterations) for the algorithms to terminate, whereas we find in the fourth column the number of crisp errors w.r.t. the 1-NP method. We can say that DD–FLVQ has a lower misclassification rate than FLVQ. However, DD–FLVQ requires more iterations than FLVQ to terminate.

The parameter $\beta_k$ computed by (8) and used for the computation of the fuzzifier in (6) is pattern-dependent. As discussed in Sec. 4, the evolution of $\beta_k$ depends on the “location” of the considered pattern. Figure 3 depicts the evolution of $\beta_k$ through iterations for five randomly chosen patterns (see Fig. 4 for a two-dimensional plot of the location of these patterns, where only the SL and PL features are shown).

These plots were obtained with $m_0 = 6.5$, $\epsilon = 0.01$, $N(\text{max. iter.}) = 100$, and $\delta = 1.1$. As expected, we notice from Fig. 3 that $\beta_k$ approaches one rapidly for patterns lying on “clear” boundaries, while its evolution remains relatively slow for patterns lying on “fuzzy” boundaries. Conversely, depending on pattern’s location, the fuzzifier take small/large values allowing crisp/fuzzy clustering processes.

5.2. Image compression based on vector quantization

Clustering algorithms can be used for codebook design, an essential task in signal and image compression. In this second set of experiments, we aim at testing our approach in clustering large-scale high-dimensional data sets. The image is encoded by replacing each block of pixels by a closest prototype w.r.t. the 1-NP method. FLVQ and DD–FLVQ were tested for image compression. The initial class prototypes were computed using (16). We used two test images of size $L \times L = (198 \times 198)$ (see Fig. 5). The pixel intensities in these images take values between zero and 255; i.e. the representation of each pixel requires eight bits.

Each image is spatially compressed into rectangular blocks of size $(3 \times 3)$ pixels that form vectors of fixed size ($= 9$). This representation of an image of size $L \times L$ produces $L \times L/9$ input vectors. Thereby, both images were divided into 4356 patterns. FLVQ and DD–FLVQ were used to compute codebooks consisting of $c = 256 = 2^8$ centroids for both images. Hence each window of 9 pixels is represented by 8 bits. In other words, the compression rate is $8/9 (\approx 0.889)$ bits per pixel (bpp). As evaluation criteria, we have used the normalized square error $\text{NSE}^{13}$ and the peak signal to noise ratio $\text{PSNR}^{9}$ defined respectively by:

$$\text{NSE} = \frac{\sum_{x_j \in \text{image}} (x_j - \hat{x}_j)^2}{\sum_{x_j \in \text{image}} x_j^2}$$

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{L^2} \sum_{x_j \in \text{image}} (x_j - \hat{x}_j)^2}$$

Recall that IRIS data, although highly correlated, consists only of 150 four-dimensional vectors.
where $\hat{x}_j$ is the pixel’s reconstructed gray level value, $x_j$ is the original gray level value and $L^2$ is the total number of pixels in the image. In this second set of experiments, we have used the following parameters: $\epsilon = 10$, $m_0 = 8.0$, $\Delta_m = 0.02$, $\delta = 1.6$, $c = 256$ and $N(\text{max. iter}) = 100$ for both images. The encoded images by $\triangledown$FLVQ and DD–$\triangledown$FLVQ are depicted in Figs. 6 and 7 respectively.

Both algorithms are equal in performance w.r.t. NSE and PSNR criteria (see Table 3). In encoding the image in Figure 5(a), both algorithms terminated in 26 epochs.\(^b\) DD–$\triangledown$FLVQ required 24 epochs.

\(^b\)Recall that at each epoch, all of the training samples are presented to the algorithm.
Evaluation of the encoded images by ↓FLVQ and DD↓FLVQ using the NSE and the PSNR criteria.

<table>
<thead>
<tr>
<th></th>
<th>NSE</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓FLVQ</td>
<td>0.029</td>
<td>22.513</td>
</tr>
<tr>
<td>DD↓FLVQ</td>
<td>0.029</td>
<td>22.506</td>
</tr>
<tr>
<td>↓FLVQ</td>
<td>0.005</td>
<td>28.587</td>
</tr>
<tr>
<td>DD↓FLVQ</td>
<td>0.005</td>
<td>28.517</td>
</tr>
</tbody>
</table>

Fig. 7. Encoded images by ↓FLVQ and DD↓FLVQ.

Fig. 8. ↓FLVQ and DD↓FLVQ error rates decay (a) image I and (b) image II.
to encode the image in Fig. 5(b), while $\downarrow$FLVQ required 29 epochs. Figure 8 depicts the evolution through epochs of the objective function $J_m(t)$ defined by (1). Remark that in the case of DD--$\downarrow$FLVQ, the objective function $J_m$ is defined as in (1) by replacing the fuzzifier $m$ by $m_k$ for all input patterns $x_k$.

Through this second set of experiments, we have shown that our DD--$\downarrow$FLVQ succeeds in clustering large-scale high-dimensional data sets as well as $\downarrow$FLVQ. Besides, these experiments reported that our scheme does not require many tests to choose the initial fuzzifier; i.e. $m_0$. In clustering IRIS data, $m_0$ took values ranging over $[2.0, 7.0]$ to avoid numerical instability of $\downarrow$FLVQ as reported by Bezdek and Pal in Ref. 4. But this heuristic does not apply for the second set of experiments. In fact, for $m_0 \in \{6.0, 6.5, 7.0\}$ $\downarrow$FLVQ was unstable. Results were mediocre for $m_0 = 7.5$ w.r.t. NSE and PSNR criteria. The best result was obtained with $m_0 = 8.0$. DD--$\downarrow$FLVQ gave approximately the same NSE and PSNR values for all these tested fuzzifiers. By embedding a heuristic that takes into consideration the amount of fuzziness between subgroups in the computation of the fuzzifier, DD--$\downarrow$FLVQ can outperform $\downarrow$FLVQ with no need to test many annealing schemes for the fuzzifier.

## 6. Discussion and Future Work

In this paper, we proposed a heuristic scheme to address the problem of the choice of the fuzziness parameter in $\downarrow$FLVQ. We made the fuzzifier depend on the data set in order to take into consideration the “amount” of fuzziness between clusters. Experimental results have shown that our modified $\downarrow$FLVQ can have a lower resubstitution misclassification rate than $\downarrow$FLVQ in clustering IRIS. Unlike $\downarrow$FLVQ, the second set of experiments on image compression have shown that DD--$\downarrow$FLVQ is less sensitive to the initial value of the fuzzifier. However, it usually requires more epochs (iterations) to terminate because it is computationally more demanding than $\downarrow$FLVQ. This could be a shortcoming when clustering a large set of data into a large number of clusters. We do not claim that DD--$\downarrow$FLVQ is better than $\downarrow$FLVQ in all cases. It is undoubtedly possible to find data sets on which one algorithm performs better than the other. By creating closer interaction between the fuzzifier and the data, the fuzzifier is able to adapt “naturally” to the amount of fuzziness between subgroups. Embedding more knowledge about the data into the computation of the fuzzifier; e.g. cluster width,$^{10}$ fuzzy hypervolume$^8$ might circumvent the problem of choosing the best fuzzifier w.r.t. the data set to cluster. For this, an analysis of the effect of such parameters on the fuzzifier should be done.

## References