Probabilistic PCA Self-Organizing Maps

Ezequiel López-Rubio, Juan Miguel Ortiz-de-Lazcano-Lobato, and Domingo López-Rodríguez

Abstract—We present a probabilistic neural model, which extends Kohonen’s self-organizing map (SOM) by performing a Probabilistic Principal Components Analysis (PPCA) at each neuron. Several self-organizing maps have been proposed in the literature to capture the local principal subspaces, but our approach offers a probabilistic model while it has a low complexity on the dimensionality of the input space. This allows to process very high dimensional data to obtain reliable estimations of the probability densities which are based on the PPCA framework. Experimental results are presented, which show the map formation capabilities of the proposal with high dimensional data, and its potential in image and video compression applications.

Index Terms—Self-organizing maps, Probabilistic Principal Components Analysis (PPCA), competitive learning, unsupervised learning, dimensionality reduction, handwritten digit recognition.

I. INTRODUCTION

T
HE concept of self-organization seems to explain several neural structures of the brain that perform invariant feature detection [15]. These structures inspire the proposal of computational maps designed to explore multidimensional data. The original self-organizing map (SOM) was proposed by Kohonen [22], where each neuron had a weight vector to represent a point of the input space. It was followed by the adaptive subspace self-organizing map (ASSOM), which was first presented as an invariant feature detector [23]. This property has been further studied [24], and its relations with wavelets and Gabor filters have been reported ([36]; [25]; [37]). Each neuron of an ASSOM network represents a subset of the input data with a vector basis. This vector basis is adapted so that the data points of the subset are as close as possible to its spanned vector subspace. Hence, a description of the local geometry of the input data is built. The concept of a neuron which represents a linear subspace can be traced to the subspace classifier by Oja [33]. The minimization of the mean squared error of the projection errors on the subspaces leads naturally to the Principal Components Analysis. The ASSOM model extends these ideas by considering the self-organization of subspace neurons, which can also be found in Dony and Haykin’s Optimally Adaptive Transform Coding [13]. The ASSOM has been successfully applied to the handwritten digit recognition problem [65] which many neural network researchers have addressed [47]. Also, it has been used for texture segmentation [43]. This work is related with a supervised variant of the ASSOM, called SASSOM (Supervised Adaptive-Subspace Self-Organizing Map), first proposed by Ruiz del Solar & Köppen [42].

Finally, SOM networks are adequate to create topographic maps, which are representations of the input space. This ability is inherited by the ASSOM network, which has been taken as a standard for comparison with other algorithms that make these maps [55]. The ASSOM does not define any probability model on the input space. This is not the case for the Generative Topographic Mapping (GTM) by Bishop et al. [8], which is a constrained mixture of Gaussians. A latent space is defined with a reduced dimensionality, and a lattice of units is set up in the latent space. The ASSOM lacks the ability to learn the local mean vectors. The PCA self-organizing map [30] solves this problem by learning both the mean vector and the covariance matrix at each neuron. The Self-Organizing Mixture Networks (SOMN), by Yin & Allinson [64] also follow this line. Unfortunately, the use of the full covariance matrix makes them computationally heavy for high dimensional data sets.

The Self-Organizing Mixture Model (SOMM) by Verbeek et al. [59] uses a version of the expectation-maximization (EM) method to produce an extension of the SOM where a mixture of restricted Gaussians is defined. Nevertheless, it has some scalability problems when the size of the map grows.

Other families of self-organizing maps include kernel-based topographic maps ([56], [57], [58]), where Gaussian kernels are defined around a centroid, and topographic Independent Component Analysis [19], which introduces the use of ICA instead of PCA.

Our aim here is to develop a self-organizing model with online learning of the local subspaces of an input distribution, which is based on the Probabilistic PCA framework. Furthermore, our proposal has a low complexity both in the size of the map and in the input space dimension, so that it is suited for high dimensional data. This sort of datasets is common in certain typical applications of self-organizing

Manuscript received September 5, 2007. This work was partially supported by the Ministry of Education and Science of Spain under Projects TIC2003-03067 and TIN2006-07362, and by the Autonomous Government of Andalusia (Spain) under Projects P06-TIC-01615 and P07-TIC-02800. Ezequiel López-Rubio and Juan Miguel Ortiz-de-Lazcano-Lobato are with the Department of Computer Languages and Computer Science, University of Málaga, Bulevar Louis Pasteur 35, 29071 Málaga, Spain. Phone: (+34) 95 213 71 55; fax: (+34) 95 213 13 97; e-mails: {ezeqlr,jmortiz}@lcc.uma.es.

Domingo López-Rodríguez is with the Department of Applied Mathematics, University of Málaga, Spain (e-mail: dlopez@ctima.uma.es).

(c) 2009 IEEE. See the IEEE Copyright Policy.
The largest eigenvalues of the transformed data. The transformation is given by the original data comes to lie in the first coordinates of the orthogonal linear transformation where the most variance of the data covariance matrix $C$, which is decomposed as $C=U\Lambda U^T$. The eigenvalues in the diagonal matrix $\Lambda$ are sorted in decreasing order. Hence, conventional PCA shows that the optimal linear dimensionality reduction to $q$ dimensions, with $q<d$, involves an orthogonal projection on the principal directions given by the eigenvectors corresponding to the $q$ largest eigenvalues of $C$: $z_q = U_q^T(t-\mu)$. The optimality is in the least squares sense, and the optimal reconstruction error is $E_r = \|t - \mu - z_q U_q^T\|$. This starting point has given rise to many dimensionality reduction techniques, in particular Probabilistic PCA and subspace methods [61].

Probabilistic PCA [53] models the probability distribution of the input data by a multivariate Gaussian. This extends conventional PCA, which does not define any probability model (two examples of non-Gaussian PCA for mixture modelling can be found in [66] and [39]). In order to achieve dimensionality reduction, PPCA considers that the input data come from a linear transformation of a $q$-dimensional vector of latent variables. This yields a probability density model which corresponds to the $q$-dimensional principal subspace of the data, with the addition of isotropic noise in all directions. The optimal dimensionality reduction can be obtained by projection of the original data on the principal subspace, just as in conventional PCA. Hence, the trailing $d-q$ directions are considered noisy and they are not even estimated.

On the other hand, the subspace method by Moghaddam and Pentland [35] splits the input space into two orthogonal subspaces: the principal subspace of dimension $q$, and its complementary subspace of dimension $d-q$. The probability density for the original input space is built by the product of two independent Gaussians, one for each subspace. The original data are projected on the two subspaces, but the projection on the complementary subspace is not explicitly computed.

Hence we obtain a procedure which has similarities to PPCA, and in fact has been proven to be equivalent [61]. The two approaches need to estimate the mean vector $\mu$ and the covariance matrix $C$. The main difficulty is the robust estimation of $C$, since a plain maximum likelihood estimator will fail to produce a full rank $C$ because of the small sample size with respect to the high dimensionality of the input space. The solution is to restrict $C$ so that it has less degrees of freedom, in order to obtain robust estimations. The subspace method projects the original data into the eigenspace of $C$, and then uses only the $q$ leading directions. The remaining $d-q$ projections are estimated by a parameter which can be shown to correspond to PPCA’s isotropic noise parameter. Therefore, the two methods are seen as different ways to implement a robust estimator of $C$ for high dimensional data with comparatively small sample sizes.

As we see, the search for linear subspaces where the data lie is guided in a probabilistic framework by the PCA principal directions associated with the $q$ leading eigenvalues of the covariance matrix of a Gaussian distribution. These models are easily extended to mixtures of Gaussians. Then we have $H$ linear transformations, one per mixture component. In principle, the mixture components are not bound to each other, and they are adjusted to optimize some objective function, such as the log-likelihood of the input data. Nevertheless, it is more convenient in data visualization applications to enforce the self-organization of the subspaces. This yields faithful representations of the input distribution, which can be used to explore the structure of complex high dimensional datasets. Under the robust estimation perspective considered before, self-organization of linear subspaces can be regarded as a way to impose additional conditions in the estimated covariance matrices, since every covariance matrix is constrained to be similar to those of the neighbouring units. This procedure effectively reduces the variability of the covariance matrices, which leads to increased robustness against the relative lack of input samples for large $d$. Furthermore, the number of input samples which contribute significantly to the estimation of a particular covariance matrix is increased, as the information is shared among neighbouring units, and this leads to more robustness.

Other approaches to restrict Gaussian mixture models include enforcing the covariance matrices to lie in a low-dimensional matrix subspace, as shown in [11]. These restrictions alleviate the data insufficiency problem, which prevents reliable estimation of full covariance mixture models.

Perhaps the best known self-organizing map model which learns linear subspaces is the ASSOM. It does not define a probability density model, but its objective function is the average expected projection error. So, it is closely related to the reconstruction error of PCA, since each ASSOM neuron stores a $q$-dimensional orthonormal vector base which could be identified with $U_q$. The dimensionality reduction capability...
of these maps is somehow reduced by the absence of a mean vector in the neurons, which is equivalent to assume $\mu = 0$.

Other self-organizing models which do not learn subspaces, such as the SOMM and kernel-based topographic maps, assume Gaussian densities. Hence we can think of them as models where the number of retained principal directions is $q = 0$. In fact, they enforce $U = I$, and no PCA transformation is performed. This implies that their covariance matrices are diagonal, and they only learn the variances in each direction given by the eigenvalue matrix $A$. We can conclude that many self-organizing map proposals include certain elements of subspace-based probability density modelling, but not completely. Some of them do not define probability densities, and others do not learn subspaces.

It should not be inferred from the discussion above that subspace modelling is only suitable for the minimization of the reconstruction error. In supervised classification applications, the relevant features for discrimination are commonly confined to low-dimensional subspaces of the input space. In this context, the objective is to capture the most relevant directions for classification. Aladjem [1] and Calo [9] have proposed methods to model class-conditional probability densities with Gaussian mixtures by projection pursuit. A related approach to supervised classification is given by Saito et al. [9], who search the most discriminant subspaces by estimating the difference among the probability distributions of the projected data belonging to each class. In this latter case, the probability densities are not assumed to be Gaussian. These methods are not so closely related with self-organizing maps, which are unsupervised systems in most cases. Nevertheless, unsupervised projection pursuit methods are also available for feature selection [10]. In these approaches it is desired that the probability density of the projected data is as non-Gaussian as possible.

III. THE PPCASOM MODEL

A. Mixture model

Each neuron of the map stores a PPCA model [53] to perform a dimensionality reduction from the observed (input) space dimension $d$ to the latent (reduced) subspace dimension $q$, with $q < d$. The observed data $t$ depend linearly on the latent variables in $x$, with a mean vector $\mu$ and a noise model $\xi$:

$$t = Wx + \mu + \xi$$

(1)

The latent variables are defined to be independent and Gaussian with unit variance and zero mean, i.e., $x \sim N(0, I)$. The noise model is also Gaussian such that $\xi \sim N(0, \sigma^2 I)$, and the $d \times q$ parameter matrix $W$ contains the factor loadings. This formulation implies that the observation vectors are also normally distributed, $t \sim N(\mu, C)$, with a covariance matrix $C = \sigma^2 I + WW^T$.

The PPCASOM model is defined as a mixture of $H$ PPCA components, with prior probabilities or mixing proportions $\pi_i$:

$$p(t_n) = \sum_{i=1}^{H} \pi_i p(t_n | i)$$

(2)

where $p(t_n | i)$ is the PPCA probability density associated with mixture component $i$:

$$p(t_n | i) = (2\pi)^d/2 |C_i|^{-1/2} \exp(-E_{ni}^2/2)$$

(3)

$$E_{ni}^2 = (t_n - \mu_i)^T C_i^{-1} (t_n - \mu_i)$$

(4)

Now we need a procedure to compute $p(t_n | i)$ in $O(d)$. From PPCA we know that the parameter matrix $W_i$ can be decomposed as

$$W_i = U_i q \{K_q q^2 \sigma_q^2 \}^{1/2} R_i$$

(5)

where the columns of the $q \times q$ matrix $U_i q$ are the eigenvectors corresponding to the $q$ principal directions of the subspace of neuron $i$, $K_q$ is a $q \times q$ diagonal matrix with the corresponding eigenvalues, and $R$ is a rotation matrix which may be computed as the matrix of eigenvectors of the $q \times q$ matrix $W_i^T W_i$. Then the decomposition of $W_i$ is completed by normalization of the columns of $W_i R_i^{-1}$.

The error term $E_{ni}$ can be expressed in terms of this decomposition (see [53] for details):

$$E_{ni}^2 = z_{ni}^T K_q^{-1} z_{ni} + E_{ni}^2 / \sigma_q^2$$

(6)

where $z_{ni}$ is the projection of $t_n - \mu_i$ onto the principal subspace of neuron $i$ and $E_{ni}$ is the reconstruction error corresponding to the reconstruction vector $i' n$:

$$z_{ni} = U_i q (t_n - \mu_i)$$

(7)

$$E_{ni}^2 = \left| I_i - i' n \right|^2$$

(8)

Finally, the determinant of $C_i$ can be computed as:

$$|C_i| = \left( \sigma_q^2 \right)^d q!$$

(9)

where the $k_j$ are the $q$ first eigenvalues of $C_i$, which are stored in $K_q^i$.

B. Self-organization

At each time step $n$ the network is presented a data sample $t_n$. We introduce a discrete hidden variable $z_n$ whose value (from 1 to $H$, where $H$ is the number of mixture components) indicates which component generated the data sample $t_n$. In order to achieve self-organization, we only allow distribution models for $z_n$ which have the property that one component is the most probable and the probability decreases with the topological distance to that component (see [32], [59]). A topology is defined in the network so that the topological distance between mixture components $i$ and $i'$ is called $\delta(i, i')$. A flat rectangular lattice may be used:

$$\delta(i, i') = \left\| (x_i, y_i) - (x_{i'}, y_{i'}) \right\|$$

(11)

where $(x_i, y_i)$ is the coordinate vector of mixture component $i$ in the lattice. Other lattice topologies and/or geometries could
be also considered, like hexagonal lattice topologies and toroidal lattice geometries.

Hence we consider the following set of distributions $Q_n^i = \{q_{ai}, \ldots, q_{oai}\}$ for $z_i$:

$$q_{ai}(z_i = i) = p_{ai}(z_i = i) = P(z_i = i \mid q_{ai}) \propto \exp \left( \frac{-\Delta(i, j)}{\Delta(n)} \right)$$

where $\sum_{i=1}^{H} q_{ai}(z_i = i) = 1$. Note that $\Delta$ is the topological distance function and $\Delta(n)$ is the neighbourhood width, which is a positive decreasing function of $n$ such that $\Delta(n) \to 0$ as $n \to N$, where $N$ is the total number of time steps. A standard choice is the linear decay:

$$\Delta(n) = \Delta(0) \left( 1 - \frac{n}{N} \right)$$

In the experiments we have used an initial value $\Delta(0) = \text{rows} + \text{cols}/2$ for rows x cols topologies. This ensures that the neighbourhoods are large at the beginning, when the map needs to be plastic.

For brevity we note

$$q_{ai} = q_{ai}(z_i = i) = p_{ai}(z_i = i) = P(z_i = i \mid q_{ai})$$

For each time step $n$ and distribution $q_{ai}$ we have

$$p(t_n \mid q_{ai}) = \sum_{i=1}^{H} q_{ai} p(t_n \mid i)$$

Then, a maximum likelihood approach is used to decide which distribution from $Q_n$ has the most probability $p(q_{ai} \mid t_n)$ of having generated the sample $t_n$. This distribution is called the winning distribution. The prior probabilities $p(q_{ai})$ are assumed equal, i.e., $p(q_{ai}) = 1/H \forall i$ and we have:

$$\text{Winner}(n) = \arg \max_{j} \left[ p(q_{ai} \mid t_n) \right] = \arg \max_{j} \left[ p(t_n \mid q_{ai}) \right]$$

This selection of a distribution from the set of distributions $Q_n$ enforces the self-organization of the network. When the winning distribution $q_{ai}$ has been selected, we may compute the posterior responsibility $R_{ai}$ of mixture component $i$ for generating the observed sample $t_n$ given the set of possible distributions $Q_n$:

$$R_{ai} = P(i \mid t_n, Q_n) = P(i \mid q_{ai}) = q_{ai}$$

where $h = \text{Winner}(n)$ (17)

After this competition, the mixture components are updated according to $R_{ai}$ and $t_n$ in order to build a self-organizing map.

C. Mixture component update

When a new sample $t_n$ has been presented to the network, its information should be used to update the mixture components. If we want to update mixture component $i$ with the information from sample $t_n$, an online version of the original expectation-maximization (EM) method of the PPCA model is required. This possibility has been examined by Sato & Ishii [45] for general PPCA mixtures. Our online EM generates the updated values $\sigma(n)$ and $W(n)$ from the old values $\sigma(n-1)$, $W(n-1)$ and the new sample $t_n$. The application of Robbins-Monro stochastic approximation algorithm yields the following update equations (see Appendix A for details):

$$\pi_i(n) = (1 - \varepsilon(n))\pi_i(n-1) + \varepsilon(n)R_{ai}$$

$$m_i(n) = (1 - \varepsilon(n))m_i(n-1) + \varepsilon(n)R_{ai}t_n$$

$$\mu_i(n) = \frac{m_i(n)}{\pi_i(n)}$$

$$M_i = \sigma_i^2 I + W_i^T W_i$$

$$E[t_n^T \mid i] = (\mu_i - \mu)^T W_i M_i^{-1}$$

$$\Omega_i(n) = (1 - \varepsilon(n))\Omega_i(n-1) + \varepsilon(n)R_{ai}(t_n - \mu_i)E[t_n^T \mid i]$$

$$\Xi_i(n) = (1 - \varepsilon(n))\Xi_i(n-1) + \varepsilon(n)R_{ai}E[t_n^T t_n^T \mid i]$$

$$W_i(n) = \Omega_i(n) \Xi_i(n)^{-1}$$

$$\sigma_i^2(n) = (1 - \varepsilon(n))\sigma_i^2(n-1) + \varepsilon(n)R_{ai}\sigma_i^2(n-1) +$$

$$\varepsilon(n)R_{ai}\left( \frac{1}{\Delta} \left[ \left( t_n - \mu_i \right)^T \right] - 2E[t_n^T \mid i]W_i(n)^T (t_n - \mu_i) +$$

$$\mu_i^T E[t_n^T t_n^T \mid i]W_i(n)^T W_i(n) \right)$$

where $\pi_i(n)$ are the mixing proportions, $m_i(n)$ are not normalized versions of the mean vectors, $\mu_i(n)$ are the mean vectors, $\Omega_i(n)$ and $\Xi_i(n)$ are matrices which are used to build the PPCA matrices $W_i(n)$, and $\sigma_i^2(n)$ are the noise variances. Note that $W_i(n)$ can not be estimated directly by stochastic approximation because it is not obtained in PPCA as an accumulated sum. This is the reason for introducing $\Omega_i(n)$ and $\Xi_i(n)$. Finally, $\varepsilon(n)$ is the step size of the Robbins-Monro algorithm and is typically chosen as

$$\varepsilon(n) = \frac{1}{an + b}, 0 < a < 1$$

In our experiments we have found that selecting $b = \frac{1}{a}$ often yields good results because the network remains plastic long enough. So, we have used in practice

$$\varepsilon(n) = \frac{1}{\varepsilon_0 n + 1}, 0 < \varepsilon_0 < 1$$

where $\varepsilon_0$ is near to zero because higher values produce an excessive plasticity (variability) of the estimations.

Please note that equations (21)-(27) are coupled because they implement the EM iteration. We see in (18)-(27) that the rate at which the new information is incorporated to the model is controlled by the step size $\varepsilon(n)$ and the posterior responsibility $R_{ai}$. 
D. Network initialization

The initialization of the network follows the standard PPCA initialization outlined in [52]. For each neuron \( i \) we select \( K \) samples \( t_n \) and we compute their mean as \( \mu_i(0) \):

\[
\mu_i(0) = \frac{1}{K} \sum_{n=1}^{K} t_n
\]  

(30)

The value of \( K \) is not crucial, provided that it is higher than \( q \) in order to avoid degenerate vector bases. We have used \( K=10q \) in the experiments. Then we choose \( q \) of these samples and compute their differences with \( \mu_i(0) \) to yield the columns of a matrix \( V_i \) of size \( d \times q \). After that we enter an iterative procedure where each column \( v_i \) of \( V_i \) is multiplied by \( S_i(0) \), which does not need to be computed explicitly:

\[
S_i(0) v_i = \sum_{n=1}^{K} (t_n - \mu_i(0))(t_n - \mu_i(0))^T v_i
\]  

(31)

Please note that the above equation can be evaluated in \( O(Kd) \) steps by following the indicated order of evaluation of the matrix products. After the multiplication the obtained vectors are orthonormalized to yield the new value of \( v_i \). The orthonormalization can be accomplished by several methods; in our implementation we have done it by Singular Value Decomposition (SVD). After a few steps this procedure converges. The final eigenvectors are stored into the \( d \times q \) matrix \( U_i \) and the corresponding eigenvalues go to the \( q \times q \) diagonal matrix \( \Lambda_i \). Then we use the following formula for \( \sigma_i(0) \):

\[
\sigma_i(0) = \text{trace}(S_i(0)) - \sum_{j=1}^{q} \lambda_j
\]  

(32)

where the trace of \( S_i(0) \) can be computed in \( O(Kd) \) steps. The starting value of \( W_i(0) \) is obtained as

\[
W_i(0) = U_i(0)(\Lambda_i(0) - (\sigma_i(0)) I)^{1/2}
\]  

(33)

Now we have initial estimations for the parameters of the original PPCA scheme. Finally, the remaining parameters can be computed following (A.10) and (A.19) - (A.21):

\[
\pi_i(0) = \frac{1}{H}
\]  

(34)

\[
m_i(0) = \frac{1}{HK} \sum_{n=1}^{K} t_n = \frac{1}{H} \mu_i(0)
\]  

(35)

\[
\Omega_i(0) = \sum_{j=1}^{K} (t_j - \mu_i(0)) E[x_j^T | i]
\]  

(36)

\[
\Xi_i(0) = \sum_{j=1}^{K} E[x_j x_j^T | i]
\]  

(37)

E. Summary

The PPCASOM model can be summarized as follows:

1. Set the initial values \( \mu_i(0), \sigma_i(0), W_i(0), \pi_i(0), m_i(0), \Omega_i(0) \) and \( \Xi_i(0) \) for all mixture components \( i \) with equations (30) and (32)-(37), respectively.

2. Choose an input sample \( t_n \) and compute the winning distribution by equation (16). Use (17) to compute the posterior responsibilities \( K_i \).

3. For every component \( i \), estimate its parameters \( \pi_i(n), m_i(n) \) and \( \mu_i(n) \) by equations (18)-(20).

4. For every component \( i \), run the EM iteration by repeated application of equations (21)-(27) until the EM method converges, i.e., until no significant changes are made. The values obtained at convergence are the updated values \( \sigma_i(n), W_i(n), \Omega_i(n) \) and \( \Xi_i(n) \).

5. If the map has converged or the maximum time step \( N \) has been reached, stop. Otherwise, go to step 2.

IV. Complexity analysis

Here we analyze the computational complexity of the PPCASOM model. The initialization requires \( O(Kd) \) steps per reduced dimension and neuron, as stated in subsection II.D, where \( K \) is a constant independent of the problem size. Hence, if the network has \( H \) neurons, the initialization of the self-organizing map needs \( O(qdH) \) steps.

Each presentation of an input sample to the network leads to a competition, which is implemented with \( O(q^2dH) \) steps. Then an EM iteration is started in each mixture component, where the equations (21) and (22) are the most computationally expensive, with \( O(q^2d) \) steps. Hence, if we regard the number of EM iterations as a constant, each input sample is processed by the map in \( O(q^2dH) \) steps.

We can conclude that the PPCASOM has linear complexity in the number of neurons \( H \). It is also linear in the size of the input space dimension \( d \): when the latent space dimension \( q \) does not grow with \( d \), anyway the complexity in \( d \) is lower than cubic (note that \( q<<d \)), particularly in typical dimensionality reduction applications with \( q<<d \). This allows to process high dimensional data sets, as we will see in the computational experiments.

V. Discussion

There are some self-organizing models in the literature with the capability of learning local linear subspaces. Now we compare them to the PPCASOM model.

1) The ASSOM model [23] is regarded as a classic example of this kind of self-organizing maps. It has been studied and applied extensively, as seen in the introduction. Nevertheless, it has serious disadvantages: first, it does not consider a mean vector; and second, the update equations need stabilization against ‘spurious’ components in the basis vectors. Furthermore, it does not define any local probability distribution, so the subspaces are ‘crisp’, with no reference to input noise. On the other hand, the PPCASOM learns the mean vectors, it does not need to artificially remove some components of the basis vectors, and it defines local PPCA probability models. In spite of the lack of these features, the ASSOM model does not offer any advantage in computational
complexity, since it processes each training sample in $O(q^2 d H)$ steps. It is most useful in applications which need to map directions instead of data points.

b) The Generative Topographic Mapping (GTM) by Bishop et al. [8] is a constrained mixture of Gaussians where the model parameters are determined by an EM algorithm. The main differences with the PPCASOM are that GTM works in batch mode and that it uses a single latent space where all the local models lie, while PPCASOM builds a local latent space per neuron. Furthermore, PPCASOM offers the possibility of having a topology with different dimensionality than the latent space dimensionality $q$, and even closed topologies (ring, toroidal, etc.), while the GTM is unable to do so. All this allows the PPCASOM to adapt to the input distribution with more flexibility. As before, these disadvantages of GTM are not compensated by a lower complexity, since GTM learns its single global latent space with complexity $O(q^2 d H)$, and PPCASOM learns $H$ local latent spaces in $O(q^2 d H)$. GTM is expected to be preferred if the task at hand needs a unique latent space, while retaining the unsupervised learning capability of self-organizing maps.

c) The Self-Organizing Mixture Model (SOMM) by Verbeek et al. [59] uses a modified version of the EM algorithm to achieve self-organization of Gaussian models with isotropic covariance matrices. The drawbacks of the SOMM are that it only works in batch mode, it is only developed for isotropic covariance matrices, and it has a heavy computational load because it is quadratic in the number of neurons: $O(d H^2)$ for SOMM versus PPCASOM’s $O(d H)$. There is a $O(d H)$ speedup of SOMM, but then it nearly reduces to a classical batch SOM with an isotropic covariance matrix. In this setup, SOMM can be seen as an improvement over SOM which features a parsimonious probabilistic model, since it does not store principal directions.

d) The kernel topographic maps by Van Hulle ([56], [57], [58]) define probability distributions, but they are rather constrained because only diagonal covariance matrices are considered. This is a problem if the local subspaces are not aligned with the input space coordinates, and the situation is worse as the input space dimension grows. On the other hand, PPCASOM does not have those constraints, so its capability to represent complex input distributions is higher. Furthermore, the version with different covariances for each axis [58] has a complexity of $O(d')$ per step, which prevents its use with high dimensional data. Hence, the best suited version for high dimensional data is that with isotropic Gaussians, which we compare with PPCASOM in the experiments.

e) The PCA self-organizing map [30] and the Self-Organizing Mixture Networks model [64] both learn the mean vector and the full covariance matrix at each neuron. Hence, they are more flexible for the representation of complex input data than some of the previous models, which either do not learn the mean vector (ASSOM) or only consider diagonal covariance matrices (SOMM, topographic maps). Nevertheless, their use of the full covariance matrix makes them run in $O(d')$ complexity. Therefore, they are not as suitable as the PPCASOM for high dimensional data.

Finally, it should be noticed that we could have presented the PPCASOM without the use of stochastic approximation, as considered in [31]. Despite the fact that the computational results are similar, the approach considered here has the advantage that the learning rate and the topology of the network are introduced within the stochastic approximation framework, which allows the convergence proof considered in Appendix A.

VI. COMPUTATIONAL RESULTS

A. Data visualization experiments

We have chosen twelve data sets in order to test the self-organization and data visualization capabilities of our proposal. We have selected high-dimensional image and video data. These kinds of data are commonly processed by self-organizing maps in application domains such as image clustering and retrieval ([18], [27], [28], [51]) and video indexing [5]. Furthermore, these experiments show the self-organization capabilities of the PPCASOM model. That is, they show how the units adjust their parameters so that a computational map emerges.

The ‘Faces’ database [50] is composed of 64×64 grayscale images (256 gray levels) which are versions of a computer-generated human face with different poses and lighting directions. This database has been used as a benchmark for high dimensional data visualization ([49], [63]). The ‘Zeros’ to ‘Nines’ databases are composed of 28×28 grayscale images (256 gray levels) of handwritten digits, and come from the MNIST Handwritten Digit Database [29]. These databases have been used as benchmarks for self-organizing maps aimed to high dimensional data processing as ours [38], and they have been also considered for the experimental evaluation of the visualization models mentioned above ([49], [63]). The ‘Video’ database [10] is composed of 64×52 grayscale images (256 gray levels), which have been obtained by reducing original video frames with 352×288 with 24 bits per pixel (RGB color space). In all cases the components of the input vectors are real numbers in the interval [0,1]. We have used no other preprocessing on the original databases available from Internet. The details of each database and the parameter selections for the PPCASOM are shown in Table I.

The choice of the dimension $q$ of the latent subspaces is driven by the dimensionality reduction needs of the application at hand. Lower values of $q$ yield more compact
organizing Mixture Model (SOMM) by Verbeek et al. In order to compare the performance of the PPCASOM hence the mean vectors are nonnegative. Levels because the input values lie in the interval [0,1], and please note that the pictures of the mean vectors are in gray format. We can see that the network self-organizes, and that the mean vectors and eigenvectors capture relevant features of the input data sets. Please note that each pixel corresponds to a different dimension of the input dataset. In order to understand the pictures of the eigenvectors, it is important to remember that the entries of an eigenvector measure the amount of variability (dispersion) in each dimension with respect to the mean vector. The positive elements of the eigenvectors have been drawn in red, and the negative elements in blue (more colour saturation indicates a larger value). Only the changes of sign are relevant, and not the sign itself, because we can negate an eigenvector to obtain other eigenvector pointing in the same direction. The null elements of the eigenvectors are drawn in black, and they correspond to the dimensions with no variability with respect to the mean. Please note that the pictures of the mean vectors are in gray levels because the input values lie in the interval [0,1], and hence the mean vectors are nonnegative.

In order to compare the performance of the PPCASOM model with similar proposals, we have selected the Self-Organizing Mixture Model (SOMM) by Verbeek et al. [59] and the joint entropy maximization kernel based topographic maps (KBTM) by Van Hulle [57]. We have considered the homoskedastic version of Van Hulle’s maps because the heteroskedastic version is $O(d^2)$ per step, which limits its use with the considered databases.

The optimized version of SOMM has been used for the tests. We have simulated the KBTM for 2,000,000 steps, with parameters $\eta_w=0.01$ and $\eta_\sigma=10^{-4} \eta_w$.

Since the three self-organizing models we are comparing define probability distributions, we have considered the average negative log-likelihood ($ANLL$) as the performance measure:

$$ANLL = -\frac{1}{K} \sum_{n=1}^{K} \log p(t_n)$$  

(38)

We have used the T-test to check the statistical significance of the advantage of PPCASOM results over SOMM and KBTM. The difference has been found to be statistically significant for all databases and topologies (less than 0.015 probability that the difference between the means is caused by chance), except for ‘Faces’ with SOMM and 64×1 topology. In this case the probability that the difference between the means is caused by chance is $0.1007$.

Hence our proposal achieves a better self-organized representation of the considered input distributions with a small computational complexity. The $ANLL$ values are expected to be lower (which is better) as the number of units $H$ grows. Hence we have fixed $H=64$ units in all the experiments. The results of the 10-fold cross-validation are shown in Tables II and III, with the standard deviations in parentheses. We have considered two different topologies: 2D rectangular maps with 8×8 units (Table II) and 1D maps with 64×1 units (Table III). We can see that the PPCASOM clearly outperforms SOMM and KBTM in the all the tests.

**Table II**

<table>
<thead>
<tr>
<th>Data set</th>
<th>PPCASOM</th>
<th>SOMM</th>
<th>KBTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zeros</td>
<td>-287.41 (5.28)</td>
<td>-214.50 (3.37)</td>
<td>9.47 (3.42)</td>
</tr>
<tr>
<td>Ones</td>
<td>-927.85 (11.85)</td>
<td>-734.64 (9.92)</td>
<td>-323.48 (10.30)</td>
</tr>
<tr>
<td>Twos</td>
<td>-178.28 (4.89)</td>
<td>-130.41 (3.12)</td>
<td>20.26 (2.76)</td>
</tr>
<tr>
<td>Threes</td>
<td>-235.48 (5.55)</td>
<td>-184.01 (4.65)</td>
<td>-29.84 (2.33)</td>
</tr>
<tr>
<td>Fours</td>
<td>-300.89 (5.18)</td>
<td>-239.84 (7.00)</td>
<td>-69.60 (2.21)</td>
</tr>
<tr>
<td>Fives</td>
<td>-240.67 (5.78)</td>
<td>-190.49 (4.24)</td>
<td>-9.23 (3.77)</td>
</tr>
<tr>
<td>Sixes</td>
<td>-328.44 (12.69)</td>
<td>-253.62 (6.68)</td>
<td>-51.13 (3.30)</td>
</tr>
<tr>
<td>Sevens</td>
<td>-401.25 (8.43)</td>
<td>-315.10 (4.98)</td>
<td>-106.16 (2.71)</td>
</tr>
<tr>
<td>Eights</td>
<td>-211.52 (7.32)</td>
<td>-165.06 (4.48)</td>
<td>-22.61 (5.03)</td>
</tr>
<tr>
<td>Nines</td>
<td>-366.91 (8.82)</td>
<td>-291.92 (5.13)</td>
<td>-96.41 (3.51)</td>
</tr>
<tr>
<td>Faces</td>
<td>-3050.3 (228.38)</td>
<td>-2846.4 (139.52)</td>
<td>-452.28 (65.23)</td>
</tr>
<tr>
<td>Video</td>
<td>-6045.1 (334.23)</td>
<td>-2732.0 (414.70)</td>
<td>-705.06 (59.57)</td>
</tr>
</tbody>
</table>

**Table III**

<table>
<thead>
<tr>
<th>Data set</th>
<th>PPCASOM</th>
<th>SOMM</th>
<th>KBTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zeros</td>
<td>-282.47 (5.24)</td>
<td>-217.00 (4.61)</td>
<td>9.24 (2.93)</td>
</tr>
<tr>
<td>Ones</td>
<td>-927.56 (8.44)</td>
<td>-740.31 (13.69)</td>
<td>-326.66 (4.92)</td>
</tr>
<tr>
<td>Twos</td>
<td>-172.33 (7.40)</td>
<td>-131.39 (5.22)</td>
<td>20.62 (2.07)</td>
</tr>
<tr>
<td>Threes</td>
<td>-230.56 (5.24)</td>
<td>-185.46 (4.11)</td>
<td>-30.03 (2.90)</td>
</tr>
<tr>
<td>Fours</td>
<td>-293.37 (11.30)</td>
<td>-240.80 (6.11)</td>
<td>-69.30 (4.27)</td>
</tr>
<tr>
<td>Fives</td>
<td>-234.45 (6.37)</td>
<td>-192.16 (4.51)</td>
<td>-8.25 (2.27)</td>
</tr>
<tr>
<td>Sixes</td>
<td>-323.62 (5.50)</td>
<td>-256.75 (7.31)</td>
<td>-51.41 (2.74)</td>
</tr>
<tr>
<td>Sevens</td>
<td>-398.73 (8.43)</td>
<td>-317.75 (4.98)</td>
<td>-106.14 (3.99)</td>
</tr>
<tr>
<td>Eights</td>
<td>-206.71 (7.87)</td>
<td>-164.77 (2.47)</td>
<td>-21.89 (3.85)</td>
</tr>
<tr>
<td>Nines</td>
<td>-366.91 (8.82)</td>
<td>-291.92 (5.13)</td>
<td>-96.41 (3.51)</td>
</tr>
<tr>
<td>Faces</td>
<td>-3074.7 (325.7)</td>
<td>-2846.4 (139.52)</td>
<td>-452.28 (65.23)</td>
</tr>
<tr>
<td>Video</td>
<td>-6177.9 (511.5)</td>
<td>-2732.0 (414.70)</td>
<td>-705.06 (59.57)</td>
</tr>
</tbody>
</table>

**TABLE I**

<table>
<thead>
<tr>
<th>Data sets and Parameter Selections</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong># of input samples</strong></td>
</tr>
<tr>
<td>Faces</td>
</tr>
<tr>
<td>Zeros…Nines</td>
</tr>
<tr>
<td>Video</td>
</tr>
</tbody>
</table>

For the final network states are pictured in Figures 1 to 3 for datasets ‘Faces’, ‘Twos’ and ‘Video’, respectively. We have plotted the mean vectors and the first eigenvectors in image format. We can see that the network self-organizes, and that the mean vectors and eigenvectors capture relevant features of the input data sets. Please note that each pixel corresponds to a different dimension of the input dataset. In order to understand the pictures of the eigenvectors, it is important to remember that the entries of an eigenvector measure the amount of variability (dispersion) in each dimension with respect to the mean vector. The positive elements of the eigenvectors have been drawn in red, and the negative elements in blue (more colour saturation indicates a larger value). Only the changes of sign are relevant, and not the sign itself, because we can negate an eigenvector to obtain other eigenvector pointing in the same direction. The null elements of the eigenvectors are drawn in black, and they correspond to the dimensions with no variability with respect to the mean. Please note that the pictures of the mean vectors are in gray levels because the input values lie in the interval [0,1], and hence the mean vectors are nonnegative.

In order to compare the performance of the PPCASOM model with similar proposals, we have selected the Self-Organizing Mixture Model (SOMM) by Verbeek et al. [59] and the joint entropy maximization kernel based topographic maps (KBTM) by Van Hulle [57]. We have considered the homoskedastic version of Van Hulle’s maps because the heteroskedastic version is $O(d^2)$ per step, which limits its use with the considered databases.

The optimized version of SOMM has been used for the tests. We have simulated the KBTM for 2,000,000 steps, with parameters $\eta_w=0.01$, $\eta_\sigma=10^{-4} \eta_w$.

Since the three self-organizing models we are comparing define probability distributions, we have considered the average negative log-likelihood ($ANLL$) as the performance measure:

$$ANLL = -\frac{1}{K} \sum_{n=1}^{K} \log p(t_n)$$  

(38)
Fig. 1. Results for the ‘Faces’ database: mean vectors (top) and first eigenvectors (bottom).

Fig. 2. Results for the ‘Twos’ database. From left to right and from top to down: mean vectors, first eigenvectors, second eigenvectors and third eigenvectors.
is used to obtain the reduced version of

The optimal reconstructed vector is

where the reconstruction vector

Some recent proposals are [40], [6], [2], [3] and [48]. As the PPCASOM is a linear subspace model, it is
natural to represent the input data by its projection on the orthonormal vector basis of each unit. The unit $j$ which yields
the least reconstruction error of an input sample $\mathbf{t}_n$ is given by

$$ j = \arg \min_i \| \tilde{\mathbf{t}}_n^i - \mathbf{t}_n \|^2 $$

(39)

where the reconstruction vector $\tilde{\mathbf{t}}_n^j$ is obtained with (9). Note
that we have dropped the subindex $n$ in $j$ for clarity. This unit
is used to obtain the reduced version of $\mathbf{t}_n$:

$$ \mathbf{z}_n = \mathbf{U}_j \mathbf{t}_n - \mathbf{u}_j $$

(40)

where the reduced (projected) vector $\mathbf{z}_n$ has dimensions $q \times 1$.

The optimal reconstructed vector is

$$ \tilde{\mathbf{t}}_n = \tilde{\mathbf{t}}_n^j $$

(41)

with $j$ given by (39). For conventional PCA, a similar
approach is followed, but with only one global vector base $\mathbf{U}_j$.

For the KBTM and SOMM models, each input sample $\mathbf{t}_n$ is
represented by the mean vector of the unit which is closest (in
the mean squared error sense) to $\mathbf{t}_n$.

In the image compression context, it is convenient to split
the original image into equally sized windows, so that each
window is an input vector. Here we have used windows of
8x8 pixels, which is a common choice in many cases such as
JPEG [60]. This corresponds to an input space dimension of
$d=64$. Smaller window sizes result in poor compression ratios,
while larger windows compromise image quality. On the other
hand, each component of the input vectors is an integer value
in the range [0,255], as we consider grayscale images with 8
bits of precision. Since each unit of the three considered self-
orGANizing models stores an approximation of the mean
vector, for these cases we need to store the following data in
the compressed file:

a) The mean vectors for each unit. First they are rounded to
8 bits, and then they are Huffman encoded to remove the
redundancy [17].

b) The index of the best matching unit for each window of
the original image. These integers are represented with
the least possible number of bits, and then Huffman encoded.

For conventional PCA, we only need to store the global
mean. Please note that the encoding has been exactly the same
in the four cases in order to make a fair comparison. Only in
the PPCASOM and PCA cases we need to store additional
data:

a) The orthogonal vector bases $\mathbf{U}_j$ for each unit are
represented with 32 bit floating point numbers. We make use
of the fact that these real numbers are in the range $[-1,1]$, since they are part of unit norm vectors.

b) The components of the projected vectors $\mathbf{z}_n$ are quantized
with a variable number of bits (from 1 to 10 bits) and run
length encoding of the sequences of zero values. The
quantization procedure involves the division of each
component of $\mathbf{z}_n$ by an integer constant, the rounding of the
obtained quotient, and the codification of the resulting integer
with the desired number of bits. Finally, the resulting string of
bits is Huffman encoded. We have tuned the quality of the
compression by choosing smaller integer constants to obtain
more quality, and vice versa.

We have divided the input data randomly into two disjoint
subsets: the training set, with 90% of the windows, and the
test set, with the remaining 10%. The data compression
performance has been evaluated by two measures. First, the
mean squared error per pixel has been computed for the test set:

$$ MSE = \frac{1}{64 \cdot N_{\text{Test}}} \sum_{n=1}^{N_{\text{Test}}} \| \mathbf{t}_n - \mathbf{i}_n \|^2 $$

(42)

where $N_{\text{Test}}$ is the number of windows of the test set, and $\mathbf{i}_n$ is the optimal reconstructed vector. For the KBTM and the
SOMM this is equal to the best matching mean vector, in the
mean squared error sense. For the PPCASOM and PCA, it is
computed with (41); for conventional PCA, we do not need to
choose among units because there is only one vector basis. On
the other hand, the size of the compressed file has been
considered, expressed as number of bits per pixel:

$$ BPP = \frac{\text{FileSize}}{N_{\text{Pix}}} $$

(43)

where FileSize is the size of the compressed file in bits, and
$N_{\text{Pix}}$ is the number of pixels of the image.

The parameter selection for the three models has been the
same as in the previous subsection, except for the size of the
maps, which has been varied in order to tune the quality of the
compression. As the maps are larger, the quality of the

Fig. 3. Results for the ‘Video’ database: mean vectors (top), first eigenvectors (middle) and second eigenvectors (bottom).
reconstructed image is better, but there are more storage requirements. For KBTM and SOMM we have experimented with maps with sizes in the range $5 \times 5$ to $50 \times 50$ units. For the PPCASOM, since each unit has more compression capability, we have used smaller maps: $2 \times 2$ to $5 \times 5$ units, with sizes of the vector bases in the range $q = 1$ to 60. In the three cases, only the results for the best performing model sizes have been shown in the plots. For conventional PCA, we have tested basis vector sizes in the range $q = 1$ to 60.

![Original images: 'lena' (top) and 'monarch' (bottom).](image1)

We can see in Figure 4 two benchmark images taken from the University of Waterloo Repertoire [54]. The ‘lena’ image has $512 \times 512$ pixels, and ‘monarch’ has $768 \times 512$ pixels. The plots of MSE versus Bits per pixel are in Figures 5 and 6. Please note that every model yields its own Bits per pixel values, so they can not be forced to produce the exact same BPP. The results show that our proposal achieves lower error at all bit rates in both of the tested images. It is worth of mention that the KBTM model is not able to learn correctly when the map size is too large for the variability of the input dataset. This is the reason because the KBTM data series have different lengths in the plots. On the other hand, Figures 7 and 8 show that PPCASOM yields better visual image quality with less bit rate. We can conclude that PPCASOM is able to take advantage of its linear subspace self-organization ability to achieve efficient dimensionality reduction.

![Fig. 5. MSE versus bits per pixel (BPP) for the ‘lena’ image. Note that nearer to the coordinate origin is better.](image2)

![Fig. 6. MSE versus bits per pixel (BPP) for the ‘monarch’ image. Note that nearer to the coordinate origin is better.](image3)
C. Video compression experiments

The third set of experiments studies the capability of our proposal to compress video data. Self-organizing maps have been proposed recently as vector quantizers for video compression ([41], [14]). Here we evaluate the potential of PPCASOM, KBTM and SOMM for this purpose. Additionally, we include conventional PCA results for comparison with a linear technique, as in the previous subsection.

Lossy compression of video data is commonly performed by dividing the frame (picture) sequence into small Groups of Pictures (GOPs). Each GOP contains three types of frames [46]:

a) Intra frames (I-Frames) are complete images. They are compressed completely independent of other frames. Hence the coding of I-Frames is a standard lossy image compression problem. Typically, the first frame of a GOP is an I-Frame.

b) Predicted frames (P-Frames) are images within the GOP that are created using information from preceding images. Typically, the difference between the P-Frame and the first frame of the GOP (which is an I-Frame) is lossy compressed.

c) Bidirectional frames (B-Frames) are images within the GOP that are created using information from preceding images and images that follow. The usage of B-Frames needs the evaluation of which other images will do the best to approximate the B-Frame. Since this is a complex issue in its own, we are not using B-Frames in these experiments in order to avoid side effects which could make the comparison unfair.

Additionally, each image is divided into small blocks of pixels in order to perform the lossy compression. In our experiments, we have considered a GOP length of 12 frames, i.e., we have the first I-Frame and 11 following P-Frames which are coded as the differences with respect to the I-Frame. In each frame we have used blocks of $8 \times 8$ pixels, and we have compressed only the Y component (luminance), which is given as an integer in the range $[0,255]$. Both choices (GOP length and block size) are typical in MPEG, for example [46]. This means that we have input vectors of dimension $d=64$ and two self-organizing maps to be trained for each GOP: one for the I-Frame and the other for the 11 difference images of the P-Frames. We have divided the input data for each self-organizing map randomly into two disjoint subsets: the training set, with 90% of the blocks, and the test set, with the remaining 10%. The encoding of the self-organizing maps have been the same as in the previous subsection.

The evaluation of the results has been similar to the image compression experiments. First, the mean squared error per pixel has been computed for the test sets, with equation (42). On the other hand, the size of the compressed file has been considered, in this case expressed as Kbits per second of video:

$$Kbps = \frac{FileSize}{NSec} \quad (44)$$

where $FileSize$ is the size of the compressed file in Kbits, and $NSec$ is the duration of the video sequence in seconds.

We have selected two freely available benchmark video sequences from the Xiph.org Test Media [62]. They are commonly used for video compression evaluation ([16], [34], [21], [4], [7]). The original sequences are in the uncompressed Common Interchange Format (CIF), with frames of $352 \times 288$ pixels. The parameter selection for the three models has been
the same as in the previous subsection. For KBTM and SOMM we have experimented with maps sizes in the range $2 \times 2$ to $30 \times 30$ units. For the PPCASOM we have used smaller maps because each unit has more compression capability: $2 \times 2$ to $5 \times 5$ units, with sizes of the vector bases in the range $q=1$ to $60$. In the three cases, only the results for the best performing model sizes have been shown in the plots. The same range of vector base sizes ($q=1$ to $60$) has been used for conventional PCA.

The plots of MSE versus Kbits per second are in Figures 9 and 10. As every model yields its own Kbits per second values, they can not be forced to produce the exact same Kbps. It can be seen that our proposal achieves lower error at all bit rates in both of the tested videos. The KBTM data series end sooner for the same reason as in the image compression experiments, i.e., it does not learn correctly with larger map sizes. On the other hand, Figures 11 and 12 show that PPCASOM yields better visual quality with less bit rate. As in the image compression experiments, these results indicate that PPCASOM is suitable for dimensionality reduction, due to its linear subspace self-organization ability.

Fig. 9. MSE versus KBits per second (Kbps) for the ‘container’ video. Note that nearer to the coordinate origin is better.

Fig. 10. MSE versus KBits per second (Kbps) for the ‘paris’ video. Note that nearer to the coordinate origin is better.

Fig. 11. Detail of the 51st frame of ‘container’. From left to right and from top to bottom: original (20.2752 Mbps), PPCASOM compressed (3.0714 Mbps), SOMM compressed (3.2522 Mbps), KBTM compressed (1.4188 Mbps), PCA compressed (3.1436 Mbps).
mixtures of homoskedastic Gaussians. Hence, our model achieves both scalability and a correct representation of the input distribution.

APPENDIX A: STOCHASTIC APPROXIMATION

First we consider the original (batch) M-step equations for \( W \) and \( \sigma \) at time step \( n \) (see [53]):

\[
\pi_i = \frac{1}{n} \sum_{j=1}^{n} R_{ij}
\]

(A.1)

\[
\mu_i = \frac{1}{\sum_{j=1}^{n} R_{ij}} \sum_{j=1}^{n} R_{ij} t_j
\]

(A.2)

\[
W_i = \left[ \sum_{j=1}^{n} R_{ij} (t_j - \mu_i) E[x_j^T | i] \right] \left[ \sum_{j=1}^{n} R_{ij} E[x_j x_j^T | i] \right]^{-1}
\]

(A.3)

\[
\sigma_i^2 = \frac{1}{nd} \sum_{j=1}^{n} R_{ij} \left( \| x_j - \mu_i \|^2 - 2E[x_j x_j^T | i] W_i^T (t_j - \mu_i) + tr\left[ E[x_j x_j^T | i] W_i^T W_i \right] \right)
\]

(A.4)

where the corresponding E-step equations are:

\[
E[x_j^T | i] = (t_j - \mu_i) W_i M_i^{-1}
\]

(A.5)

\[
E[x_j x_j^T | i] = \sigma_i^2 M_i^{-1} + \left( E[x_j^T | i] W_i^T \right) E[x_j x_j^T | i]
\]

(A.6)

\[
M_i = \sigma_i^2 I + W_i^T W_i
\]

(A.7)

Several weighted means appear implicitly in equations (A.1) to (A.4), where the weights are the responsibilities \( R_{ij} \). Let \( \Theta=(\mu, W_i, \sigma) \) be a vector comprising the parameters for mixture component \( i \), and let \( \phi(\Theta, t) \) be an arbitrary function of \( \Theta \) and the input sample \( t \). Then we define the weighted mean of \( \phi(\Theta, t) \) as:

\[
\langle \phi \rangle_i = E[P(i | t) \phi(\Theta_i, t)]
\]

(A.8)

If we have \( n \) samples (finite case), the linear least squares approximation for \( \langle \phi \rangle_i \) is:

\[
\langle \phi \rangle_i = \frac{1}{n} \sum_{j=1}^{n} P(i | t_j) \phi(\Theta_j, t_j) = \frac{1}{n} \sum_{j=1}^{n} R_{ij} \phi(\Theta_i, t_j)
\]

(A.9)

As \( n \to \infty \), the approximation of (A.9) converges to the exact value given by (A.8). In this case we can rewrite equations (A.1) to (A.4) in terms of the weighted means \( \langle \phi \rangle_i \):

\[
\pi_i = \langle 1 \rangle_i
\]

(A.10)

\[
\mu_i = \langle 1 \rangle_i
\]

(A.11)

\[
W_i = \left( (t_i - \mu_i) E[x_i^T | i] \right) \left( E[x x^T | i] \right)^{-1}
\]

(A.12)

\[
\sigma_i^2 = \langle t_i - \mu_i \rangle^2 - 2E[x x^T | i] W_i^T (t_i - \mu_i) +
\]

VII. CONCLUSIONS

We have presented a probabilistic self-organizing neural model, which features online learning of the local principal subspaces of the input data. It is based on a mixture of Gaussians where only a certain number \( q \) of relevant principal directions is considered. It is particularly suited for high dimensional data because it has a low computational complexity. Experimental results have been presented that show the self-organization capabilities of the model and its performance in image and video compression applications. In particular, it outperforms two self-organizing maps based on
On the other hand, we can apply the Robbins-Monro stochastic approximation algorithm (see [26]) to estimate iteratively the value of the weighted means \( \langle \varphi \rangle \): 

\[
\langle \varphi \rangle (0) = P(i \mid t_0) \rho(\bm{\Theta}, t_0)
\]

(\ref{eqn:robbins_monro_0})

(\ref{eqn:robbins_monro_1})

where \( \varepsilon(n) \) is the step size, which must satisfy the following conditions in order to guarantee convergence of the Robbins-Monro method:

\[
\varepsilon(n) > 0, \quad \lim_{n \to \infty} \varepsilon(n) = 0, \quad \sum_{n=1}^{\infty} \varepsilon^2(n) < \infty
\]

(\ref{eqn:robbins_monro_2})

In order to fulfill these requirements, \( \varepsilon(n) \) is typically selected as

\[
\varepsilon(n) = \frac{1}{an+b}, \quad 0 < a < 1
\]

(\ref{eqn:robbins_monro_3})

On the other hand, equation (A.15) is more conveniently written as:

\[
\langle \varphi \rangle (n) = \langle \varphi \rangle (n-1) + \varepsilon(n) R_{ni} \varphi(\Theta_n, t_n)
\]

(\ref{eqn:robbins_monro_4})

Now we derive an online EM algorithm by applying equation (A.18) to equations (A.10)-(A.13). First we define three auxiliary variables:

\[
\Omega_i = \langle t - \mu_i \rangle \mathbb{E} \left[ x^T \right]_i
\]

(\ref{eqn:auxiliary_1})

\[
\Xi_i = \mathbb{E} \left[ x^T x^T \right]_i
\]

(\ref{eqn:auxiliary_2})

\[
\mathbf{m}_i = \langle t_i \rangle
\]

(\ref{eqn:auxiliary_3})

The corresponding update equations are:

\[
\Omega_i (n) = (1 - \varepsilon(n)) \Omega_i (n-1) + \varepsilon(n) R_{ni} (t_n - \mu_i) \mathbb{E} \left[ x^T \right]_i
\]

(\ref{eqn:auxiliary_4})

\[
\Xi_i (n) = (1 - \varepsilon(n)) \Xi_i (n-1) + \varepsilon(n) R_{ni} \mathbb{E} \left[ x^T x^T \right]_i
\]

(\ref{eqn:auxiliary_5})

\[
\mathbf{m}_i (n) = (1 - \varepsilon(n)) \mathbf{m}_i (n-1) + \varepsilon(n) R_{ni} t_n
\]

(\ref{eqn:auxiliary_6})

Then we are ready to rewrite (A.10)-(A.13):

\[
\pi_i (n) = (1 - \varepsilon(n)) \pi_i (n-1) + \varepsilon(n) R_{ni}
\]

(\ref{eqn:auxiliary_7})

\[
\mu_1 (n) = \frac{\mathbf{m}_1 (n)}{\pi_1 (n)}
\]

(\ref{eqn:auxiliary_8})

\[
W_i (n) = \Omega_i (n) \Xi_i (n)^{-1}
\]

(\ref{eqn:auxiliary_9})

\[
\alpha_i^2 (n) = (1 - \varepsilon(n)) \alpha_i^2 (n-1) + \varepsilon(n) R_{ni} \frac{1}{2} \mathbf{m}_i (n) - \mu_i \right)^T - 2 \mathbb{E} \left[ x^T \right]_i \mathbf{m}_i (n) + \mathbf{m}_i (n)^T \mathbb{E} \left[ x^T x^T \right]_i W_i (n) \mathbf{m}_i (n)
\]

(\ref{eqn:auxiliary_10})

Proposition: If (A.16) holds, then the stochastic approximation algorithm of (A.25)-(A.28) converges to a maximum of the likelihood.

Proof:

The general form of the Robbins-Monro stochastic algorithm is

\[
\theta(n+1) = \theta(n) + \varepsilon(n) Y(n)
\]

(\ref{eqn:stoch_1})

where, in our case we take

\[
\theta(n) = \langle \varphi \rangle (n) \left| \frac{\partial}{\partial \theta} \right|_{\theta(n)}
\]

(\ref{eqn:stoch_2})

That is, we include all the weighted means in the current estimation vector \( \theta(n) \). We also take

\[
Y(n) = \xi(n) - \theta(n)
\]

(\ref{eqn:stoch_3})

where the new data to be incorporated into the estimation is

\[
\xi(n) = \langle P(i \mid t_n, \theta(n), Q_n) \varphi(\Theta_n, t_n) \rangle \left| \frac{\partial}{\partial \theta} \right|_{\theta(n)}
\]

(\ref{eqn:stoch_4})

In the above equation, \( \Theta = \mu_k, \ \mathbf{W}_n, \ \sigma_k \) is the complete parameter vector of the self-organizing map, and \( Q_n \) is the distribution set which is used to enforce self-organization at time step \( n \). The goal of the stochastic algorithm is to find a root of the equation

\[
g(\theta(n)) = 0
\]

(\ref{eqn:stoch_5})

and \( Q_n \) is the limit distribution set,

\[
q_{kj} = 1, \quad q_{kj} = 0 \forall i \neq j
\]

(\ref{eqn:stoch_6})

which is such that the \( Q_n \) tend to \( Q_\infty \) as \( n \to \infty ;

\[
\lim_{n \to \infty} q_{kj} = q_{kj} \forall i, j
\]

(\ref{eqn:stoch_7})

In order to prove the convergence of the algorithm, we are going to prove that the "noise" in the observation \( Y_n \) is a martingale difference (see [26]). That is, there is a function \( g_\theta(\cdot) \) of \( \theta \) such that

\[
E[Y_n \mid \theta(0), Y_j, j < n] = g_\theta(\theta(n))
\]

(\ref{eqn:stoch_8})

This is readily verified:

\[
g_\theta(\theta(n)) = (E[P(i \mid t, \Theta_0, Q_\infty) \varphi(\Theta_i, t)] - \langle \varphi \rangle (n)) \left| \frac{\partial}{\partial \theta} \right|_{\theta(n)}\]

(\ref{eqn:stoch_9})

Note that \( g_\theta(\theta(n)) \) only depends on \( n \) and \( \theta(n) \), because \( Q_n \) is obtained from \( n \) and \( \theta(n) \) can be computed from \( \theta(n) \). So we have that

\[
y_n = g_\theta(\theta(n)) + \delta M_n
\]

(\ref{eqn:stoch_10})

where \( \delta M_n \) is a martingale difference. The associated "bias" process is defined as:

\[
\beta_n = E[Y_n \mid \theta(0), Y_j, j < n] - g(\theta(0))
\]

(\ref{eqn:stoch_11})

We can guarantee convergence by proving the following three conditions:

\[
\sup_n E[|Y_n|^2] < \infty
\]

(\ref{eqn:stoch_12})

\[
\lim_{n \to \infty} \beta_n = 0
\]

(\ref{eqn:stoch_13})

\[
\forall m > 0 \forall \theta, \lim_{n \to \infty} \sum_{i=1}^{m} E[|g_\theta(\theta) - g(\theta)|] = 0
\]

(\ref{eqn:stoch_14})

Next we examine the first condition:

\[
E[|y_n|^2] = E[|\xi(n) - \theta(n)|^2] = \sum_{\theta \in Q_n} E[P(i \mid t, \theta(n), Q_n) \varphi(\Theta_i, t)] - \langle \varphi \rangle (n)^2
\]

(\ref{eqn:stoch_15})

The above value is finite if we assume that the input distribution has a compact support. This assumption should
hold in practice, since real data always appear in a finite domain. Alternatively we can relax this assumption by considering that the input density decreases exponentially as $t \to \infty$.

Now we study the second condition:
\[
\lim_{n \to \infty} \beta_n = \lim_{n \to \infty} \left[ E[Y_n \mid \Theta(0), Y_j, j < n] - \mathcal{P}(\Theta(n)) \right] = \\
\lim_{n \to \infty} \left[ E[P(i \mid t, \Theta(n), Q_i) \mathcal{P}(\Theta(n), t)] - \mathbb{E}[P(i \mid t, \Theta(n), Q_i) \mathcal{P}(\Theta(n), t)]_{q_{i,j}} \right] (A.45)
\]
This limit is zero because (A.36) implies that
\[
\lim_{n \to \infty} P(i \mid t, \Theta(n), Q_n) = P(i \mid t, \Theta(n), Q_n). (A.46)
\]
Finally, for the third condition we have
\[
\forall \Theta, \lim_{j \to \infty} \left[ g_j(\Theta) - g(\Theta) \right] = \\
\lim_{j \to \infty} \left[ E[P(i \mid t, \Theta, Q) \mathcal{P}(\Theta, t)] - \mathbb{E}[P(i \mid t, \Theta, Q) \mathcal{P}(\Theta, t)]_{q_{i,j}} \right] (A.47)
\]
which is also zero because of (A.36). In turn this means that the limit
\[
\lim_{n \to \infty} \sum_{j=1}^{m+1} e_j \left[ g_j(\Theta) - g(\Theta) \right] (A.48)
\]
is zero $\forall m > 0, t$ since it is the norm of a finite sum of two factors where both tend to zero.

Hence we have proved that the algorithm converges to a root of equation (A.33). At convergence we have
\[
\mathcal{P}(\Theta) = E[P(i \mid t, \Theta, Q) \mathcal{P}(\Theta, t)] (A.49)
\]
which is equivalent to the maximum likelihood condition
\[
\frac{\partial L}{\partial \Theta} = 0 (A.50)
\]
where
\[
L(\Theta) = E[\log P(t \mid \Theta, Q)] (A.51)
\]
because (A.49) is a fixed point of (A.10)-(A.13).

ACKNOWLEDGEMENTS

The authors wish to thank the associate editor and the anonymous reviewers for their valuable comments and suggestions.

REFERENCES


Ezequiel López-Rubio (born 1976) received his MSc and PhD (honors) degrees in Computer Engineering from the University of Málaga, Spain, in 1999 and 2002, respectively. He joined the Department of Computer Languages and Computer Science, University of Málaga, in 2000, where he is currently an Associate Professor of Computer Science and Artificial Intelligence. His technical interests are in unsupervised learning, pattern recognition and image processing.

Juan Miguel Ortiz-de-Lazcano-Lobato (born 1978) obtained his M. Tech. degree in Computer Science from the University of Málaga (Spain) in 2003. Then he received his PhD degree in Software Engineering and Artificial Intelligence from the University of Málaga in 2007. Since 2008, he has been working as an Associate Professor in the University of Málaga. His research interests include fields such as multidimensional data analysis, pattern recognition and neural networks, especially competitive and self-organizing neural networks.
Domingo López-Rodríguez (born 1978) received his PhD degree in Applied Mathematics from the University of Málaga, Spain, in 2007.

Since 2003, he has been working as a researcher with the Soft Computing and Image Analysis Unit at the University of Málaga. His research interests include fields such as optimization, neural networks and combinatorial optimization problems. He is particularly interested in the application of multivalued recurrent networks to solve optimization problems on graphs, and in the development of a general framework to study competitive learning. Web-based education and distance learning are also fields of his study.