Self-Organizing Mixture Models

J.J. Verbeek *, N. Vlassis, B.J.A. Kröse

University of Amsterdam, Faculty of Science, Informatics Institute, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

Abstract

We present an expectation-maximization (EM) algorithm that yields topology preserving maps of data based on probabilistic mixture models. Our approach is applicable to any mixture model for which we have a normal EM algorithm. Compared to other mixture model approaches to self-organizing maps, the function our algorithm maximizes has a clear interpretation: it sums data log-likelihood and a penalty term that enforces self-organization. Our approach allows principled handling of missing data and learning of mixtures of self-organizing maps. We present example applications illustrating our approach for continuous, discrete, and mixed discrete and continuous data.

Key words: self-organizing maps, mixture models, EM algorithm.

1 Introduction

The self-organizing map, or SOM for short, was introduced by Kohonen in the early 1980's and it combines clustering of data with topology preservation. The clusters found in the data are represented on a, typically two dimensional, grid, such that clusters with similar content are nearby in the grid. The representation thus preserves topology in the sense that it keeps similar cluster nearby. The SOM allows one to visualize high dimensional data in two dimensions, e.g. on a computer screen, via a projection that may be a non-linear function of the original features in which the data was given. The clusters are sometimes also referred to as 'nodes', 'neurons' or 'prototypes'. Since their introduction, self-organizing maps have been applied in many engineering problems, see e.g. the list of over 3000 references on applications of SOM in [15]. Some examples

* Corresponding author.

Email addresses: jverbeek@science.uva.nl (J.J. Verbeek), vlassis@science.uva.nl (N. Vlassis), krose@science.uva.nl (B.J.A. Kröse).

of applications include visualization and browsing of large image and document databases, where similar database items are projected to nearby neurons of the SOM.

Probabilistic mixture models are densities (or distributions) that can be written as a weighted sum of component densities, where the weighting factors are all non-negative and sum to one [19]. By using weighted combinations of simple component densities a rich class of densities is obtained. Mixture models are used in a wide range of settings in machine-learning and pattern recognition. Examples include classification, regression, clustering, data visualization, dimension reduction, etc. A mixture model can be interpreted as a model that assumes that there are, say k, sources that generate the data: each source is selected to generate data with a probability equal to its mixing weight and it generates data according to its component density. Marginalizing over the components, we recover the mixture model as the distribution over the data. With a mixture model we can associate a clustering of the data, by assigning each data item to the source that is most likely to have generated the data item. The expectation-maximization (EM) algorithm [7] is a simple and popular algorithm to fit the parameters of a mixture to given data.

Several variations of the original SOM algorithm have been proposed in the literature, they can be roughly divided into two groups. First, different divergence measures have been proposed, suitable to assign data points to clusters when using different types of data. Second, alternative learning algorithms for SOMs have been proposed. In this paper we show how to combine the benefits of self-organizing maps and mixture models. We present a general learning algorithm, similar to Kohonen's original SOM algorithm, that can, in principle, be applied to any probabilistic mixture model. The algorithm is the standard EM learning algorithm using a slightly modified expectation-step. Our contribution can be considered as one in the category of SOM papers presenting new learning algorithms. However, since our modified EM algorithm can be applied to any mixture model for which we have a normal EM algorithm, it can be applied to a wide range of data types. Prior knowledge or assumptions about the data can be reflected by choosing an appropriate mixture model. The mixture model will, implicitly, provide the corresponding divergence measure. We can thus state that our work also gives a contribution of the first type: it helps to design divergence measures implicitly by specifying a generative model for the data.

The main merits of our work are the following. First, we have a SOM-like algorithm with the advantages of probabilistic models, like principled handling of missing data values, the ability to learn mixtures of self-organizing maps, etc. Second, compared to other mixture model like approaches to self-organizing maps, the objective function the algorithm optimizes has a clear interpretation: it sums the data log-likelihood and a penalty term which enforces the

topology preservation. Finally, since we merely modify the expectation-step, we can directly make a self-organizing map version of any mixture model for which we have a normal EM algorithm. We only need to replace normal E-step with the modified E-step presented here to obtain a self-organizing map version of the given mixture model.

The rest of the paper is organized as follows: first we briefly review self-organizing maps and motivate our work. Then, in Section 3, we review the EM algorithm. In Section 4 we show how a modified E-step can be used to enforce self-organization in any mixture model. Two extensions of the basic idea are presented in Section 5. Related work on self-organizing maps is discussed in Section 6. In Section 7 we provide examples of applying the modified EM algorithm to several different mixture models. Conclusions are provided in the last section.

2 Self-Organizing Maps

Self-organizing maps are used for different purposes, however their main application is data visualization. If the data are high dimensional or non-numerical a self-organizing map can be used to represent the data in a two-dimensional space, often called 'latent' space. Nearby locations in the latent space represent similar data. For example, the self-organizing map can be used to represent images in a two-dimensional space, such that similar images are nearby. This latent representation can then be used to browse a large image data-base.

Another application of SOMs to image data is the PICSOM system [1] [18] for content-based image retrieval. The system provides a search mechanism that allows a user to find images of interest through several interactions with the system. First, the PICSOM system presents some random images from the data-base to the user and then the user identifies relevant images: i.e. images that are (most) similar to the target. Alternatively, the user could directly provide some relevant images to the system. The system uses a self-organizing map to find similar pictures and presents those to the user. The process can then be iterated by letting the user add new images (from the presented ones) to the set of relevant images. The system uses several different representations of the images, among others color and texture content are used. As the user adds more images to the set of relevant images, it may become apparent that they are all similar in respect to their texture content but not in their color content, the system identifies that mainly texture content matters and will present new images mainly on the basis of similar texture content.

[1] An online demo of the system can be found at `http://www.cis.hut.fi/picsom/`.

The basic self-organizing map (SOM) algorithm assumes that the data are given as vectors. The algorithm then fits a set of reference vectors or prototypes to the data. We use the variable s throughout to index the prototypes. With each prototype s a location in the latent space $\mathbf{g}_s$ is associated. The latent space is typically two-dimensional in visualization applications of the SOM. The prototypes are fitted in such a way that nearby prototypes in the latent space will also be nearby in the data space. There are two ways to fit the prototypes, we can process data items one-by-one, referred to as online, and all-at-once, referred to as batch. The online algorithm is useful when the data items cannot be stored because that might take too much storage.

Both batch and online algorithm make use of a so called 'neighborhood' function that will encode the spatial relationships of the prototypes in the latent space. The neighborhood function is a function that maps the indices of two prototypes to a real number. Typically the neighborhood function is a positive function which decreases as the distance between two prototypes in the latent space increases. Often, the exponent of the negative squared distance between the two prototypes in the latent space is used as neighborhood function, i.e. a Gaussian centered at one prototype evaluated at the other. In that case, having fixed the locations of the prototypes in the latent space at $\mathbf{g}_s$ for prototype s, the neighborhood function evaluated at prototypes r and s is given by:

$$h_{rs} = \exp(-\lambda \parallel \mathbf{g}_r - \mathbf{g}_s \parallel^2),$$

where λ controls the width of the neighborhood function. Sometimes, the neighborhood function is considered a function of one argument fixing one of the prototypes. We will use the following notation to reflect this:

$$h_r(s) = \exp(-\lambda \parallel \mathbf{g}_r - \mathbf{g}_s \parallel^2), \tag{1}$$

The online SOM training algorithm can be summarized as follows:

Basic online SOM algorithm
Initialize the prototype vectors as randomly selected data points.
Iterate these steps:

- **Assignment:** Assign the current data item to its nearest (typically in terms of Euclidean distance) prototype r^* in the data space, this prototype is called the 'winner'.
- **Update:** Move the each prototype s toward the data vector by an amount proportional to $h_{r^*}(s)$: the value of the neighborhood function evaluated at the winner r^* and prototype s.

Thus a prototype is moved toward the data that is assigned to it, but also

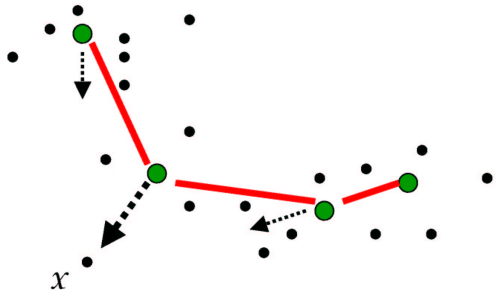

Fig. 1. The update of the online algorithm. The small dots are data points and the big dots are the prototype vectors. The ordering of the prototypes in the latent space is indicated by the bars connecting them. The size of the arrows indicates the force by which the nodes are attracted toward the point x.

toward data that is assigned to prototypes nearby in the latent space. Fig. 1 illustrates the update for a point indicated with x. Alternatively, the prototypes can be initialized by linearly mapping the $\mathbf{g}_s$ to the subspace spanned by the principal components of the data, such that the covariance of the mapped $\mathbf{g}_s$ matches the covariance of the data in the principal components subspace.

The batch version of the SOM algorithm processes all data at once in each step by first doing the assignment for all data vectors. In the update step, each prototype s is then placed at the weighted average of all data, where each data vector is weighted by a factor proportional to the neighborhood function evaluated at s and the winner for that data vector. If we use $\boldsymbol{\mu}_r$ to denote prototype r and s_n to indicate the winner for data point $\mathbf{x}_n$, the batch algorithm sets:

$$\boldsymbol{\mu}_r = \frac{\sum_{n=1}^{N} h_{s_n}(r)\mathbf{x}_n}{\sum_{n=1}^{N} h_{s_n}(r)}.$$

The Euclidean distance used in the assignment step of the SOM algorithm is not always appropriate, or even not applicable, e.g. for data that is not expressed as real-valued vectors. For example, consider data that consists of vectors with binary entries. Then, we might still use the Euclidean distance, but how will we represent the prototypes? As binary vectors as well? Or should we allow for scalars in the interval $[0, 1]$? In general it is not always clear to decide whether Euclidean distance is appropriate and if it is not it may be hard to come up with a divergence measure that is appropriate. Moreover, it is not clear whether the 'prototypes' should be the same type of object as the data vectors (e.g. binary vs. real valued vectors).

Hence, an ongoing issue in research on self-organizing maps is to find appropriate distance measures, or more generally divergence measures, for different

types of data. However, there are no general ways to translate prior knowledge about the data into distance measures. The EM algorithm we present in the subsequent sections does provide us with a simple way to encode prior knowledge about the data into a self-organizing map algorithm. The mixture model can be designed based on our assumptions or prior knowledge on how the data is generated. Hence, we can separate the design of the model from the learning algorithm. For example, for binary vectors we might want to use a mixture of Bernoulli distributions to model their probability of occurrence.

Furthermore, in some applications some values in the data might be missing, e.g. a sensor on a robot brakes down or an entry in a questionnaire was left unanswered. In such cases it is common practice when applying Kohonen's SOM algorithm, see e.g. [15], to simply ignore the missing values and use an adapted divergence measure in order to compute the winning unit and to update the prototypes. The EM algorithm for self-organizing maps, just as the normal EM algorithm, provides a simple and justified way to deal with missing data for many types of mixtures.

In the next section, we review the standard EM algorithm for mixture models and in Section 4 we show how the EM algorithm can be modified to produce self-organizing maps.

3 Mixtures models and the EM algorithm

As stated in the introduction, mixture models are densities of the form:

$$p(\mathbf{x}) = \sum_{s=1}^{k} \pi_s p(\mathbf{x} \mid s),$$

where the π_s are the non-negative mixing weights that sum to one. The $p(\mathbf{x}|s)$ are called the component densities, parameterized by $\boldsymbol{\theta}_s$. The model thus makes the assumption that the data is generated by first selecting with probability π_s a mixture component s and then drawing a data item from the corresponding distribution $p(\cdot|s)$.

Given data $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, and initial parameters, collectively denoted as: $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k, \pi_1, \ldots, \pi_k\}$, the EM algorithm finds a local maximizer $\boldsymbol{\theta}$ of the data log-likelihood $L(\boldsymbol{\theta})$. Assuming data are independent and identically distributed (iid):

$$L(\boldsymbol{\theta}) = \log p(X; \boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}_n; \boldsymbol{\theta}).$$

Learning is facilitated by introducing N 'hidden' variables $z_1, \ldots, z_N$, collectively denoted as Z. Hidden variable z_n indicates which of the k mixture components generated the corresponding data point $\mathbf{x}_n$. The z_n are called hidden variables since we are only given $\mathbf{X}$ and do not know the 'true' values of the z_n. Intuitively, the EM algorithm [7] iterates between (E-step) guessing the values of the z_n based on the data and the current parameters, and (M-step) updating the parameters accordingly.

The EM algorithm maximizes a lower bound F on the data log-likelihood $L(\boldsymbol{\theta})$. The lower-bound is known as the negative free-energy because of its analogy with the free-energy in statistical physics [20]. The lower-bound F on $L(\boldsymbol{\theta})$ is a function of $\boldsymbol{\theta}$ and also depends on *some* distribution Q over the hidden variables Z, and is defined as:

$$F(Q, \boldsymbol{\theta}) = \mathrm{E}_Q \log p(\mathbf{X}, Z; \boldsymbol{\theta}) + H(Q) = L(\boldsymbol{\theta}) - D_{KL}(Q \parallel p(Z|\mathbf{X}; \boldsymbol{\theta})).$$

In the above we used H and D_{KL} to denote respectively Shannon entropy and Kullback-Leibler (KL) divergence [6]. The EM algorithm consists of coordinate ascent on F, iterating between steps in $\boldsymbol{\theta}$ and steps in Q.

The two forms in which we expanded F are associated with the M-step and the E-step of EM. In the M-step we change $\boldsymbol{\theta}$ as to maximize, or at least increase, $F(Q, \boldsymbol{\theta})$. The first decomposition includes $H(Q)$ which is a constant w.r.t. $\boldsymbol{\theta}$, so we essentially maximize the expected joint log-likelihood in the M-step. In the E-step we maximize F w.r.t. Q. Since the second decomposition includes $L(\boldsymbol{\theta})$ which is constant w.r.t. Q, what remains is a KL divergence which is the effective objective function in the E-step of EM. Note that, due to the non-negativity of the KL-divergence, for *any* Q we have: $F(Q, \boldsymbol{\theta}) \leq L(\boldsymbol{\theta})$.

For iid data the Q that maximizes F factors over the individual hidden variables and using such a factored distribution we can decompose the lower-bound, just like the log-likelihood, as a sum over data points:

$$F(Q, \boldsymbol{\theta}) = \sum_n F_n(q_n, \boldsymbol{\theta}). \tag{2}$$

$$F_n(q_n, \boldsymbol{\theta}) = \mathrm{E}_{q_n} \log p(\mathbf{x}_n, z_n = s; \boldsymbol{\theta}) + H(q_n) \tag{3}$$

$$= \log p(\mathbf{x}_n; \boldsymbol{\theta}) - D_{KL}(q_n \parallel p(z_n \mid \mathbf{x}_n; \boldsymbol{\theta})). \tag{4}$$

In the above, we wrote q_n for the distribution on the mixture components for data point n. Sometimes we will also simply write $p(s|\mathbf{x}_n)$ to denote $p(z_n = s|\mathbf{x}_n; \boldsymbol{\theta})$ and similarly q_{ns} for $q_n(z_n = s)$.

In standard applications of EM (e.g. mixture modelling) Q is unconstrained, which results in setting $q_n = p(z_n \mid \mathbf{x}_n; \boldsymbol{\theta})$ in the E-step since the non-negative KL divergence equals zero if and only if both arguments are equal. Therefore,

after each E-step $F(Q, \boldsymbol{\theta}) = L(\boldsymbol{\theta})$ and it follows immediately that the EM iterations can never decrease the data log-likelihood. Furthermore, to maximize F w.r.t. $\boldsymbol{\theta}_s$ and π_s, the parameters of component s, the relevant terms in F, using (2) and (3), are:

$$\sum_n q_{ns} \left[\log p(s) + \log(\mathbf{x}_n|s)\right].$$

Thus $\boldsymbol{\theta}_s$ is updated as to maximize a weighted sum of the data log-likelihoods. The q_{ns} are therefore also referred to as 'responsibilities', since each component is tuned toward data for which it is 'responsible' (large q_{ns}) in the M step.

For some probabilistic models it is not feasible to store the distribution $p(Z|\mathbf{X})$, e.g. in a hidden Markov random field [5] where the posterior has to be represented using a number of values that is exponential in the number of variables. In such cases variational methods [12] can be used: instead of allowing *any* distribution Q attention is restricted to a certain class $\mathcal{Q}$ of distributions allowing for tractable computations. In the hidden Markov random field example, using a factored distribution over the states would make computations tractable and the corresponding lower-bound on the data log-likelihood is also known as the mean-field approximation. Variational EM maximizes F instead of L, since we can no longer guarantee that F equals $L(\boldsymbol{\theta})$ after each E-step. The function F can be interpreted as summing log-likelihood and a penalty which is high if the true posterior is far from all members of $\mathcal{Q}$. Thus, applying the algorithm will find us models that (a) assign high likelihood to the data and (b) have posteriors $p(Z|\mathbf{X})$ that are similar to some member of $\mathcal{Q}$. In the next section, we show how we can constrain the distributions Q, not to achieve tractability, but to enforce self-organization.

4 Self-organization in mixture models

In this section we describe how we can introduce the self-organizing quality of Self-Organizing Maps in the mixture modelling framework. First, we will explain the idea in general and then give a simple example for modelling data with a Gaussian mixture. In the last two sections we discuss algorithmic issues.

4.1 The neighborhood function as a distribution.

Suppose we have found an appropriate class of component densities for our data, either based on prior knowledge or with an educated guess, for which we have an EM algorithm. By appropriately constraining the set $\mathcal{Q}$ of allowed

distributions q_n in the E-step we can enforce a topological ordering of the mixture components, as we explain below. This is much like the approach taken in [23,26], where constraints on the q_n are used to globally align the local coordinate systems of the components of a probabilistic mixture of factor analyzers. The latter works actually inspired the material presented here.

Here we consider the neighborhood function of Kohonen's SOM as a function of one component index by fixing the other component index. If we constrain the neighborhood function to be non-negative and sum to unity, it can be interpreted as a *distribution* over the components. We refer to such neighborhood functions as 'normalized' neighborhood functions. Thus, using the notation of (1), we have:

$$h_r(s) \propto \exp(-\lambda \parallel \mathbf{g}_r - \mathbf{g}_s \parallel^2), \qquad \sum_s h_r(s) = 1.$$

In order to obtain the self-organizing behavior in the mixture model learning we constrain the q_n to be normalized neighborhood functions. Thus, for fixed shape of the neighborhood function, the E-step will consist of selecting for each data item n the component r^* such that F_n is maximized if we use $q_n = h_{r^*}$. As before, we will call r^* the 'winner' for data item n. Below, we will analyze the consequences of constraining the q_n in this way.

What happens in the M-step if the q_n in the EM algorithm are normalized neighborhood functions? Similar as with Kohonen's SOM, if a specific component is the winner for $\mathbf{x}_n$ it receives most responsibility and will be forced to model $\mathbf{x}_n$, but also its neighbors in the latent space will be forced, but to a lesser degree, to model $\mathbf{x}_n$. So by restricting the q_n in the EM-algorithm to be a normalized neighborhood function centered on one of the units we thus force the components with large responsibility for a data point to be close in the latent space. In this manner we obtain a training algorithm for mixture models similar to Kohonen's SOM.

The same effect can also be understood in another way: consider the decomposition (4) of the objective function F, and two mixtures, parameterized by $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, that yield equal data log-likelihood $L(\boldsymbol{\theta}) = L(\boldsymbol{\theta}')$. The objective function F prefers the mixture for which the KL-divergences are the smallest, i.e. the mixture for which the posterior on the mixture components looks the most like the normalized neighborhood function. If the posteriors are truly close to the normalized neighborhood function then this implies that nearby components in the latent space model similar data. If the penalty term is large, this indicates that the posteriors do not resemble the neighborhood function and that the mixture is poorly 'organized'.

Note that these observations hold for any mixture model and that the al-

gorithm is readily generalized to any mixture model for which we have a maximum-likelihood EM algorithm.

Since the E-step is constrained, in general the objective function F does not equal log-likelihood after the E-step. Instead, the objective function sums data log-likelihood and a penalty term which measures the KL divergence between the (best matching, after the E-step) neighborhood function and the true posteriors for each data point. The modified EM algorithm can be summarized as follows:

EM algorithm for SOM
Initialize the parameters of the mixture model.
Iterate these steps:
- **E:** Determine for each $\mathbf{x}_n$ the distribution $q^* \in \mathcal{Q}$ that maximizes F_n, i.e. $q^* = \arg\min_{q \in \mathcal{Q}} D_{KL}(q \parallel p(s|\mathbf{x}_n))$, and set $q_n = q^*$.
- **M:** perform the normal M-step for the mixture, using the q_n computed in the E-step.

In fact, in the M-step we do not need to *maximize* F with respect to the parameters $\boldsymbol{\theta}$. As long as we can guarantee that the M-step does not decrease F, we are guaranteed that the iterations of the (modified) EM algorithm will never decrease F.

Finally, in order to project the data to the latent space, we average over the latent coordinates of the mixture components. Let $\mathbf{g}_s$ denote the latent coordinates of the s-th mixture component and let $\mathbf{g}_n$ denote the latent coordinates for the n-th data item. We then define $\mathbf{g}_n$ as:

$$\mathbf{g}_n = \sum_s p(s|\mathbf{x}_n)\mathbf{g}_s. \tag{5}$$

4.2 Example with Gaussian mixture.

Next, we consider an example of this general algorithm for a Mixture of Gaussians (MoG). A mixture giving a particularly simple M-step is obtained by using equal mixing weights $\pi_s = 1/k$ for all components and using isotropic Gaussians as component densities:

$$p(\mathbf{x}|s) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_s, \sigma^2\mathbf{I}).$$

For this simple MoG, the F_n can be rewritten up to some constants as:

$$F_n(q_n, \boldsymbol{\theta}) = -\frac{1}{2} D \log \sigma^2 - \sum_s q_{ns} \left[\sigma^{-2} \parallel \mathbf{x}_n - \boldsymbol{\mu}_s \parallel^2 /2 + \log q_{ns} \right].$$

The parameters of the mixture are $\boldsymbol{\theta} = \{\sigma^2, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$. The parameters $\boldsymbol{\theta}$ maximizing F for given $q_1, \ldots, q_N$ are obtained easily through differentiation of F and the complete algorithm for D-dimensional data reads:

EM algorithm for SOM for simple MoG
Iterate these steps:

- **E:** Determine for each $\mathbf{x}_n$ the distribution $q^* \in \mathcal{Q}$ that maximizes F_n, set $q_n = q^*$.
- **M:** Set: $\boldsymbol{\mu}_s = \sum_n q_{ns} \mathbf{x}_n / \sum_n q_{ns}$ and $\sigma^2 = \sum_{ns} q_{ns} \parallel \mathbf{x}_n - \boldsymbol{\mu}_s \parallel^2 /(ND)$.

A Matlab implementation of SOMM for the isotropic MoG is available at: `http://www.science.uva.nl/~jverbeek/software/`.

Note that the update for the means $\boldsymbol{\mu}_s$ coincides with that of Kohonen's batch SOM algorithm when using the Euclidean distance: it places each prototype at the weighted average of all data, where each data vector is weighted proportionally to the neighborhood function centered at the winning prototype for that data vector and evaluated at the prototype s. Only definition of the winner is different in this case: instead of the minimizer of $\parallel \mathbf{x}_n - \boldsymbol{\mu}_r \parallel^2$ we select the minimizer of $\sum_s h_r(s) \parallel \mathbf{x}_n - \boldsymbol{\mu}_r \parallel^2$. Hence, the selection of the winner also takes into account the neighborhood function, whereas this is not the case in the standard Kohonen SOM.

4.3 Shrinking the neighborhood function

A particular choice for the normalized neighborhood function is obtained by associating with each mixture component s a latent coordinate $\mathbf{g}_s$. It is convenient to take the components as located on a regular grid in the latent space. We then set the normalized neighborhood functions to be discretized isotropic Gaussians in the latent space, centered on one of the components. Thus, for a given set of latent coordinates of the components $\{\mathbf{g}_1, \ldots, \mathbf{g}_k\}$, we set $\mathcal{Q}$ to be the set of distributions of the form:

$$q(s) = \frac{\exp\left(-\lambda \parallel \mathbf{g}_s - \mathbf{g}_r \parallel^2\right)}{\sum_t \exp\left(-\lambda \parallel \mathbf{g}_t - \mathbf{g}_r \parallel^2\right)}, \qquad \text{with } r \in \{1, \ldots k\}.$$

A small λ corresponds to a broad distribution, and for large λ the distribution q becomes more peaked.

Since the objective function might have local optima and EM is only guaran-

teed to give a local optimizer of F, good initialization of the parameters of the mixture model is essential to finding a good solution. Analogous to the method of shrinking the extent of the neighborhood function with the SOM, we can start with a small λ (broad neighborhood function) and increase it iteratively until a desired value of λ (i.e. the desired width of the neighborhood function) is reached. In implementations we started with λ such that the q are close to uniform over the components, then we run the EM algorithm until convergence. Note that if the q are almost uniform the initialization of $\boldsymbol{\theta}$ becomes irrelevant, since the responsibilities will be approximately uniform over all components. The subsequent M step will give all components similar parameters. After convergence we set $\lambda^{new} \leftarrow \eta\lambda^{old}$ with $\eta > 1$ (typically η is close to unity). In order to initialize the EM procedure with λ^{new}, we initialize $\boldsymbol{\theta}$ with the value found in running EM with λ^{old}.

Note that in the limit of $\lambda \to \infty$ it is *not* the case that we recover the usual EM algorithm for mixtures. Instead, a 'winner-take-all' (WTA) algorithm is obtained since in the limit the distributions in $\mathcal{Q}$ tend to put all mass on a single mixture component. The WTA algorithm tends to find more heterogeneous mixture components than the usual EM algorithm [14]. Arguably, this can be an advantage for visualization purposes.

4.4 Sparse winner search

The computational cost of the E-step is $O(Nk^2)$, a factor k slower than SOM and prohibitive in large-scale applications. However, by restricting the search for a winner in the E-step to a limited number of candidate winners we can obtain an $O(Nk)$ algorithm. A straightforward choice is to use the l components with the largest joint likelihood $p(\mathbf{x}_n, s)$ as candidates, corresponding for our simple example MoG to smallest Euclidean distances to the data point. If none of the candidates yields a higher value of $F_n(q_n, \boldsymbol{\theta})$ we keep the winner of the previous step, in this way we are guaranteed never to decrease the objective in every step. We found $l = 1$ to work well and fast in practice, in this case we only check whether the winner from the previous round should be replaced with the node with highest joint-likelihood $p(s, \mathbf{x}_n)$.

Note that in the $l = 1$ case our algorithm is very similar to Kohonen's batch SOM algorithm for the simple MoG model. We will return to this issue in Section 6.

5 Extensions of the basic framework

In this section we describe two extensions of the basic framework for self-organization in mixture models. In Section 5.1 we consider how data with missing values can be dealt with and in Section 5.2 we consider how the adaptive-subspace SOM can be fitted in the proposed framework.

5.1 Modelling data with missing values

When the given data has missing values the EM algorithm can still be used to train a mixture on the incomplete data [8]. The procedure is more or less the same as with the normal EM algorithm to train a mixture model. The missing data are now also handled as 'hidden' variables. Below, we consider the case for iid data.

Let us use $\mathbf{x}_n^o$ to denote the observed part of $\mathbf{x}_n$ and $\mathbf{x}_n^h$ to denote the missing or 'hidden' part. The distributions q_n now range over all the hidden variables of $\mathbf{x}_n$: the generating component z_n, and possibly $\mathbf{x}_n^h$ if there are some unobserved values for $\mathbf{x}_n$. We can now again construct a lower-bound on the (incomplete) data log-likelihood as follows:

$$\begin{aligned} F(Q, \boldsymbol{\theta}) &= \sum_n F_n(q_n, \boldsymbol{\theta}). \\ F_n(q_n, \boldsymbol{\theta}) &= \log p(\mathbf{x}_n^o; \boldsymbol{\theta}) - D_{KL}(q_n \parallel p(z_n, \mathbf{x}_n^h | \mathbf{x}_n^o)) \\ &= \mathrm{E}_{q_n} \log p(\mathbf{x}_n^o, \mathbf{x}_n^h, z_n; \boldsymbol{\theta}) + H(q_n). \end{aligned} \tag{6}$$

The EM algorithm then iterates between setting in the E-step: $q_n = p(z_n, \mathbf{x}_n^h | \mathbf{x}_n^o)$. The M-step updates $\boldsymbol{\theta}$ to maximize (or at least increase) the expected joint log-likelihood: $\sum_n \mathrm{E}_{q_n} \log p(\mathbf{x}_n^o, \mathbf{x}_n^h, z_n; \boldsymbol{\theta})$. After each E-step F equals the (incomplete) data log-likelihood, so we see that the EM iterations can never decrease the log-likelihood.

Now consider the decomposition: $q_n(z_n, \mathbf{x}_n^h) = q_n(z_n) q_n(\mathbf{x}_n^h | z_n)$. We can rewrite (6) as:

$$\begin{aligned} F_n(q_n, \boldsymbol{\theta}) = &H(q_n(z_n)) \\ &+ \mathrm{E}_{q_n(z_n)} \left[H(q_n(\mathbf{x}_n^h | z_n)) + \mathrm{E}_{q_n(\mathbf{x}_n^h | z_n)} \log p(z_n, \mathbf{x}_n^h, \mathbf{x}_n^o) \right]. \end{aligned} \tag{7}$$

Just as before, we can now constrain the $q_n(z_n)$ to be a normalized neighborhood function. We then find that in order to maximize F w.r.t. Q in the E-step we have to set $q_n(\mathbf{x}_n^h | s) = p(\mathbf{x}_n^h | \mathbf{x}_n^o, s)$. The optimization of F w.r.t. $q_n(z_n)$ is

then achieved by choosing $q_n(z_n)$ to be the neighborhood function that maximizes (7). In the M-step we then have to maximize (or at least increase) the expected joint (complete) data log-likelihood w.r.t. $\boldsymbol{\theta}$:

$$\sum_n \sum_{z_n} q_n(z_n) \left[\log \pi_{z_n} + \mathrm{E}_{q_n(\mathbf{x}_n^h|z_n)} \log p(\mathbf{x}_n^o, \mathbf{x}_n^h|z_n) \right] .$$

The ease with which the concrete M step can be derived and computed, and whether we can do maximization of F or just increase in F in the M step, depends on the specific mixture that is employed.

5.2 The Adaptive-subspace Self-Organizing Map

The adaptive-subspace self-organizing map (ASSOM) [16] learns invariant features of data patterns. For example, the data patterns could be small images of various textures where each texture pattern can be transformed in several ways, e.g. translated, rotated and scaled. The ASSOM can be used to find the features of the various textures that are invariant for the different transformations [15,22].

The ASSOM does not process single data points (patterns) but small sets, called 'episodes', of slightly transformed patterns. The data points in an episode are forced to be assigned the same winner, thereby forcing the nodes of the map to fit patterns *and* their transformations. The nodes of the basic ASSOM are parameterized by a set of vectors that together span a subspace of the complete signal space that contains the patterns, and their transformations, assigned to a particular node.

The ASSOM can be readily cast in the probabilistic framework presented here as well. To do this we force all patterns $\mathbf{x}$ in an episode of transformed patterns $\mathcal{E}$ to share the same responsibilities $q_{\mathcal{E}}(s)$ over the components s of the mixture. In the E-step we then have to set $q_{\mathcal{E}}$ to maximize:

$$\begin{aligned} F_{\mathcal{E}} &= \sum_{\mathbf{x} \in \mathcal{E}} \sum_s q_{\mathcal{E}}(s) \left[\log p(\mathbf{x}, s) - \log q_{\mathcal{E}}(s) \right] \\ &= |\mathcal{E}| \sum_s q_{\mathcal{E}}(s) \left[\log p(s) + \langle \log p(\mathbf{x}|s) \rangle_{\mathcal{E}} - \log q_{\mathcal{E}}(s) \right] , \end{aligned}$$

where $\langle \log p(\mathbf{x}, s) \rangle_{\mathcal{E}} = \frac{1}{|\mathcal{E}|} \sum_{\mathbf{x} \in \mathcal{E}} \log p(\mathbf{x}, s)$ and $|\mathcal{E}|$ denotes the cardinality of the episode. Compared to the basic E-step presented in Section 4, we simply replace the complete log-likelihood $\log p(\mathbf{x}, s)$ with the average log-likelihood over the episode: $\langle \log p(\mathbf{x}, s) \rangle_{\mathcal{E}}$.

Since the ASSOM nodes span subspace of the original signal space, it is intuitive to use a mixture of probabilistic principal component analyzers (PPCA) [24] as a generative model. The components of a mixture of PPCA are Gaussians with a constrained form of covariance matrix: $\Sigma = \sigma^2\mathbf{I} + \Lambda\Lambda^\top$. The d columns of the matrix Λ span the subspace with large variance and the $\sigma^2\mathbf{I}$ term adds some small variance to each direction that makes the covariance matrix non-singular.

Recently, we also applied the idea of using 'shared' responsibilities for several data points to speed-up the training of Gaussian mixtures on large databases [28]. Before training the model, the data is partitioned into a desired number of disjoint subsets or 'cells', within each cell the responsibilities will be shared. The EM algorithm can then be performed based only on sufficient statistics of the cells. This reduces the linear dependence of the time complexity of the EM iterations on the number of data points to a linear dependence on the number of used cells. It is easily shown that this still maximizes a lower-bound, similar to the bound F used here, on the true data log-likelihood. Note that this idea can also be combined with the self-organizing mixture models of the previous sections.

6 Comparison with related work

In this section we will compare the constrained EM algorithm presented here with several other self-organizing map algorithms.

6.1 Kohonen's SOM

Let us first consider the original SOM algorithm proposed by Kohonen [15] and already discussed in Section 2. When applied to the simple MoG of Section 4, our approach is very close to Kohonen's SOM. We already noted that the update of the means of the Gaussians is the same as the update for the prototypes in Kohonen's SOM with Euclidean distance as divergence.

Note that for this simple model the conditional log-likelihood $\log p(\mathbf{x}|s)$ is given by the negative squared distance $-\parallel \mathbf{x} - \boldsymbol{\mu}_s \parallel^2$, up to some additive and multiplicative constants. If we use a sparse search for the winner (E-step) with only $l = 1$ candidate, the difference with Kohonen's winner selection is that we only accept the closest node, in terms of Euclidean distance, as a winner when it increases F, and keep the previous winner otherwise.

In comparison to the standard Kohonen SOM, we presented a SOM algorithm

applicable to any type of mixture model. We can use prior knowledge of the data to design a specific generative mixture model for given data and then train it with the constrained EM algorithm. Also, the probabilistic formulation of the SOM presented here allows us to handle missing data in a principled manner and to learn mixtures of self-organizing maps. Furthermore, the probabilistic framework helps us design SOM algorithms for data types which were difficult to handle in the original SOM framework, e.g. data that consists of variable-size trees or sequences, data that combines numerical with categorical features, etc.

6.2 Soft Topographic Vector Quantization

Soft Topographic Vector Quantization (STVQ) [10,11], is a SOM-like algorithm similar to the one presented here. Given some divergence measure $d(\mathbf{x}_n, s)$ between data items and neurons, the following error function is minimized by STVQ:

$$\mathcal{E}_{STVQ} = \beta \sum_{ns} p_{ns} \sum_{r} h_s(r) d(\mathbf{x}_n, s) + \sum_{ns} p_{ns} \log p_{ns},$$

with all $p_{ns} \geq 0$ and $\sum_s p_{ns} = 1$. The error function sums an error term based on d (which could for example be based on squared Euclidean distance, but other errors based on the exponential family can be easily plugged in) and an entropy term. The neighborhood function, implemented by the $h_s(r)$, is fixed, but the winner assignment, given by the p_{ns}, is soft. Instead of selecting a single 'winner' as in our work, an (unconstrained) *distribution* p_{ns} over the units is used. Since we use a complete distribution over the units, we cannot apply the speed-up of 'sparse' search for the winning neuron here. However, speed-ups can be obtained by using a 'sparse' EM algorithm [20] in which the p_{ns} a adapted only for a few components s (different ones for each data item) and freezing the values p_{ns} for other components.

The parameter β is used for annealing in STVQ: for small β the entropy term, with only one *global* optimum, becomes dominant, whereas for large β the quantization error, with many local optima, becomes dominant. By gradually increasing β more structure is added to the objective function. In comparison, in our work we started with a broad neighborhood function and shrink it gradually to the desired shape in order to be less sensitive to the initialization of the parameters.

The error function $\mathcal{E}_{STVQ}$ can be rewritten as a sum of log-likelihood of the data under a mixture model $\tilde{p}$ plus a penalty term independent of the data at hand [11]. Using squared Euclidean distance as divergence measure, the components and mixing weights of the mixture model $\tilde{p}$ are given by :

$$
\begin{aligned}
\tilde{p}(\mathbf{x}) &= \sum_{s=1}^{k} \tilde{\pi}_s \mathcal{N}(\mathbf{x}; \tilde{\boldsymbol{\mu}}_s, \beta^{-1}\mathbf{I}), \\
\tilde{\pi}_s &= \frac{\exp\left(-\beta v_s\right)}{\sum_{s'=1}^{k} \exp\left(-\beta v_{s'}\right)}, \\
\tilde{\boldsymbol{\mu}}_s &= \sum_{r=1}^{k} h_{sr} \boldsymbol{\mu}_r, \\
v_s &= \sum_{r=1}^{k} h_{sr} \parallel \boldsymbol{\mu}_s - \boldsymbol{\mu}_r \parallel^2 .
\end{aligned}
$$

The validity of this interpretation hinges on the particular divergence measure that is used, the squared Euclidean distance here. The divergence measure should be derived from distributions in the exponential family to validate the mixture model interpretation. Note that the annealing parameter β, which is set by hand or through an annealing schedule, appears as a parameter in the corresponding mixture model.

In comparison, the relation of our error function F to the log-likelihood of the data under a mixture model (log-likelihood plus a penalty between the true posterior and the neighborhood function) also holds for models outside the exponential family. Furthermore, the annealing scheme used, in our case the width of the neighborhood function, is in the objective function, but appears in the penalty term rather than in the log-likelihood term. We feel our model allows for a clearer link to log-likelihood under a mixture model, in particular because the parameters of the mixture model are learned directly and do not include an annealing parameter.

6.3 Other probabilistic SOM algorithms

A different way to cast the SOM in a probabilistic framework is given by [1]. The log-likelihood function that is maximized is:

$$
L' = \sum_{n} \log \sum_{s=1}^{k} p_{ns} p(\mathbf{x}|s).
$$

Notice that effectively each data point has its likelihood evaluated under its own mixture of the component densities $p(\cdot|s)$ with mixing weights p_{ns}. Two steps are iterated to maximize this likelihood. In the first step, similar to the E-step, we set:

$$
p_{ns} = h_{r_n s} \quad \text{with: } r_n = \arg\max_{r} \sum_{s} h_{rs} p(\mathbf{x}_n|s).
$$

The M-step finds a new parameter vector for given p_{ns}. However, the new parameter vector is not guaranteed to increase L' for the fixed p_{ns}.

For this model it is in principle also possible to use any mixture model. However, this algorithm does not optimize data log-likelihood under a *single* mixture, since the mixing weights vary for each data item and change throughout the iterations of the algorithm. Moreover, the learning steps are not guaranteed to improve the objective function. The algorithm has run-time $O(Nk^2)$, but can benefit from the same speed-up discussed in the previous section.

In [17] a probabilistic model is given based on isotropic Gaussian components. The maximum-likelihood estimation procedure of the means of the Gaussians coincides with the estimation of the vector quantizers with the batch version of Kohonen's SOM algorithm. There are, however, some difficulties with this approach. The density is Gaussian in each of the Voronoi regions associated with the vector quantizers and thus not smooth in the data space as a whole. Second, the estimation of the variance of the Gaussians needs to be done by numerical optimization techniques. Third, to evaluate the likelihood a normalization constant has to be computed. The constant is defined in terms of integrals that can not be analytically computed and therefore sampling techniques are needed to approximate it. Furthermore, the density model seems to be restricted to Gaussian mixtures with the corresponding squared Euclidean distance as divergence measure.

Other maximum-likelihood approaches for mixture models achieve topographic organization by a smoothness prior [25] on the parameters. For this method is it not clear whether it generalizes directly to mixture models outside the exponential family and whether speed-ups can be applied to avoid $O(nk^2)$ run-time.

6.4 Generative Topographic Mapping

Generative Topographic Mapping (GTM) [3] achieves topographic organization in a quite different manner than the self-organizing map. GTM fits a constrained probabilistic mixture model to given data. With each mixture component s a latent coordinate $\mathbf{g}_s$ is associated, like in our approach. The mixture components are parameterized as a linear combination of a fixed set of smooth nonlinear basis functions of the fixed locations of the components in the latent space. Due to the smooth mapping from latent coordinates of the mixture components to their parameters, nearby components in the latent space will have similar parameters. The parameters of the linear map are fitted by the EM algorithm to maximize the data log-likelihood. Although originally presented for Gaussian mixtures, the GTM can be extended to mix-

tures of members of the exponential family [9,13]. To our knowledge it is not known how to extend the GTM to mixtures of component densities outside the exponential family.

The main benefit of GTM over SOMM is that the parameter fitting can be done directly in the maximum likelihood framework. For SOMM we need to specify the number of mixture components and a desired shape and width of the neighborhood function. For GTM we also need to set parameters: the number of mixture components, the number of basis functions, their smoothness and their shape. For GTM these parameters can be dealt with in a Bayesian manner, by specifying appropriate priors for these parameters. Optimal parameter settings can then be found by inferring their values using approximate inference techniques if enough computational resources are available [2]. For the SOMM approach it is not clear whether such techniques can be used to set the parameters of the algorithm. This may be considered a drawback of the SOMM approach, however in situations with limited computational resources the SOMM has fewer parameters to be set and may therefore be preferred.

7 Example applications

In this section we illustrate the modified EM algorithm with some examples. We start with a more extensive treatment of the example already used in Section 4. The second example uses a mixture of (products of) discrete Bernoulli distributions. In the third example, we use mixtures for data that has both nominal and continuous variables.

In all experiments we used discretized Gaussians as neighborhood function and the same scheme to shrink the neighborhood function. The mixture components were arranged as (i) a regular rectangular grid on the unit square for a two-dimensional latent space or (ii) at regular intervals on the real line between zero and one for a one-dimensional latent space. At initialization we set the neighborhood function width as that of a isotropic Gaussian with unit variance, i.e. $\lambda = 1/2$. After the algorithm converges with a particular λ we increase it with a factor $\eta = 11/10$. The algorithm halts if a winning node receives more than 90% of the mass of its neighborhood function.

7.1 The simple Gaussian mixture

In the first example we show the configurations of the means of the simple MoG model used in Section 4. We generated a data set by drawing 500 point uniformly random in the unit square. We trained two SOMM models on the

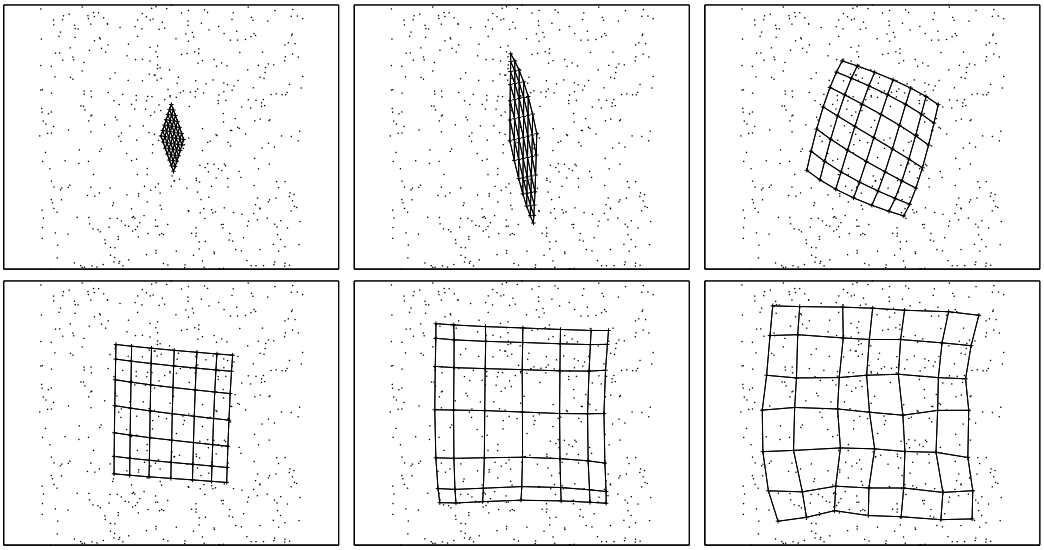

Fig. 2. Configuration of the SOM training after $10, 20, \ldots, 60$ EM iterations.

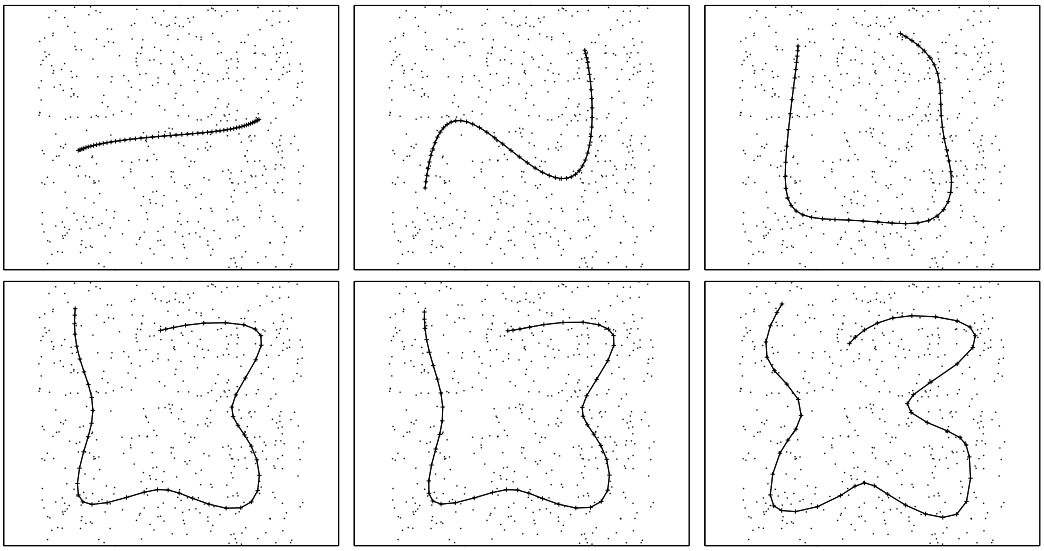

Fig. 3. Configuration of the SOM training after $20, 40, \ldots, 120$ EM iterations.

data.

First we used 49 nodes on a 7 by 7 grid in a two dimensional latent space. In Fig. 2 we show the means of the Gaussians, connected according to the grid in the latent space. Configurations after each 10 EM iterations are shown for the first 60 iterations. It can be clearly seen how the SOMM spreads out over the data as the neighborhood function is shrunk until it has about 80% of its mass at the winner at iteration 60.

Fig. 3 shows the same experiment but with 50 Gaussians placed at a regular intervals in a 1d latent space. Here we showed the configuration after each 20 iterations. Again, we can observe the SOMM spreading out over the data as

k	d	F_{KSOM}	σ_{KSOM}	F_{SOMM}	σ_{SOMM}	$F_{SOMM} - F_{KSOM}$
10	1	-2861.2	17.4	-2866.3	12.1	-5.1
25		-3348.2	21.0	-3354.6	13.4	-6.2
100		-4131.6	21.9	-4138.6	13.7	-7.0
9	2	-2812.7	90.6	-2796.9	164.2	-15.8
25		-3367.5	10.3	-3365.0	10.3	2.5
100		-4097.5	10.5	-4096.3	8.5	1.1

Fig. 4. Comparison of KSOM and SOMM in terms of the SOMM objective.

the neighborhood function width decreases. However, in this case the SOMM has to fold itself in order to approach the uniform distribution of the data.

These results are very similar to results reported on this data for other SOM algorithms. The example is included here to show this similarity for readers already familiar with the SOM. On the other hand it provides an illustrative example of the training of the SOM for readers unfamiliar with SOM. To quantify the similarity with Kohonen's SOM (KSOM) we compared the results obtained with KSOM with the results obtained with our SOMM. For both methods we evaluated the objective function of SOMM. In order to be able to evaluate this objective for KSOM, we used the means as obtained with KSOM and used the variances obtained with SOMM to specify a mixture model. Both methods used the same neighborhood shrinking scheme. In Fig. 4 we summarize the results for different numbers of mixture components k, and dimensions of the latent space d. The reported results F_{SOMM} and F_{KSOM} are averages over 50 experiments, the standard deviation of the results is given in the σ columns. It can be observed that the differences are quite small considering the standard deviations.

7.2 Mixture of Bernoulli's

Here we use a SOM model for word occurrence data in documents. The example shows the application of the EM algorithm for SOM on a mixture with non-Gaussian components. The data consists of a matrix with entries that are either zero or one. Each row represents a word and each column represents a document. A one indicates that the specific word occurs in a specific document. Thus, the words are the data items and the occurrence in documents are the features. The word occurrence data were obtained from the 20 newsgroup dataset [2] . The data set contains occurrence of 100 words in 16242 newsgroup postings.

[2] The 20-newsgroup dataset is available through the internet at: `http://www.ai.mit.edu/~jrennie/20Newsgroups/`.

We use very simple mixture components that assume independence between all the features and each feature is Bernoulli distributed. We use N to indicate the number of words and D to denote the number of documents. Words are indexed by n and we use $x_n^d \in \{0, 1\}$ to denote the value of the d-th feature (presence in document d) of the n-th word $\mathbf{x}_n$. The probability of the d-th feature being '1' according to the s-th mixture component is denoted p_{sd}. Formally we have:

$$p(\mathbf{x}_n|s) = \prod_{d=1}^{D} p_{sd}^{x_n^d}(1 - p_{sd})^{(1-x_n^d)}.$$

The mixing weights of all $k = 25$ components were taken equal.

The EM algorithm to train this model is particularly simple. For the E-step we need the posteriors $p(s|\mathbf{x}_n)$ which can be computed using Bayes rule:

$$p(s|\mathbf{x}) = \frac{p(\mathbf{x}|s)\pi_s}{\sum_s p(\mathbf{x}|s)\pi_s}.$$

In the M-step we have to maximize the expected joint log-likelihood w.r.t. the parameters p_{sd}. Which yields:

$$p_{sd} = \frac{\sum_n q_{ns} x_n^d}{\sum_n q_{ns}},$$

which is simply the relative frequency of the occurrence of the words in document d where all the words are weighted by the responsibility q_{ns}.

Due to the huge dimensionality of this data, the posteriors are quite peaked, i.e. the posteriors have very low entropy resulting in $\mathbf{g}_n$ almost equal to one of the $\mathbf{g}_s$. Although the obtained assignment of words to components is sensible, the latent coordinates obtained with (5) are less useful. This is because peaked posteriors will drive the $\mathbf{g}_n$ close to one of the $\mathbf{g}_s$, and thus plotting the words at the corresponding coordinates $\mathbf{g}_n$ will result in plotting the words with high posterior for the same component on top of each other. In order to reduce this problem, for visualization we did not use the actual posterior but the distribution $p'(s|\mathbf{x}_n) \propto p(s|\mathbf{x}_n)^\alpha$ for $\alpha < 1$ such that the entropies of the $p'(s|\mathbf{x}_n)$ were close to two bits.[3] In Fig. 5 we plotted the words at their expected latent coordinate, the bottom plot zooms in on area indicated by the box in the top plot.

[3] The justification for using the normalized exponentiated posteriors is that if we have a distribution and want the distribution that is closest in Kullback-Leiber sense to the original under a constraint on the entropy, then we should use the normalized

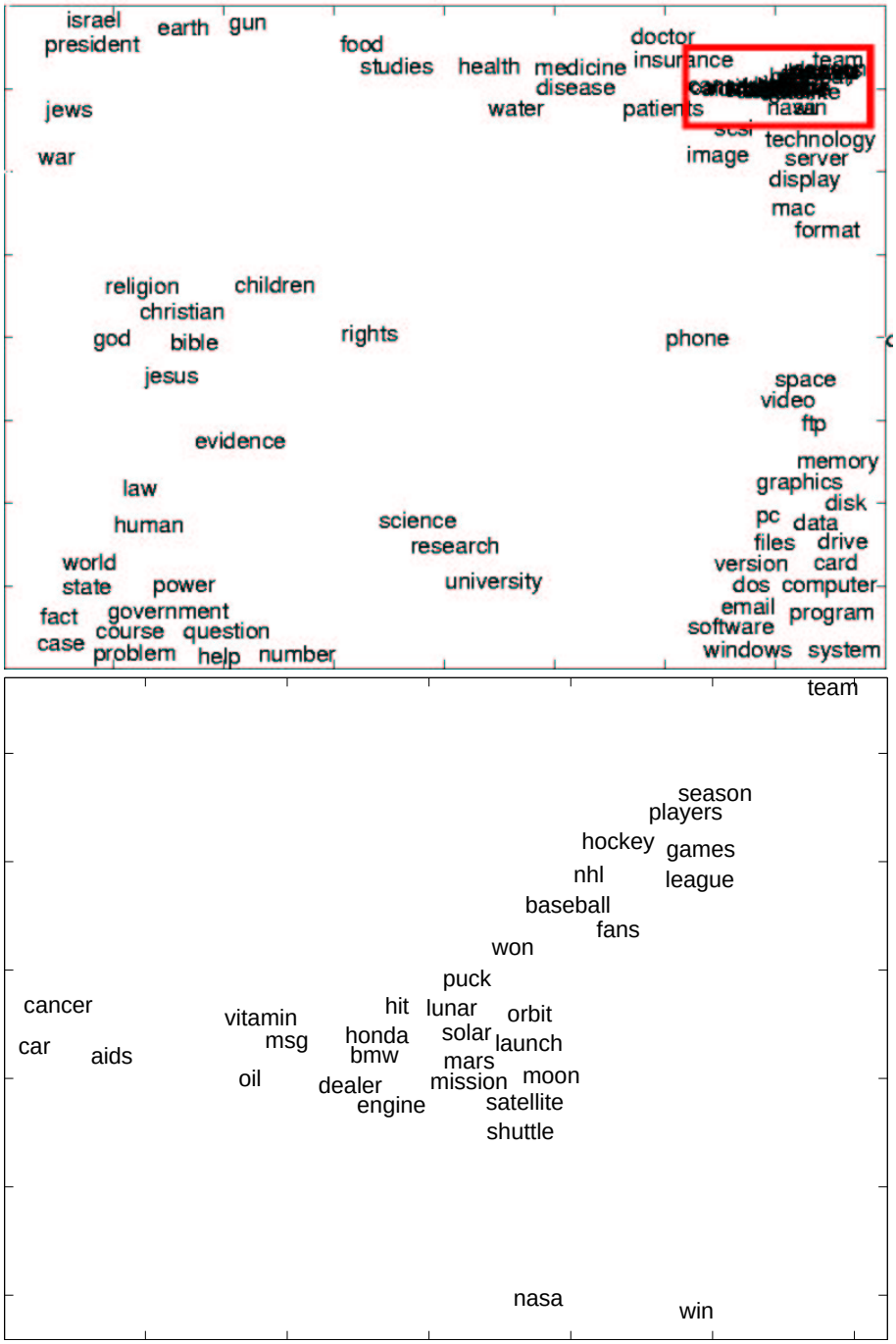

Fig. 5. Self-organizing Bernoulli models, $k = 25$. Bottom plot displays area in the box in the top plot. Note that the ambiguous word 'win' (referring to the operating system 'windows' and to 'winning' a game) is located between the computer terms (lower and middle right area) and the sports terms (upper right area).

exponentiated distribution.

7.3 Modelling credit card application data

In this section we consider a self-organizing mixture model for data of 690 credit card applications. The data set is available from the UCI machine learning repository [4]. The data has 6 numerical features and 9 nominal attributes. For some records the values of one or more attributes are missing, in the example below we removed these cases from the data set for simplicity. The meaning of the attributes is not provided with the database for privacy reasons, except for the attribute that indicates whether the credit card application was approved or not. Therefore, interpretation of modelling results in terms of customer profiles is difficult. However, below we summarize and compare results we obtained when applying different SOM algorithms on this data. The SOM models all had 25 units arranged in a 5 by 5 grid.

The usual approach to apply the KSOM to data that has nominal attributes is to use the one-of-n ($1/n$) encoding. In this encoding, each nominal variable is represented with a binary vector, one bit for each possible value, with a '1' in the location of the particular value a data item has for the nominal variable. The KSOM model can be trained either by proceeding as usual on the $1/n$ encoding, or by constraining the means of the neurons to be in the $1/n$ encoding as well [21]. In our experiments we implemented the last option by doing the normal batch KSOM update step, but then mapping the updated mean vector to the $1/n$ encoding with a '1' in the position of the maximum value of the mean vector. In the results below the normal KSOM on the data in $1/n$ encoding is indicated by KSOM and the version with constrained means is indicated with KSOM-$1/n$.

In this experiment we considered SOMMs with several types of mixture components. The density on the continuous data was a multivariate Gaussian with differently shaped covariance matrices: isotropic, diagonal, and general. We used two different distributions on the nominal variables. The first distribution modelled all 9 nominal variables independently. The second distribution modelled some pairs of nominal variables jointly. The 4 pairs of nominal variables that were modelled jointly were selected by computing (based on the complete data set) the mutual information between pairs of nominal variables and we selected the pairs with maximum mutual information in a greedy manner. For two of the four pairs the mutual information was in the order of the entropy of one the variables in the pair, indicating strong correlations between the variables. We compared all six combinations of covariance matrix structure and independent/pair-wise nominal variable models. Different combinations are indicated by SOMM-xy, where x denotes the covariance matrix structure: 'i', 'd' or 'g' and y denotes the nominal variable model: 'i' or 'p'.

The continuous variables had considerably different ranges of values (differing

method	F	D
KSOM	-6070.0	443.1
KSOM-$1/n$	-6389.8	532.5
SOMM-ii	-5678.8	297.2
SOMM-di	-1481.3	380.3
SOMM-gi	-1398.3	411.7
SOMM-ip	-5147.2	232.1
SOMM-dp	-1001.9	342.8
SOMM-gp	-1096.4	342.5

Fig. 6. Comparison in terms of the SOMM objective F and the penalty term D.

several orders of magnitude). Therefore, we scaled all continuous variables to have unit variance before applying the different SOM(M) models. Note that for diagonal and general covariance matrices this scaling is superfluous, since the model can take the different variances into account.

In order to compare the KSOM models with the SOMM models, we used the objective function of SOMM. When we train a KSOM model this yields a final assignment of data items to neurons. We used the neighborhood function of the winning neuron for a data point as its responsibility over mixture components. We then performed a single M-step of SOMM with isotropic covariance matrix and an independent model for the nominal variables. We used this probabilistic model since it is most closely related the results obtained with the KSOM algorithm. We then evaluated the SOMM objective function for the resulting mixture model. The resulting averages of the SOMM objective function F and the penalty term D for the different models are presented in Fig. 6. Averages were taken over 20 experiments, each time selecting a random subset of 620 of the total 653 records. The results support two conclusions.

First, when using independent distributions for the nominal and isotropic covariance matrix for the continuous variables, we see that the SOMM-ii algorithm obtains higher objective values than the KSOM algorithms. The KSOM yields better results than the KSOM-$1/n$ algorithm. Also taking into account the differences in the penalty terms, we can conclude that on average the SOMM models yield both a higher likelihood and better topology preservation.

Second, using models that use more expressive component densities we can obtain considerably higher scores of the objective function. In particular the models that do not assume equal variance in all continuous variables (isotropic covariance) obtain relatively high scores. This indicates that these models give a much better fit on the data and thus that they can provide more realistic descriptions of the customers in the clusters.

8 Conclusions

We showed how we can change any mixture model for which we have an EM algorithm into a self-organizing map version by a simple modification of the E-step of the EM algorithm. The use of mixture models as basis for SOMs allows for easy design of divergence measures by designing a generative model for the data based on prior knowledge and/or assumptions. The EM framework allows one to deal with missing data in a principled way, by estimating in each EM iteration the missing values based on the current parameters of the mixture model. The objective function is in a clear manner related to the data log-likelihood under the mixture model. In our view the modified EM algorithm for SOM does not come with any drawbacks as compared to existing SOM algorithms. In our approach it is not obvious how hyperparameters such as the neighborhood width or the number of components could be set in an automatic manner. For the GTM, Bayesian techniques can be used to set hyperparameters, such as a penalty term on the smoothness of the non-linear basis functions. However, such Bayesian techniques are computationally expensive. In some applications the simplicity of the SOMM might therefore be preferred, depending on the computational resources.

Acknowledgments

This research is supported by the Technology Foundation STW (project nr. AIF4997) applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs. We like to thank the organizers of ESANN 2003 for inviting us to submit this extended version of our paper [27] to Neurocomputing. We would also like to thank the reviewers for their detailed comments and suggestions.

References

[1] F. Anouar and F. Badran and S. Thiria. Probabilistic self-organizing map and radial basis function networks. *Neurocomputing*, 20(1-3):83–96, August 1998.

[2] C. M. Bishop, M. Svensén, and C. K. I. Williams. Developments of the generative topographic mapping. *Neurocomputing*, 21:203–224, 1998.

[3] C. M. Bishop, M. Svensén, and C. K. I Williams. GTM: the generative topographic mapping. *Neural Computation*, 10:215–234, 1998.

[4] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[5] G. Celeux, F. Forbes, and N. Payrard. EM procedures using mean field-like approximations for Markov model-based image segmentation. *Pattern Recognition*, 36:131–144, 2003.

[6] T. Cover and J. Thomas. *Elements of information theory*. Wiley, 1991.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[8] Z. Ghahramani and M. I. Jordan. Supervised learning from incomplete data via an EM approach. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 120–127. Morgan Kaufmann Publishers, Inc., 1994.

[9] M. Girolami. The topographic organisation and visualisation of binary data using mutivariate-Bernoulli latent variable models. *IEEE Transactions on Neural Networks*, 12(6):1367–1374, 2001.

[10] T. Graepel, M. Burger, and K. Obermayer. Self-organizing maps: generalizations and new optimization techniques. *Neurocomputing*, 21:173–190, 1998.

[11] T. Heskes. Self-organizing maps, vector quantization, and mixture modeling. *IEEE Transactions on Neural Networks*, 12:1299–1305, 2001.

[12] M. I. Jordan, Z. Ghahramani, T. Jaakola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999.

[13] A. Kaban and M. Girolami. A combined latent class and trait model for the analysis and visualization of discrete data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:859–872, 2001.

[14] M. J. Kearns, Y. Mansour, and A. Y. Ng. An information-theoretic analysis of hard and soft assignment methods for clustering. In M.I. Jordan, editor, *Learning in Graphical Models*, pages 495–520. Kluwer, 1998.

[15] T. Kohonen. *Self-organizing maps*. Springer, 2001.

[16] T. Kohonen, S. Kaski, and H. Lappalainen. Self-organized formation of various invariant-faeture filters in the adaptive-subspace SOM. *Neural Computation*, 9(6):1321–1344, 1997.

[17] T. Kostiainen and J. Lampinen. On the generative probability density model in the self-organizing map. *Neurocomputing*, 48:217–228, October 2002.

[18] J. Laaksonen, K. Koskela, S. Laakso, and E. Oja. Self-organizing maps as a relevance feedback technique in content-based image retrieval. *Pattern Analysis and Applications*, 4(2+3):140–152, June 2001.

[19] G. J. McLachlan and D. Peel. *Finite mixture models*. John Wiley & Sons, 2000.

[20] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M.I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer, 1998.

[21] S. Negri and L. Belanche. Heterogeneous Kohonen networks. In *Connectionist Models of Neurons, Learning Processes and Artificial Intelligence : 6th Int. Work-Conference on Artificial and Natural Neural Networks*, volume 2084 of *Lecture Notes in Computer Science*, pages 243–252. Springer-Verlag, 2001.

[22] D. De Ridder, J. Kittler, O. Lemmers, and R. P. W. Duin. The adaptive subspace map for texture segmentation. In A. Sanfeliu, J. J. Villanueve, M. Vanrell, R. Alquezar, J. O. Eklundh, and Y. Aloimonos, editors, *Proceedings of the 15th IAPR International Conference on Pattern Recognition*, pages 216–220. IEEE Computer Society Press, 2000.

[23] S.T. Roweis, L.K. Saul, and G.E. Hinton. Global coordination of local linear models. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.

[24] M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482, 1999.

[25] A. Utsugi. Density estimation by mixture models with smoothing priors. *Neural Computation*, 10(8):2115–2135, 1998.

[26] J. J. Verbeek, N. Vlassis, and B. J. A. Kröse. Coordinating principal component analyzers. In J.R. Dorronsoro, editor, *Proc. Int. Conf. on Artificial Neural Networks*, pages 914–919, Madrid, Spain, 2002. Springer.

[27] J. J. Verbeek, N. Vlassis, and B. J. A. Kröse. Self-organization by optimizing free-energy. In M. Verleysen, editor, *Proc. of European Symposium on Artificial Neural Networks*. D-side, Evere, Belgium, 2003.

[28] J. J. Verbeek, N. Vlassis, and J. R. J. Nunnink. A variational EM algorithm for large-scale mixture modeling. In *Proc. 8th Ann. Conf. of the Advanced School for Computing and Imaging (ASCI 2003)*, Het Heijderbos, Heijen, 2003.