Deterministic Generative Models for Fast Feature Discovery

Machiel Westerdijk (machiel@mbfys.kun.nl), David Barber
(barberd@aston.ac.uk) and Wim Wiegerinck (wimw@mbfys.kun.nl)
Department of Medical Physics and Biophysics, University of Nijmegen, The Netherl ands

Abstract. We propose a vector quantisation method which does not only provide a compact description of data vectors in terms codebook vectors, but also gives an explanation of codebook vectors as binary combinations of elementary features. This corresponds to the intuitive notion that, in the real world, patterns can be usefully thought of as being constructed by compositions from simpler features. The model can be understood as a generative model, in which the codebook vector is generated by a hidden binary state vector. The model is non-probabilistic in the sense that it assigns each data vector to a single codebook vector. We describe exact and approximate learning algorithms for learning deterministic feature representations. In contrast to probabilistic models, the deterministic approach allows the use of message propagation algorithms within the learning scheme. These are compared with standard mean-field/Gibbs sampling learning. We show that Generative Vector Quantisation gives a good performance in large scale real world tasks like image compression and handwritten digit analysis with up to 400 data dimensions.

Keywords: vector quantisation, feature discovery, generative models, mean-field methods, message passing algorithms, handwritten digits analysis, image compression

1. Introduction

Many techniques for data analysis can be regarded as seeking for a description of data in terms of elementary features. An advantage of a feature representation is that it reduces redundancy in the input patterns (Barlow, 1989). Furthermore, a description in terms of features can provide a lucid explanation of objects (input patterns), which can in addition be helpful in understanding the hidden data generating process. Areas in which feature representations are particularly relevant can be found in biological modelling, image processing and data mining.

Currently, the most widely applied techniques for feature extraction are linear. Well known examples are principal component and factor analysis. Both these techniques give a meaningful representation of the data only if the data are Gaussian distributed around some low dimensional linear subspace. More recent non-Gaussian linear methods include independent component analysis (Bell, 1995) and the sparse coding approach by (Olshausen, 1996). A significant advantage of linear methods is their speed. In addition, linear models provide an easily interpretable feature representation of the data, often in terms of the basis spanning the linear subspace. One important drawback of linear models is that they can not describe multi-modal distributions.

The most well known and simplest method for finding multi-modal structure in the data is vector quantisation (VQ) (Gray, 1984). The drawback of vector quantisation, however, is its lack of a feature representation. To overcome this problem, more advanced non-linear probability models have recently been promoted by several authors in the context of feature extraction (Sallans, 1998; Ghahramani, 1998; Saul, 1996; Attias, 1999). In contrast to standard vector quantisation, where a data point is explained in terms of a single codevector, these models explain a data point in terms of a *combination* of elementary features. Each such combination is formed or generated by the state of a set of *hidden* or latent variables.

The model that we propose in this paper, the Generative Vector Quantizer (GVQ), is exactly such a generative model, with a binary hidden layer and a continuous visible layer representing the codebook vectors. Hence, in GVQ a codebook vector is considered to be composed of a binary combination of features in which a given feature is either fully present or fully absent. To provide an easy data interpretation, GVQ associates only a single codebook vector and therefore a single feature composition to each data pattern. This is in contrast with probabilistic models which associate a data pattern with a distribution over compositions where, in principle, each possible composition has a contribution. In this sense, one can view the GVQ model as a special deterministic variant within the larger class of noisy, probabilistic models.

In addition to interpretability, there is another important advantage of using a deterministic model. To learn a generative model from a given data set there exists an accurate and rapidly converging algorithm. This EM-algorithm (Dempster, Laird & Rubin, 1977) iterates between an Expectation step (E-step) and a Minimisation step (M-step). The E-step determines which hidden states (which combinations of features) are responsible for generating a given data pattern. If these states are known then it is computationally straightforward to optimize the feature values in the M-step. The basic problem in the application of the EM-algorithm is that there is no efficient way to determine which of the, exponentially many, states generated the data point. This makes the E-step computationally intractable. A major issue in developing learning methods for Generative models is to find accurate and tractable approximate solutions for the E-step. In data-mining applications, where databases are often large and high dimensional, this issue becomes particularly important. The deterministic property of the GVQ model makes the use of a special class of message passing algorithms within the learning scheme directly relevant. These algorithms (Pearl, 1988) are used within graphical models to infer marginal probabilities given some evidence. In the deterministic approach the distribution of a multi-dimensional state space is given simply by the product of the marginals of this distribution. In other words, in the deterministic limit, algorithms which estimate marginal 2probabilities well, will necessarily estimate the full distribution equally well.

Message passing algorithms are interesting alternatives to the methods for probabilistic models such as the mean-field approximation (Ghahramani, 1995; Zemel, 1994; Saul, 1996). In this paper we will describe how message passing algorithms can be used for learning feature representations in the form of a GVQ model. In addition, we will present an extensive comparison between these algorithms and the mean-field method for learning deterministic GVQ models. We will indicate under which circumstances a specific algorithm should be preferred over others.

In section (2) and section (3) we present the basic idea of GVQ and its relationship to standard vector quantisation, along with the GVQ learning algorithm. The crucial issue of the tractable implementation of this algorithm is discussed at some length in section (4).

In order to tune the representation for a particular application there are some useful types of constraints one can impose on the model. In some applications one knows that are certain distinct classes present in the data. For example in handwritten digit analysis the features for constructing 2's are not used for constructing 4's which have their own distinct set of features. Furthermore, to learn a multiple feature set model it is desirable that the sets compete in a winner-take-all fashion, so that the sets force each other to specialize on different structures in the data. We will show in section (5) that in the GVQ model such a multiple set representation can be built in in a natural way.

High dimensional real world problems, namely handwritten digits and image compression, are studied in sections 6.1 and 6.2. The relation of this work to other models is discussed in section (7), along with potential benefits to discrete optimisation using message passing schemes.

2. Standard Vector Quantisation

The aim of vector quantisation (VQ) is to represent a dataset by a smaller set of representative vectors. This set can be used to code a data pattern with a small number of bits. The code of a data pattern is given by the index of the closest representative vector. For this reason the collection of representative vectors is called a codebook and the vectors themselves are the codebook vectors. An example of a data set with its representative codebook vectors is shown in fig(1) (a).

An important advantage of standard vector quantisation methods is that, for a given data set, they can quickly construct a set of representative codebook vectors. For this reason these techniques are widely used in many application domains for compression or for clustering of data. Standard vector quantisation does not, however, represent objects as a collection of features. To overcome this deficiency, we introduce generative vector quantisation, as described in the following section.

3. Generative Vector Quantisation

In Generative Vector Quantisation (GVQ) the objective is similar to that of VQ, namely to find a codebook representation of the data. In contrast to standard vector quantisation, GVQ reduces the number of representative vectors by using a smaller set of basic feature vectors $\{\mathbf{f}^1, \ldots, \mathbf{f}^n\}$ which exist in the same d-dimensional space as the data. Each codebook vector is then formed by some binary combination of these feature vectors,

$$\sum_{i=1}^{n} s_i \mathbf{f}^i \equiv F\mathbf{s},$$

where the feature matrix $F = \left[\mathbf{f}^1 \mathbf{f}^2 \ldots \mathbf{f}^n\right]$, and the state vector $\mathbf{s} \in \{0,1\}^n$. There are therefore $M = 2^n$ possible codebook vectors $F\mathbf{s}^1, \ldots, F\mathbf{s}^M$.

An example of a set of codebook vectors generated by 3 features in a 2-dimensional space is shown in fig(1) (b). To contrast this approach with the standard approach, the data used for GVQ is the same as in fig(1) (a). In fig(1)b), the circles represent the generated codebook vectors which correspond to the 8 states (000), (100), ... , (111). The features are given by the codebook vectors corresponding to the unary state vectors (100), (010) and (001) etc. The zero state vector is the origin of the representation. The remaining codebook vectors are then ‘generated’ by combinations of these basic codebook vectors, or features. For example, the codebook vector corresponding to state (011) is given by adding the features corresponding to state (010) and (001), see fig(1) (b).

Note that in GVQ the number of features n is not related to the dimensionality of the data space. Hence, there may be more or less basic feature vectors than there are dimensions in the data space.

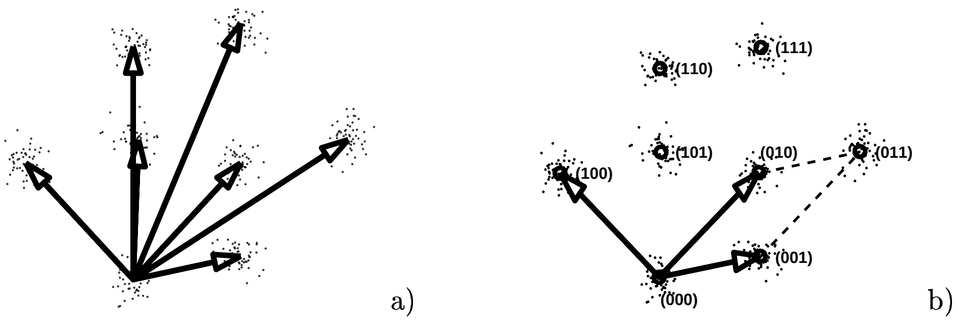

Figure 1. a) A codebook vector representation by standard vector quantisation. b) In GVQ the codebook vectors (circles) are generated by a small set of basic features $\mathbf{f}_1, \mathbf{f}_2$ and $\mathbf{f}_3$, which correspond to the states (100), (010) and (001), respectively. The codebook vector corresponding to the state (011), for example, is given by the sum of the vectors corresponding to the two states (001) and 010 (see broken lines).

3.1. The GVQ learning algorithm

In GVQ, as in standard VQ, each data point $\mathbf{x}^\mu$ is associated with a particular codebook vector, indexed by c_μ. Typically, this association is made such that $\mathbf{x}^\mu$ is assigned to the closest codebook vector, in the Euclidian sense. The squared Euclidian distance between the whole data set $\mathcal{D} = \{\mathbf{x}^\mu | \mu = 1, \ldots, P\}$ and its codebook representation, $\{F\mathbf{s}^{c_\mu} | \mu = 1, \ldots, P\}$, is

$$E = \sum_{\mu=1}^{P} \|\mathbf{x}^\mu - F\mathbf{s}^{c_\mu}\|^2. \tag{1}$$

The task is, therefore, to find both the optimal associations of data points to codebook vectors, and the best feature vectors in order to minimize E. Since the associations between data points and codebook vectors will change if the feature matrix F is changed, minimising (1) directly with respect to F and the associations is not practical. For this reason we make use of a two-step iteration procedure.

After initialisation of the features F the GVQ learning algorithm iterates between an association step 1 which finds, for each data-point, the most nearby codebook vector and a minimisation step 2 which finds the optimal feature configuration for the given association:

1. For $\mu = 1, \ldots, P$

$$c_\mu \leftarrow \underset{j}{\operatorname{argmin}} \|\mathbf{x}^\mu - F\mathbf{s}^j\|^2, \tag{2}$$

2.

$$F \leftarrow \underset{\tilde{F}}{\operatorname{argmin}} \sum_\mu \|\mathbf{x}^\mu - \tilde{F}\mathbf{s}^{c_\mu}\|^2 \tag{3}$$

The second step only involves the mean $\langle \mathbf{x} \rangle_k$ of each group of data associated with a single codebook vector k since,

$$\begin{aligned}
&\min_F \textstyle\sum_k \displaystyle\sum_{\mu \in C_k} \|\mathbf{x}^\mu - F\mathbf{s}^{c_k}\|^2 = \\
&\min_F \textstyle\sum_k \displaystyle\sum_{\mu \in C_k} \left[\|F\mathbf{s}^{c_k}\|^2 - 2\mathbf{x}^\mu F\mathbf{s}^{c_k}\right] = \\
&\min_F \textstyle\sum_k \displaystyle\sum_{\mu \in C_k} \left[\|F\mathbf{s}^{c_k}\|^2 - 2\mathbf{x}^\mu F\mathbf{s}^{c_k}\right] + N_k \langle \mathbf{x} \rangle_k{}^2 = \\
&\min_F \textstyle\sum_k \displaystyle N_k \|\langle \mathbf{x} \rangle_k - F\mathbf{s}^{c_k}\|^2, \qquad (4)
\end{aligned}$$

where $\mathcal{C}_k$ represents the set of all data points associated with the state $\mathbf{s}^{c_k}$ and where N_k is the number of data points in cluster k. The expectation value $\langle \cdot \rangle_k$ is taken over the data in cluster $\mathcal{C}_k$. The additive constant does not depend on F. The objective function (4) can be minimized efficiently by use of Singular Value Decomposition (see for example Press, 1992, chapter 14.3).

The first step, (2), is computationally more difficult since, in principle, it involves a search through all 2^n binary states $\mathbf{s}$. In section (4) we will discuss and compare different approximate algorithms which can reduce this computational overhead.

4. Approximate association

In the association step (2) we want, for a given fixed data point $\mathbf{x}$, to minimize the error function[1]

$$E(\mathbf{s}, \mathbf{x}) = \|\mathbf{x} - \sum_i \mathbf{f}_i s_i\|^2 \qquad (5)$$

with respect to $\mathbf{s}$. Since there is an exponential number of binary states $\mathbf{s}$, an exhaustive search over all these states rapidly becomes computationally intractable for even a moderate number of features.

In this section we will compare two types of approach for finding the optimal state $\mathbf{s}^*$ which minimizes (5)[2], drawing heavily on the terminology of graphical models (Pearl, 1988; Neal, 1998). In doing so we make a distinction between two types of methods. The first class of methods only considers relations, implicitly given by (5), between the binary variables[3] S_i. As will be explained the relationships between the binary variables can, in that case, be represented by an *undirected* graph. In section (4.1) we will present a number of specialized approximating algorithms for undirected graphical structures.

The second class of methods considers explicitly the relations between binary variables S_i and visible variables X_i. The corresponding graphical dependency structure is then *directed.* Section (4.2) discusses an algorithm which exploits this graphical structure.

[1] In this section we refer, for notational convenience, directly to a specific binary state $\mathbf{s}$ and omit the upper indices used in section (3.1). Furthermore, we do the association for a single data point $\mathbf{x}$. It is clear that the problem is the same if we instead use the cluster means $\langle \mathbf{x} \rangle$ of (4).

[2] Note, that the minimizing state $\mathbf{s}^*$ need not be unique. Here we do not specify a prior preference *i.e.* we regard each solution to be equally valid.

[3] Note, that we use capitals to refer to variables and lowercase letters to refer to their values.

In section (4.3) we shall give an experimental comparison between these methods for a range of GVQ architectures.

4.1. Undirected graph methods

Since we are interested in finding the state $\mathbf{s}$ which minimizes the error function $E(\mathbf{s},\mathbf{x})$ for a given $\mathbf{x}$, we can define a new error function which contains only dependencies on $\mathbf{s}$. First note that,

$$\begin{aligned} E(\mathbf{x},\mathbf{s}) &= \mathbf{x}^2 - 2\mathbf{x}\sum_i f_i s_i + \sum_{ij} \mathbf{f}_i \cdot \mathbf{f}_j s_i s_j \\ &= \mathbf{x}^2 - 2\mathbf{x}\sum_i f_i s_i + \sum_i \mathbf{f}_i^2 s_i^2 + 2\sum_i \sum_{j>i} \mathbf{f}_i \cdot \mathbf{f}_j s_i s_j, \end{aligned} \tag{6}$$

and that $s_i = s_i^2$. The new $\mathbf{s}$-dependent error $E_\mathbf{x}(\mathbf{s})$ is defined as

$$E_\mathbf{x}(\mathbf{s}) = \sum_i \left\{ h_i s_i + \sum_i \sum_{j>i} w_{ij} s_i s_j \right\}, \tag{7}$$

where $w_{ij} = 2\mathbf{f}_i \cdot \mathbf{f}_j$, $h_i = -2\mathbf{f}_i \cdot \mathbf{x} + \mathbf{f}_i^2$. The $\mathbf{s}$ that minimizes $E_\mathbf{x}(\mathbf{s})$ is equal to the $\mathbf{s}$ that minimizes $E(\mathbf{s},\mathbf{x})$. Note that this error function contains only pairwise and symmetric dependencies between the variables s_i. This dependency structure, given by the weight matrix w_{ij}, can be represented as an undirected graph. An example of a fully connected graph, i.e. all weights w_{ij} are non-zero, is shown in fig(2).

The following subsections discuss three different algorithms which make use of this undirected graph structure.

4.1.1. *Belief Propagation (BP)*

The error function (7) can be used to define a probability distribution $p_\mathbf{x}(\mathbf{s})$ on the set of binary states $\mathbf{s}$,

$$p_\mathbf{x}(\mathbf{s}) = \frac{1}{Z_\mathbf{x}} \exp\left(-\frac{1}{2\sigma^2} E_\mathbf{x}(\mathbf{s})\right), \tag{8}$$

where $Z_\mathbf{x}$ is a normalization constant. In this formulation the state $\mathbf{s}^* \equiv \underset{\mathbf{s}}{\text{argmin}} E_\mathbf{x}(\mathbf{s})$ with the smallest error in (7) now corresponds to the state with the largest probability in the distribution $p_\mathbf{x}(\cdot)$. In the case that the noise σ in (8) is decreased, $\mathbf{s}^*$ will start to dominate the distribution. In the limit $\sigma \to 0$ the corresponding probability $p_\mathbf{x}(\mathbf{s}^*)$ saturates to the value 1. It is easy to see that the *marginal probabilities* $p_\mathbf{x}(s_i)$, i.e. probabilities of individual units, also saturate to the values 0 or 1. A useful property of the zero noise limit is that the single unit states s_i^* for which $p_\mathbf{x}(s_i^*) \to 1$ together form the global objective state $\mathbf{s}^* = (s_1^*, s_2^*, \dots, s_n^*)$ of the whole graph. Hence, by computing the marginals $p_\mathbf{x}(s_i)$ from (8) we can, by reducing σ in (8), find the minimising state $\mathbf{s}^*$. However, computation of the marginals $p_\mathbf{x}(s_i)$ has the same computational complexity as the original minimisation problem (7) since it involves a summation over all combinations of the states of all the other units $j \neq i$.

The belief propagation algorithm (Pearl, 1988) provides a computationally inexpensive approximate method to compute the marginals $p_\mathbf{x}(s_i)$. The basic idea of this method is to decompose the summation into a sequence of local operations which take place at the individual nodes. In appendix (A) we will present this method and give an example for a simple network. The computational complexity of this algorithm scales quadratically $\mathcal{O}(n^2)$ with the number of binary nodes n.

Our implementation of belief propagation starts with a large noise value σ. While running Belief Propagation, in an attempt to avoid local minima, the noise σ is slowly reduced to zero. The corresponding state $\mathbf{s}^*$ for which all $p_{\mathbf{x}}(s_i^*) = 1$ is then taken as the solution for our minimisation problem.

4.1.2. *Belief Revision (BR)*

Belief Revision (Pearl, 1988) is an algorithm for directly tackling the minimisation problem $\min_{\mathbf{s}} E_{\mathbf{x}}(\mathbf{s})$. Again, the trick is to carefully exploit the graphical structure of the problem, given by the weights w_{ij}, in order to decompose the minimisation problem into local operations. In fact, it can be shown that the belief propagation algorithm is equivalent to Belief revision by taking the limit $\sigma \to 0$. We refer to appendix (B) for a derivation of the Belief Revision algorithm and a description of our implementation of the algorithm for the GVQ model. The computational complexity of this algorithm is $\mathcal{O}(n^2)$.

4.1.3. *Mean-Field (MF)*

The basic idea of variational algorithms (of which the mean-field method is a special case) is to replace the intractable objective function with a tractable approximation to it, so that the optimization of the approximate objective function can be carried out efficiently, see for example (Saul, 1996). Based on this principle we derive a mean-field variational algorithm in appendix (C) to find the minimising $\mathbf{s}^*$ state of the objective function (7). The complexity of the mean-field algorithm is $\mathcal{O}(n^2)$.

4.2. Approximation in the Directed Graph

Instead of representing only the relation in each pair ij of binary variables S_i and S_j as in the previous section, we can also form a graphical representation of the relation in each pair of *all* the variables *i.e.* binary variables S_i *as well* as continuous variables X_i. The most efficient way to do this is to represent the relations with a directed graph.

To explain this we represent our GVQ model as a joint probability model $p(\mathbf{x}, \mathbf{s})$ of binary states $\mathbf{s}$ and visible states $\mathbf{x}$. The prior distribution of the binary units $p(\mathbf{s})$ is constant, i.e. $p(s_i) = \frac{1}{2}$ and $p(\mathbf{s}) = \frac{1}{2^n}$. The joint probability distribution can be constructed as follows:

$$p(\mathbf{x}, \mathbf{s}) = p(\mathbf{s})p(\mathbf{x}|\mathbf{s}) = 2^{-n}\left(2\pi\sigma^2\right)^{-\frac{d}{2}} \exp\left\{-\frac{1}{2\sigma^2}E(\mathbf{s}, \mathbf{x})\right\}, \tag{9}$$

where $E(\mathbf{s}, \mathbf{x})$ is our original objective function (5). An example, of the graphical representation of (9) is shown in fig(2). As can be seen, there are no direct links between binary hidden units reflecting the fact that the prior distribution $p(\mathbf{s})$ is factorized, i.e. $p(\mathbf{s}) = \prod p(s_i)$. The arrows reflect the relation between hidden states $\mathbf{s}$ and visible states $\mathbf{x}$ given by $p(\mathbf{x}|\mathbf{s}) \propto \exp\left\{-(\mathbf{x} - \sum_i \mathbf{f}_i s_i)^2/2\sigma^2\right\}$ so that the set of arrows expanding from unit S_i correspond to feature $\mathbf{f}_i$.

4.2.1. *Belief Propagation in the Directed Graph (DBP)*

In appendix (D) we present a belief propagation algorithm which explicitly takes the directed graphical structure into account by passing messages from hidden units to visible units and vice versa. Using the same noise reduction process

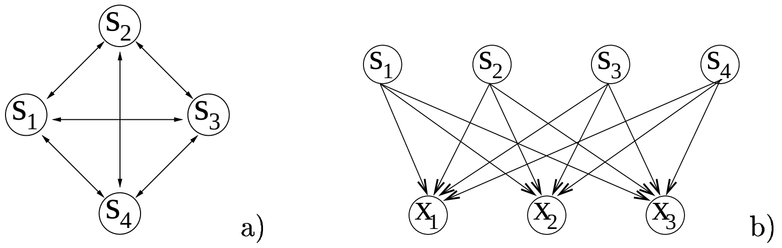

Figure 2. a) Undirected graph representation. b)Directed Graphical structure of GVQ

$\sigma \to 0$ as in section (4.1.1), the desired state $\mathbf{s}^*$ can be inferred by computing the conditional probabilities $p(s_i|\mathbf{x})$ from (9). Again, in the limit $\sigma \to 0$ the marginal probabilities $p(s_i^*|\mathbf{x}) \to 1$ together form our desired objective state $\mathbf{s}^* = (s_1^*, s_2^*, \dots, s_n^*)$. For general probability distributions with bi-partite structures as in fig(2) the computational complexity of this algorithm scales exponentially with the number of connections between a visible unit X_i and hidden units S, that is, the algorithm is exponentially complex in the number of parents of the visible units. However, for the special case of our GVQ model, which has a quadratic dependence between binary states $\mathbf{s}$ and visible states $\mathbf{x}$, we can reduce this complexity to $\mathcal{O}(n^2 \times d)$, by introducing an integral transform. However, this is potentially at the cost of decreased accuracy[4].

4.2.2. *Belief Revision in the Directed Graph (DBR)*

In section (4.1.2) we transformed the probabilistic belief propagation algorithm into a noiseless algorithm by taking the limit $\sigma \to 0$. The same operation can be applied on the DBP algorithm for the directed graph of section (4.2.1). The resulting belief revision algorithm still takes the directed graphical structure into account. However, in contrast to the probabilistic algorithm, the complexity of the algorithm can no longer be reduced to polynomial. It remains exponential in the number k of connections that each visible unit has with the binary units *i.e.* the overall complexity is $\mathcal{O}(n \times d \times 2^k)$. We refer to appendix (E) for a more technical discussion of this approach.

4.3. Experimental comparison

So far we have not characterized the type of problems for which we can find a sensible GVQ representation. In practice we can expect a large range of situations where we want to find a feature representation. For example, some situations require a large number of features in a low dimensional data space (over-complete basis) or, in the opposite case, they require a few nearly orthogonal features in a high dimensional space. The purpose of this section is to determine under which circumstances the approximating algorithms are most suitable.

4.3.1. *Influence of connectivity structure*

In this sub-section we monitor the performance of the algorithms if we gradually increase the complexity of the graphical structure of the GVQ model. To do this, we generated a number of artificial problems. In each experiment we sampled a fixed number $n = 12$ hidden units of $d = 4$ dimensional features (visible units).

[4] As explained in appendix (D).

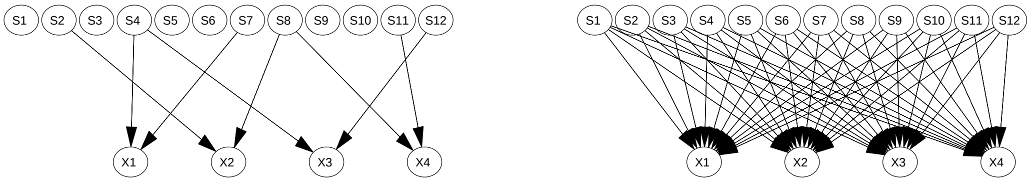

Figure 3. a) GVQ networks with $n = 12$ features and $d = 4$ visible units. a) Each visible unit X_i is connected with only two binary parent units ($k = 2$). b) Fully connected network ($k = n$).

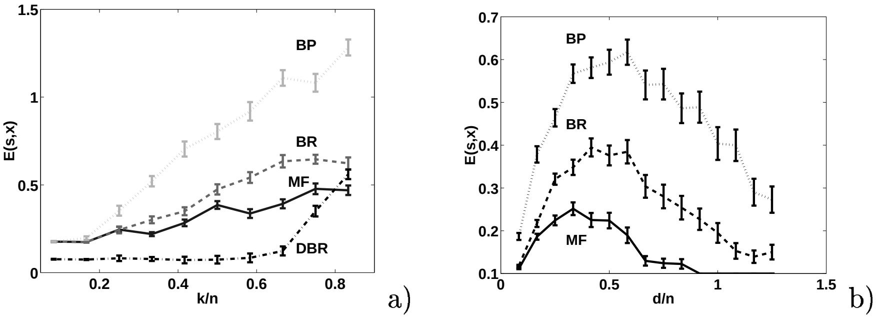

Figure 4. Comparison of the Belief Propagation (BP, dotted lines), the Belief Revision (BR, dashed lines), Mean-Field (MF, solid lines) and the Belief Revision in the Directed graph (DBR, dot-dash lines) algorithm. a) The error $E(\mathbf{s}, \mathbf{x})$ as a function of the number of parent connections k (given as a fraction k/n of the total number of parents n). b) The error $E(\mathbf{s}, \mathbf{x})$ as a function of the number of input dimensions d (given as a fraction d/n of the total number of parents n).

The features $\mathbf{f}_i$ are chosen such that each visible node X_i has at most k connections with the binary layer, see fig(3). The connections are selected randomly. The feature values f_{ij} that determine the strength of these connections are sampled according to $f_{ij} \sim \mathcal{N}(2, 1)$, a Gaussian distribution with mean 2 and variance 1. We chose a non-zero mean to avoid non-realistic symmetries in the generated data. Together with each feature set we randomly chose a binary state $\mathbf{s}^*$, according to $p(s_i^*) = 0.5$. Then for fixed $\mathbf{f}_i$ and $\mathbf{s}^*$ we generated an input pattern $\mathbf{x}$ using $\mathbf{x} = \sum_i \mathbf{f}_i s_i^* + \epsilon$, where ϵ is adding a small amount of random noise. The components of ϵ are sampled from $\mathcal{N}\left(0, \frac{1}{10}\right)$. Given the input pattern $\mathbf{x}$, each method was used to recover the generating state $\mathbf{s}^*$. We then computed the error E which is defined here as the average absolute error per input [5] dimension *i.e.* $E = \frac{1}{d} \sum_i |x_i - \sum_j f_{ij} s_j|$ which is directly related to (5). Note that the minimum error is ϵ. This procedure was repeated 100 times for each connectivity number k. Fig(4) (a) shows the average of these results as a function of the number of connections k. For each method the error clearly increases with increasing number

[5] We look at the input space and not at errors in the binary latent space since we are interested in the reconstruction errors of data examples. Two codebook vectors with the same distance to data point $\mathbf{x}$ but with a large distance to each other in the binary latent space binary space are considered equally valid.

of connections k. If the number of connections is small ($k/n < 0.6$) the error of the directed belief revision algorithm is close to the minimum error ϵ *i.e.* it is close to the exact solution. In this region DBR outperforms all undirected algorithms. At a certain point ($k/n \approx 0.65$) the error of DBR starts to rise quickly. This is expected since the loops are shorter in denser networks. The transition point is somewhat dependent on the imposed noise ϵ. For smaller values of ϵ the point shifts to the right. In the extreme case $\epsilon = 0$ DBR does not make errors anymore, *i.e.* $E(\mathbf{s}, \mathbf{x}) = 0$ for $\epsilon = 0$ (which is not true for the other methods). However, for the value of ϵ we used here the DBR algorithm performs poorly for fully connected networks. In that case we should use the mean-field algorithm.

In fig(4) we do not include the performance of DBP. To obtain comparable performance to DBR, we found that we needed to anneal σ to such a small level that retaining accuracy of the integral transform became computationally burdensome.

As can be seen from fig(4), the error of BP is in all cases larger than the BR error. To get a BP error closer to BR we need to anneal to even smaller values of σ. The anticipated positive effect of avoiding local minima was not present.

In fig(4) (b) we show the result for fully connected networks ($n = 15$, $f_{ij} \sim \mathcal{N}(0, 1)$) where we increase the number of input dimensions d. For the fully connected case we omitted the DBR and DBP methods since we know, from the previous experiment, that their performances will be poor. We see that for these fully connected GVQ networks mean-field outperforms the other methods over the whole region. For large values of d/n the thresholds h_i will dominate the contributions given by the interactions w_{ij} in the error function (7). Hence, effectively the units S_i will become more independent in which case all methods perform better explaining the decreasing errors in fig(4) (a).

In the experiments with the BR method we made an interesting observation. In *all* the trials where the messages converged to a stationary value the final error was equal to the minimum error ϵ. The BR error in fig(4) comes from the remaining non-converging trials. This indicates that by looking at the convergence behaviour of BR we are able to determine whether the final answer is correct. If it does not converge we can always do the association with MF instead.

4.3.2. *Performance as a function of the similarity between the features*

In the experiments above the features were sampled from zero mean normal distributions. This may not be particularly representative of features in real-world problems. A crude attempt to address this issue is given by generating features which have a degree of similarity. Here we do this by drawing the features from a Gaussian with non-zero mean. The feature values f_{ij} are sampled from a normal distribution with mean γ, that is $f_{ij} \sim \mathcal{N}(\gamma, 1)$. Hence, the larger γ the more the features $\mathbf{f}_i$ 'point in the same direction'. The result is shown in fig(5). We see that if $\gamma > 4.5$ the message passing methods BR and DBP both outperform MF. In the previous experiments, the mean-field method performed better for fully connected architectures. However, the results in fig(5) indicate that under certain circumstances message passing algorithms may outperform mean-field also for fully connected networks.

5. Imposing constraints on the basic model

In some applications one has a priori knowledge about the probability distribution which generated the data. In that case it can be of great help to impose

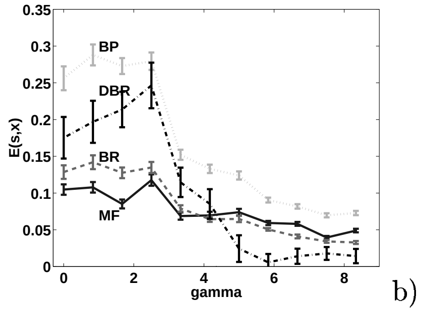

Figure 5. Comparison of the Belief Propagation (BP, dotted line), the Belief Revision (BR, dashed line) and Mean-Field algorithm (MF, solid line). The error is shown $E(\mathbf{s}, \mathbf{x})$ as a function of the similarity γ between the features ($d = 4; n = 10$). The error bars indicate the variation in the mean.

constraints on the basic GVQ model which incorporate this knowledge. In this section we propose three constraining methods which we believe to be useful for a large class of applications. The first two of these methods impose constraints on the distribution of the hidden states $\mathbf{s}$. The third represents a constraint on the distribution of values of the input patterns $\mathbf{x}$.

5.1. Multiple feature sets and multi-valued features

In many real world problems we can expect that there are different unrelated groups of patterns. In each of these groups the patterns are built up out of features from a set which is specific for the group. It is easy to adjust the GVQ learning algorithm for learning multiple feature sets by simply constraining the set of allowable states. Fig(6) shows the result of learning two sets of 3 features from a data set of 300 samples. Features corresponding to one set can only be combined with features from the same set. The origins of the different sets, indicated with the dashed lines in fig(6), which are considered as constant ‘on’ features for that set, are also determined by the optimization process. For a given data point there is only one feature set responsible for generating the closest code vector. In this sense we can interpret the multiple set model as a winner-take-all configuration of multiple GVQ’s.

Allowing multiple sets makes it possible to find different groups of objects, i.e. each GVQ within the winner-take-all configuration learns to represent a certain class of objects in an un-supervised manner. An example of this is given in section 6.1 in which handwritten 3’s and 5’s are separated in an un-supervised manner using multiple feature sets. Note that in the extreme case of using one feature per set, GVQ is equivalent to standard vector quantisation.

5.2. Penalty constraints

Another way of biasing the solution to those consistent with prior beliefs is given by adding an extra penalty term to the energy function (1). For example, one can bias the final representation to be *sparse*, i.e. each object is composed of a small number of features from a large set, by adding for example the term $\lambda \mathbf{s}^T \mathbf{s}$ to the energy function.

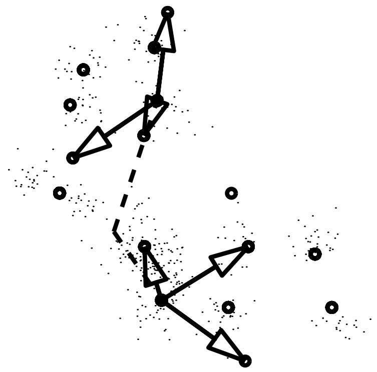

Figure 6. Result of learning two sets of 3 features using a data set of 300 samples from a mixture of 10 randomly distributed Gaussian clusters . The 'origin' features are represented by the dashed lines which come together in the origin.

5.3. Binary features for binary data

In the case that the elements of the data are binary, it is desirable that the feature combinations result in values close to 1 and 0. Experiments with the original error function (1) on this type of data, however, result in feature combinations with smoothly varying values, i.e. the reconstructed data patterns are not binary patterns but have values between the binary states.

Better results are obtained by using sigmoid squashing functions so that the codebook vectors are forced to have values close to 0 and 1. For this reason we use a sigmoid function to 'squash' the combination of features. For ease of interpretation, it is also advantageous that the features themselves are constrained to be binary. This can be implemented in a 'soft' manner by defining the constrained features as $F \equiv \sigma(\hat{F})$ such that we can minimize the error function with respect to the unconstrained matrix $\hat{F}$. Our error function for binary data is thus

$$E = \sum_{\mu=1}^{P} \|\mathbf{x}^{\mu} - \sigma_{\beta}\left(\sigma\left(\hat{F}\right)\mathbf{s}^{c_{\mu}}\right)\|^{2} \tag{10}$$

in which

$$\sigma(\mathbf{x}) = \left(1 + e^{-\mathbf{x}}\right)^{-1},$$

and

$$\sigma_{\beta}(\mathbf{y}) = \sigma(\beta(\mathbf{y} - 1/2)),$$

where β is a parameter which controls the steepness of the squashing function. In our implementation (10) is minimized with respect to $\hat{F}$ using the scaled conjugate gradient method (Press, 1992).

After incorporating the squashing functions, the model becomes more closely related to other models such as sigmoid belief networks (Neal, 1991) and the "multiple cause mixture" representation proposed by (Saund, 1995). Note, however, that the deterministic approach, see section (7.1), and the learning procedure, discussed in section (4), of GVQ sets it apart from these methods.

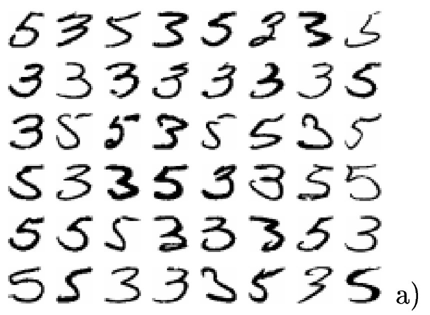

Figure 7. A random sample of 48 handwritten 'threes' and 'fives' from the CEDAR CDROM. Each image consists of 20 × 20 bits.

6. Results on Real-World Data

In this section we demonstrate the application of GVQ in two practical situations. First we extract features of handwritten digits. Using a small number of basic features GVQ can find nice reconstructions of the original digits. Finding a feature representation can for example be useful as a pre-processing step in a classifier. In the second application we demonstrate the advantage of GVQ over standard vector quantization in image compression. We show that when using a feature representation images can be compressed into an even smaller number of data bits.

6.1. Handwritten digits

We randomly selected 400 training images of handwritten 'threes' and 'fives' from the CEDAR CDROM 1 database (Hull, 1994). Since the original images contain different numbers of pixels, we rescaled all images to 20 × 20 pixels. A typical sample of these images is shown in fig(7).

We decided to fit a GVQ model consisting of 4 mutually exclusive sets of 5 features (including in each set an origin feature), see section (5.1). For this application we made use of the binary feature binary data version of GVQ as discussed in section (5.3) with $\beta = 4.5$. The features, which were obtained, are shown in fig(8). Each row in the figure corresponds to a feature set and the last feature on the right hand side in each row corresponds to the origin feature of the set. By inspection it is clear that the first two sets (top two rows) specialize on 'threes' whereas the last set (last row) specializes on constructing 'fives'. The third set can construct both 'fives' and 'threes'. These properties become more clear if we look at fig(9) where 37 of the most representative feature combinations (codebook vectors) are shown. By most representative we mean those feature combinations (codebook vectors) which account for most of the data. The codebook vectors in the left sub-figure of fig(9) are combinations of features from the first set in fig(8), which clearly are all 'threes'. As we see from the second sub-figure in fig(9), the second feature set, although primarily concerned with modeling 'threes', is nevertheless able to construct a 'five'. This is even more apparent in the third sub-figure in fig(9) containing both 'threes' and 'fives'. The reconstructions in this set show that there is a class of handwritten digits containing 'threes' and 'fives' which share at least one feature, namely the origin feature. This origin feature can be supplemented with an additional feature to become either a 'three' or a 'five'.

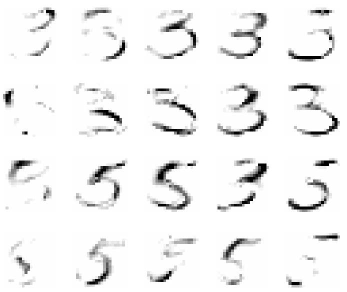

Figure 8. Features that were obtained after learning from a database of 400 handwritten 'threes' and 'fives'. Each of the 4 rows represents an independent set which consists of 4 features and an 'origin' feature (most right in each row).

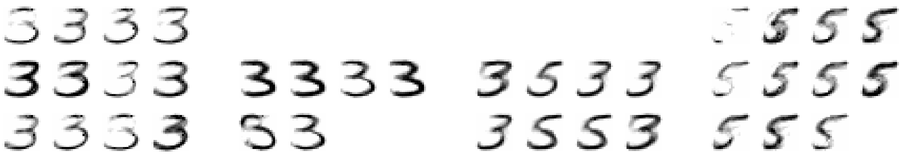

Figure 9. The 37 most representative feature combinations. Each group corresponds to reconstructions using features from the same set, given by the rows in fig(8).

6.2. Image Compression

A well known application of vector quantisation is in image compression. In this section we demonstrate the additional advantage for image compression gained by describing codebook vectors in terms of a small number of features.

As an example, we used GVQ to compress the image in fig(10)a)[6] and compared the result with standard vector quantisation. The original image consists of 768 × 704 pixels with 256 possible gray levels for each pixel which corresponds to 865 kbits of information. The image was split into $P = 2112$ segments of 16×16 pixels. We used standard VQ to construct 16 codebook vectors to represent this set of P segments. We then reconstructed the image using the closest codebook vector to each image segment. The result is shown in fig(10)-b. We also applied standard GVQ using $n = 8$ features (plus an additional origin feature) to construct a representative set of codebook vectors for the image segments. Fig(10)-c shows the reconstructed image. The superior performance of GVQ over VQ in representational accuracy is given by the codebook flexibility. In standard VQ only 16 codebook vectors can be used, compared to $2^8 = 256$ codebook vectors in GVQ. Despite there being more codebook vectors available in GVQ, the information required to define the compressed image using GVQ is *less* than that in VQ, as we show in the following section (section (6.2.1)).

The features, which were learned to construct codebook vectors representing the segments of the Vermeer image, are shown in fig(11). Interestingly, the final features can be seen to be slightly biased to modeling variation around the vertical direction, which is plausible given the large number of almost vertical shadows in the original image.

[6] The Girl with a Pearl Earring (1665) by Johannes Vermeer, Mauritshuis, The Hague (The Netherlands)

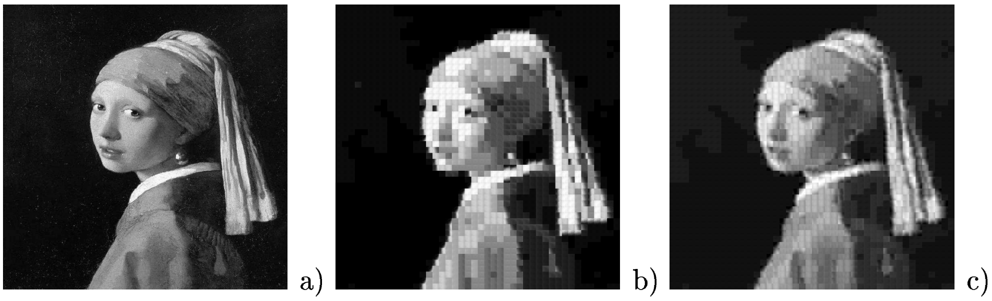

Figure 10. a) The Vermeer image prior to compression consists of 865 kbits. After compression: b) With standard vector quantisation $I_v = 74$ kbits, c) with GVQ using 1 set of 8 features (see fig(11)) $I_{GVQ} = 56$ kbits.

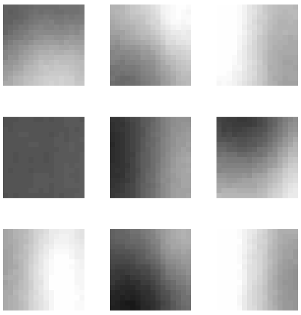

Figure 11. Features which were learned for representing the 2112-16 × 16 bit segments constituting the Vermeer image. The lower-right feature is the origin feature, which is always on.

6.2.1. *Information Requirements*

Since a codebook vector in GVQ is constructed out of a set of n features, we need at most n bits to specify a feature combination vector. If the data have a clustered structure, the number of bits needed will be smaller than n since some combinations will never be used. Therefore, if the number of used feature combinations is N_{gvq} we also need $\log N_{gvq} \leq n$ bits to specify a codebook vector in GVQ. Similarly, in standard vector quantisation we need $\log N_{vc}$ bits to specify a codebook vector if N_{vc} is the number of learned codebook vectors.

Consider the case that the image to be compressed is unique, in the sense that we can not use features (or codebook vectors) which were used to encode previously encountered images. In order to compare the compression efficiencies, we need to take into account the information to describe the codebook vectors in VQ and the information in the features in GVQ. This information is proportional to the number of pixels n_p used in an image segment, and the information required to determine the gray value of a pixel I_g. Hence, if the original image is split into P segments, each made up of n_p pixels, the information I_{VQ} in the compressed image using VQ is

$$I_{VQ} = I_g n_p N_{vq} + P \log N_{vq}$$

and the information I_{GVQ} in the compressed image using GVQ is

$$I_{GVQ} = I_g n_p n + P \log N_{gvq}.$$

If there is a moderately number of segments P, which is the case if we compress a single specific image, then for a given compression quality[7] the difference between I_{VQ} and I_{GVQ} is determined mainly by the difference between N_{vq} and n. Since we can construct a large number of codebook vectors with a small number of features it is expected that $n \ll N_{vq}$, especially if the distribution of the image segments has a structured multi-modal form.

The image in fig(10)(b), obtained after compression with standard VQ, consists of $I_{VQ} = 74$ kbits. In contrast, if we apply our GVQ algorithm using $n = 8$ features the compressed image, fig(10)(c), consists of $I_{GVQ} = 56$ kbits. While containing 18 kbits less of information, the GVQ compressed image gives without doubt a superior representation of the original image.

7. Relation of GVQ to other models

A large number of data modeling techniques can be seen as special cases of using Gaussian mixture models. Although not necessary for the motivation for GVQ, in this section we describe how GVQ can be seen as part of an ongoing tradition by relating it to the framework of Gaussian mixture models. This will enable us to clarify the relation of our model to other recently proposed techniques and approximations to them.

In the present context, a Gaussian mixture model can be conveniently considered as a layer of hidden or latent variables $\mathbf{s} = (s_1 \dots s_n)$ connected to a layer of visible variables $\mathbf{x} = (x_1 \dots x_N)$. Each data point then corresponds to an instantiation of the visible units. The distribution on the visible units is obtained from the marginal of the joint distribution over the hidden and visible units,

$$p(\mathbf{x}) = \sum_{\mathbf{s}} p(\mathbf{x}|\mathbf{s})p(\mathbf{s}), \tag{11}$$

where the likelihood term is given by

$$p(\mathbf{x}|\mathbf{s}) = \left(2\pi\sigma^2\right)^{-d/2} e^{-\frac{1}{2\sigma^2}||\mathbf{x}-g(\mathbf{s})||^2}. \tag{12}$$

For convenience, we write the prior distribution of the hidden states $p(\mathbf{s})$ in terms of an energy function $\phi(\mathbf{s})$,

$$p(\mathbf{s}) = \frac{1}{Z} e^{-\frac{\lambda}{\sigma^2}\phi(\mathbf{s})}, \tag{13}$$

where Z is the normalizing constant for the prior. Within this model, each hidden state $\mathbf{s}$ corresponds to a uniquely located Gaussian distribution in the visible layer with mean $g(\mathbf{s})$ with a prior belief that each Gaussian is responsible for the visible variable given by the prior $p(\mathbf{s})$.

In this section we discuss how various aspects of the GVQ model can be related to different choices of the mean function $g(\mathbf{s})$ and prior $p(\mathbf{s})$ in the limit that σ goes to zero.

[7] By compression quality we mean the similarity between the compressed image and the original image.

7.1. The deterministic vs the noisy approach

Taking the zero σ limit in the Gaussian mixture model provides a hard assignment of data points $\mathbf{x}$ to closest Gaussian centers. The resulting $\sigma \to 0$ model corresponds to standard vector quantisation in which the codebook vectors are the cluster centers $g(\mathbf{s})$. In GVQ the additional constraint is that the codebook vectors are generated by binary combinations of vectors. This corresponds to the choice $g(\mathbf{s}) = \sum_i \mathbf{f}_i s_i$ for the mean function, where the hidden states are now coded as binary vectors $\mathbf{s} \in \{0,1\}^n$. As discussed in section (4.2) the finite noise model corresponding to GVQ is given by

$$p(\mathbf{s},\mathbf{x}) = p(\mathbf{s})\left(2\pi\sigma^2\right)^{-\frac{d}{2}} \exp\left\{-\frac{1}{2\sigma^2}\|\mathbf{x} - \sum_i \mathbf{f}_i s_i\|^2\right\}, \tag{14}$$

where, in the basic approach of section (3.1), the prior probability of a feature combination is the same for each combination *i.e.* $p(\mathbf{s})$ in (14) is constant. This finite noise model is closely related to the Cooperative Vector Quantizer proposed in (Zemel, 1994) which was further investigated in the context of mean-field learning in (Ghahramani, 1995). GVQ corresponds to (14) with $\sigma \to 0$. This implies that the posterior probabilities $p(\mathbf{s}|\mathbf{x})$ reduce to $p(\mathbf{s}|\mathbf{x}) = \delta(\mathbf{s}^* - \mathbf{s})$ in which $\mathbf{s}^*$ is the unique binary state that is associated with $\mathbf{x}$. Since $\delta(\mathbf{s}^* - \mathbf{s}) = \prod_i \delta(s_i^* - s_i)$, in the zero noise limit the multi-dimensional distribution of the hidden binary states $\mathbf{s}$ is given by the product of its marginals. For this reason the message passing algorithms, described in section (4), which infer marginal probabilities $p(s_i)$ become directly relevant in the zero noise limit. As we saw in section (4.3) there are indeed situations where these message passing algorithms outperform the variational algorithms for finite noise models.

In summary, GVQ forces a solution in which each data point is explained by a single process - that is, there is a unique explanation for each data point, found by a competitive process subject to this requirement. On the other hand, in a finite σ model, each data point is associated with each Gaussian center with a certain probability. If one attempts to interpret each data point in terms of features one would then need to evaluate the contribution of each Gaussian to the explanation for the data point in some manner. In applications where clusters have strong overlap, these contributions are important. Otherwise, a finite σ model would make interpretation unnecessary complicated. Another motivation for considering the zero noise limit is that taking the most probable explanation (nearest Gaussian center) in the finite σ models may not give rise to a satisfactory interpretation since the competition between Gaussian centers for the single best explanation for each data point has not been optimized during learning.

GVQ is most appropriate in cases in which one believes that any data point is well explained by a single process (codebook vector).

7.2. Alternative choices for the prior distribution

Within the probabilistic framework, some interesting connections can be made to the sparse coding work of (Olshausen, 1996) and the independent factor analysis (IFA) work of (Attias, 1999). The main difference between these models and our model is in the assumption for the hidden state distribution $p(\mathbf{s})$. Both authors consider hidden state variables with *continuous* values, that is $\mathbf{s} \in \Re^n$. In IFA Attias (Attias, 1999) considers a product of Gaussian mixture distributions (independent factors) for the hidden states $\mathbf{s}$. If, within IFA, the number of mixtures for each hidden state variable s_i is set to 2 (bi-modal distribution), the distribution of the visible patterns can be regarded as a noisy version of the binary feature combinations in GVQ.

In contrast to the multi-modal assumption of IFA and the binary assumption of GVQ, Olshausen and Field (Olshausen, 1996) consider a sharply peaked *unimodal* distribution for continuous hidden variables $\mathbf{s}$. This choice encourages a sparse representation of the data patterns since the hidden variables s_i will be in the 'off' state most of the time. In this case an individual pattern will be constructed as a combination of only a small number of features out of a large, typically over-complete, set of features. As discussed in section (5.1), a similar property can be incorporated in our method by replacing the basic constant GVQ prior in (13) with a soft prior $\phi(\mathbf{s}) = \mathbf{s}^T\mathbf{s}$. Note, however, that when this penalty term becomes too large, only a single feature will be used to represent a pattern. In other words GVQ will tend to standard vector quantisation as the solution is strongly encouraged to be sparse.

7.3. Binary Lattice Vector Quantisation

In another context, research has been done on Binary Lattice Vector Quantizers or Direct Sum Quantizers. In fact, the binary codebook representation of GVQ model is formally equivalent to the representation of a Binary Lattice Vector Quantizer. These representations have been studied for the purpose of data transmission across noisy channels. The main objective there is to transmit coded data such that the reconstruction error is minimal. An important sub-problem within that objective is the 'Index Assignment Problem' which is to find an optimal binary index assignment to codebook vectors as to minimize the mean-squared error caused by channel errors, see (McLaughlin, 1995) and (Knagenhjelm, 1996).

For 'direct sum quantizers', an alternative method for the association step is studied in (Barnes, 1993). This method is a heuristic compromise between component wise optimization and exhaustive search where the association is done in multiple stages bringing the codebook closer to the data-point at each stage. Whether this method is more accurate and efficient than the methods studied in section (4) remains to be investigated.

To our knowledge binary lattice vector quantizers have, however, not been studied for the purpose of feature extraction and clustering of non-homogeneous data, which has been the purpose of the present paper.

8. Conclusion

Generative Vector Quantisation is a method, which performs salient feature extraction at modest computational expense. The simplicity of GVQ, which searches for descriptions in terms of binary feature combinations, may lead to a lucid data representation, which is important in many data exploration tasks. A central thesis of the GVQ model is that data points are explained by a single generating process. Unlike a probabilistic model, GVQ constructs a competition between alternative explanations for a data point, in which there can be only one winning explanation. This winner-take-all process provides the basis for a clear feature representation. The deterministic nature of GVQ allows the use of a larger class of (approximate) association methods, such as Belief Revision, within the learning scheme. However, in the case that the data cannot be expected to be explained by a winner-take-all process, a probabilistic approach may be a more appropriate.

GVQ is potentially a powerful tool for exploring and representing data in a deterministic manner. Ultimately, the strength of GVQ lies in it's transparent simplicity, being based on the intuitive notion that, although data may appear complex, it's construction may be well understood in terms of a small number of elementary building blocks.

References

Attias, H. Independent factor analysis. *Neural Computation*, 11(4):803–851, 1999.

Barlow, H. B. Unsupervised learning. *Neural Computation*, 1(3):295–311, 1989.

Barnes, C.F. and Frost, R.L. Vector Quantizers with Direct Sum Codebooks. *IEEE Transactions on Information Theory*, 36(2):565–580, 1993.

Bell, A. J. and Sejnowski, T.J. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995.

Dempster, A.P. , Laird, N.M. and Rubin, D.B. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society Series B*, 39:1–38, 1977.

Ghahramani, Z. Factorial Learning and the EM algorithm In *Advances in Neural Information Processing Systems*, volume 7. The MIT Press, 1995.

Ghahramani, Z. and Hinton, G.E. Hierarchical non-linear factor analysis and topographic maps. In *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

Gray, R.M. Vector quantisation. *IEEE ASSP Magazine*, pages 4–29, 1984.

Hull, J.J. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1997.

Knagenhjelm, P., and Agrell., E. The Hadamard transform–a tool for index assignment. *IEEE Transaction on Information Theory*, 42(4):1139–1151, 1996.

McEliece, R., Rodemich, E., and Cheng, J. The Turbo Decision Algorithm. *Proc. 33rd Allerton Conference on Communications, Control and Computing*, 366–379, 1995.

McLaughlin, S.W., Neuhoff, D.L., and Ashley, J.J. Optimal binary index assignments for a class of equiprobable scalar and vector quantizers. *IEEE Transactions on Information Theory*, 41(6):2031–2037, 1995.

Mehes, A. and, Zeger, K. Binary lattice vector quantisation with linear block codes and affine index assignments. *IEEE Transactions on Information Theory*, 44(1):79–94, 1998.

Neal, R. M. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113, 1991.

Neal, R.M. and Hinton, G. E. *Learning in Graphical Models*, chapter A view of the EM algorithm that justifies incremental, sparse, and other variants, pages 355–368. Kluwer Academic Publishers, 1998.

Olshausen, B. A. and Field, D. J. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

Pearl, J. *Probabilistic Reasoning in Intelligent systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc., 1988.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B.P. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, 2 edition, 1992.

Sallans, B., Hinton, G.E., and Ghahramani,Z. A hierarchical community of experts. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, NATO ASI Series F, pages 269–284. Springer-Verlag, 1998.

Saul, L.K., Jaakkola, T., and, Jordan, M.I. Mean Field Theory for Sigmoid Belief Networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

Saund, E. A multiple cause mixture model for unsupervised learning. *Neural Computation*, 7(1):51–71, 1995.

Weiss, Y. Belief Propagation and Revision in Networks with Loops. Technical report, MIT-AILab, AIM-1616,1997.

Zemel, R.S. A minimum description length framework for unsupervised learning. Technical report, University of Toronto, CRG-TR-93-2, 1994.

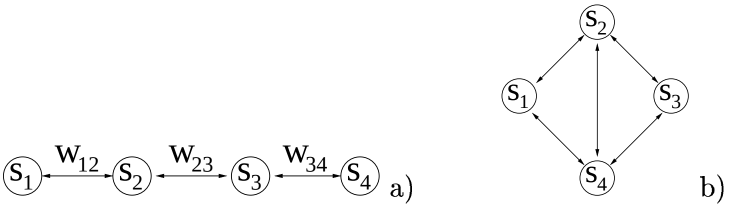

Figure 12. Pairwise graphical model with (a) a chain structure (b) loops

Appendix

A. Belief Propagation

An efficient technique to find an approximate solution to the marginal $p_{\mathbf{x}}(s_i)$ of (8) is to decompose the global summation operation into distributed local operations, reducing the exponential summation to quadratic time.

For expositional clarity, consider an energy function $E_{\mathbf{x}}(\mathbf{s})$, with weights w_{ij} defined to give a chain structure as shown in figure 12(a) (technically, the weight matrix has zeros everywhere except for entries along the first diagonals adjacent to the main diagonal). For the chain structure in fig(12)(b) the marginal probability that unit 1 is in state s_1 is

$$p(s_1) \propto \sum_{s_2,s_3,s_4} e^{(h_1 s_1 + s_1 w_{12} s_2 + h_2 s_2 + s_2 w_{23} s_3 + h_3 s_3 + s_3 w_{34} s_4 + h_4 s_4)} \tag{15}$$

which can be decomposed into local operations as follows

$$p(s_1) \propto e^{\nu[h_1 s_1]} \sum_{s_2} \left(e^{\nu[h_2 s_2 + s_2 w_{12} s_1]} \sum_{s_3} \left(e^{\nu[h_3 s_3 + s_2 w_{23} s_3]} \left(\sum_{s_4} e^{\nu[h_4 s_4 + s_3 w_{34} s_4]} \right) \right) \right) \tag{16}$$

where $\nu = -\frac{1}{\sigma^2}$ and . Distributing the marginalization in this manner results in a summation over a number of states that scales only linearly with the network size instead of over an exponentially scaling number of states in (15). To write this in a more general form we define the message that node S_j sends to S_i as

$$\lambda_{ij}(s_i) = \alpha \sum_{s_j} e^{\nu[h_j s_j + s_i w_{ij} s_j]} \left(\prod_{k \in \mathcal{C}_j \setminus i} \lambda_{jk}(s_j) \right), \tag{17}$$

where $\mathcal{C}_j$ is the set of all nodes connected to node j. Combining the incoming messages λ_{ij} into node S_i gives the marginal probability distribution of that node

$$p(s_i) \propto e^{\nu h_i s_i} \prod_j \lambda_{ij}(s_i). \tag{18}$$

The recurrent marginalization procedure defined by (17) and (18) will give an exact solution for all connection weights w_{ij} that define singly connected graphs *i.e.* graphs without loops.

For graphs with loops, for example fig(12)(b), which corresponds to the energy function

$$\begin{aligned} E_{\mathbf{x}}(\mathbf{s}) = & h_1s_1 + s_1w_{12}s_2 + h_2s_2 + s_2w_{23}s_3 + \\ & + h_3s_3 + s_3w_{34}s_4 + h_4s_4 + s_1w_{14}s_4 + s_2w_{24}s_4 \end{aligned} \tag{19}$$

the method is still applicable although no longer guaranteed to find the optimal solution. Nevertheless, there is experimental evidence (McEliece, 1995; Weiss, 1997) that for large classes of graphs with loops the belief propagation algorithm gives good solutions.

In our implementation the messages are initialized as $\lambda_{ij}(s_i) = \alpha \sum_{s_j} e^{\nu[h_j s_j + s_i w_{ij} s_j]}$ with $\nu = -1/(2(0.1)^2) = -50$. After initialisation an iteration in the procedure is as follows:

- A random ordering of the nodes is chosen, which are then sequentially visited in that order.
- For each node, all messages coming into the node are updated according to the rule (17) and the state of node i is updated to $s_i^* = \underset{s_i}{\operatorname{argmax}}\Big(e^{\nu h_i s_i} \prod \lambda_{ij}(s_i)\Big)$.
- The variance σ^2 is halved in $\nu = 1/(2\sigma^2)$.

This iterative process is repeated until $\sigma = 10^{-4}$. We then choose that state $\mathbf{s}$ which in the iterations had the lowest energy $E_{\mathbf{x}}(\mathbf{s})$.

B. Belief revision

The inference problem that we need to solve is to find a single hypothesis or explanation $\mathbf{s}$ for each observed state $\mathbf{x}$.

The minimisation problem for the chain in fig(12)(a) is

$$E^* = \min_{s_1,s_2,s_3,s_4} \left(h_1s_1 + s_1w_{12}s_2 + h_2s_2 + s_2w_{23}s_3 + h_3s_3 + s_3w_{34}s_4 + h_4s_4\right) \tag{20}$$

which can in analogy with (16) be decomposed into local operations as follows

$$E^* = \min_{s_1}\left(h_1s_1 + \min_{s_2}\left(s_1w_{12}s_2 + h_2s_2 + \min_{s_3}\left(s_2w_{23}s_3 + h_3s_3 + \min_{s_4}\left(s_3w_{34}s_4 + h_4s_4\right)\right)\right)\right). \tag{21}$$

Note that (21) has the same de-componential structure as (16) except that the summation operator is changed into a minimisation operator and the messages are combined as a summation instead of as a product.

In a more general form, we define the consider the *message* $\lambda_{ij}(s_i)$ that node S_j sends to S_i as

$$\lambda_{ij}(s_i) \equiv \min_{s_j}\left(s_iw_{ij}s_j + h_js_j + \sum_{k\in\mathcal{C}_j\setminus i} \lambda_{jk}(s_j)\right), \tag{22}$$

where $\mathcal{C}_j$ is the set of all nodes connected to node j. With this definition we see from (21) that the minimisation problem for the network in fig(12) can be rewritten in the following recurrent form

$$E^* = \min_{s_1} \left(\sum_{j \in \mathcal{C}_j \backslash 1} \lambda_{1j}(s_1) + h_1 s_1 \right). \tag{23}$$

The recurrent minimisation procedure defined by (22) and (23) will give an exact solution for all connection weights w_{ij} that define singly connected graphs *i.e.* graphs without loops. Nevertheless, there is experimental evidence (McEliece, 1995; Weiss, 1997) that for large classes of graphs with loops the belief propagation algorithm gives good solutions.

In our *implementation*, the messages are initialized as $\lambda_{ij}(s_i) = \min_{s_j} \left(s_i w_{ij} s_j + h_i s_i\right)$. After initialisation an iteration in the procedure is as follows:

- A random ordering of the nodes is chosen, which are then sequentially visited in that order.

- For each node, all messages coming into the node are updated according to the rule (22) and the state of node i is updated to $s_i^* = \underset{s_i}{\mathrm{argmin}} \Big\{ h_i s_i + \sum_{j \in \mathcal{C}_i \backslash i} \lambda_{ij}(s_i) \Big\}$.

This iterative process is repeated until the messages converge. If they do not converge, the iterations are stopped after a predefined maximum number of iterations. We then choose that state $\mathbf{s}$ which in the iterations had the lowest energy.

The computational complexity of this algorithm is quadratic in the number of nodes since, for each of the n nodes, there are n messages, in the fully connected case.

C. Mean Field

The basic idea of variational algorithms (of which the mean-field method is a special case) is to replace the intractable objective function with a tractable approximation to it, so that the optimization of the approximate objective function can be carried out efficiently.

To explain the mean-field approximation for the association step in GVQ learning we first formulate the model as a probability distribution with finite noise σ.

$$p_x(\mathbf{s}) \propto \exp\left\{ -\frac{1}{2\sigma^2} E_x(\mathbf{s}) \right\}. \tag{24}$$

Finding the most probable state $\mathbf{s}$ of $p_x(\mathbf{s})$ is equivalent to minimising $E_x(\mathbf{s})$, and in the limit $\sigma \to 0$, the distribution $p_x(\mathbf{s})$ becomes deterministic. That is, the *mean* state $\mathbf{s}$ is equal to the most probable state. We can therefore use an algorithm that attempts to approximate the mean of $p_s(\mathbf{s})$ for finite σ and, in the limit that $\sigma \to 0$, this will become an approximation for the most probable state.

One way to find an approximation to the mean of the variables of an intractable distribution is to use a simpler, tractable approximating distribution. Specifically, in the variational method the objective is to find an approximating $Q_x(\mathbf{s})$ distribution to the state distribution $P_x(\mathbf{s})$ with which the associations can be tractably computed. The optimal approximating $Q_x(\mathbf{s})$ is found by minimising the Kullback-Leibler divergence between the two distributions

$$KL = \sum_{\mathbf{s}} Q_x(\mathbf{s}) \log \frac{Q_x(\mathbf{s})}{P_x(\mathbf{s})} \geq 0 \tag{25}$$

with respect to the parameters of $Q_x(\mathbf{s})$. Note that the Kullback-Leibler divergence is a positive measure of the difference between two distributions.

In its most basic form, the variational approximating distribution $Q_x(\mathbf{s})$ is factorial. This is known as the mean-field assumption for $Q_x(\mathbf{s})$

$$Q(\mathbf{s}|\mathbf{x}) = \prod_i q_i(s_i) = \prod_i \mu_i^{s_i}(1-\mu_i)^{1-s_i}, \tag{26}$$

where $\mu_i \in [0,1]$ are called the mean-field parameters. Substitution of $Q_x(\mathbf{s})$ into KL gives (up to a constant):

$$KL' = -\sum_i \mu_i \log \mu_i + (1-\mu_i)\log(1-\mu_i) - \frac{1}{2\sigma^2}\left(\sum_l \sum_{j>l} \mu_l \mu_j w_{lj} + \sum_i \mu_i h_i\right). \tag{27}$$

To find a solution for the μ_i, we set the derivatives w.r.t. to mean-field parameters μ_i equal to zero, which leads to the following mean-field fixed point equations,

$$\mu_i = \mathrm{sig}\left(\frac{1}{2\sigma^2}\left\{\sum_{l \neq i} \mu_l w_{li} + h_i\right\}\right), \tag{28}$$

where $\mathrm{sig}(x) = (1-\exp(-x))^{-1}$. In the limit $\sigma \to 0$ these equations become

$$\mu_i = \Theta\left(\sum_{l \neq i} \mu_l w_{li} + h_i\right), \tag{29}$$

where $\Theta(x) = 0$ for $x \leq 0$ and $\Theta(x) = 1$ for $x > 0$. Hence, the solutions for μ_i become binary and there no longer exists a distinction between state values s_i and state probabilities μ_i. For a given input $\mathbf{x}$ (29) defines an iterative procedure to find an associated state $\mathbf{s}$.

In our implementation, the mean-field parameters are initialized as $\mu_i = \frac{1}{2} + \epsilon$, where ϵ is small random noise and σ is initialized as $\sigma = 100$. After initialisation an iteration in the procedure is as follows:

- A random ordering of the nodes is chosen, which are then sequentially visited in that order.
- Each node, is then updated according to equation (28).
- The noise σ is then reduced according to $\sigma \leftarrow \sigma/\alpha$.

The noise reduction parameter α is chosen such that $\sigma = 10^{-4}$ at the final iteration.

C.0.1. *Gibbs Sampling*

Another well known optimization technique for stochastic models is Gibbs sampling. In Gibbs sampling the state of a unit is updated according to the probability

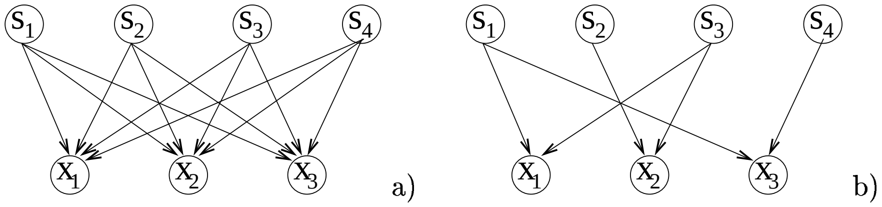

Figure 13. Graphical structure of GVQ

$p(s_i|\mathbf{s}_{-i})$, where $\mathbf{s}_{-i}$ contains the values of all units besides unit i. This conditional probability for the model (28) is

$$p(s_i|s_{-i}) = \mathrm{sig}\left(\frac{1}{2\sigma^2}\left\{\sum_{l\neq i} s_l w_{li} + \theta_i\right\}2\left(s_i - \frac{1}{2}\right)\right). \tag{30}$$

In the GVQ limit $\sigma \to 0$ this equation defines an iterative scheme which is the same as that for the mean-field method. Hence, in the limit $\sigma \to 0$ the mean-field method and the Gibbs sampling method are equivalent.

D. Belief Propagation in the Directed Graph

Our goal is to compute marginal probabilities $p(s_i|\mathbf{x})$ in a directed graphical model with a structure as shown in fig(13)a). For graphs with tree like structures one can, analogous to the undirected case, decompose the summation into local operations. For a complete treatment of how this is done we refer to (Pearl, 1988), here we simply state the results. For a directed network there are two types of messages, namely ρ-messages that are send in the direction of the arrows from parent nodes (binary units S_j) to child nodes (visible units X_i) and λ-messages that are sent in the opposite direction. The following recursive procedure is guaranteed to give the exact solution for directed graphs without cycles such as shown in fig(13)b):

- The message that visible unit X_i sends to hidden unit S_j is given by

$$\lambda_{X_i S_j}(s_j) = \sum_{\mathbf{s}'\in\{\mathrm{Pa}(X_i)\backslash S_j\}} p(x_i|s_j, \mathbf{s}') \prod_{S_k\in\mathrm{Pa}(X_i)\backslash S_j} \rho_{S_k X_i}(s'_k), \tag{31}$$

  where $\mathrm{Pa}(X_i)\backslash S_j$ is the set of parent units of unit X_i excluding unit S_j. The set of states $\mathbf{s}'$ of this set is notated as $\{\mathrm{Pa}(X_i)\backslash S_j\}$.

- Message from hidden unit S_k to visible unit X_i

$$\rho_{S_k X_i}(s_k) \propto \pi(s_k) \prod_{X_j\in\mathrm{Ch}(S_k)\backslash X_i} \lambda_{X_j S_k}(s_k), \tag{32}$$

  where $\mathrm{Ch}(S_k)$ is the set of child units of binary node S_k.

The marginal probabilities $p(s_i)$ are given by

$$p(s_i) \propto \pi(s_i) \prod_{X_k\in\mathrm{Ch}(S_i)} \lambda_{X_k S_i}(s_i) \tag{33}$$

As for the undirected methods it is not guaranteed that this method gives the exact result for $p(s_i|\mathbf{x})$ if the network contains cycles. There is evidence (Weiss, 1997), however, that for certain structures, even with loops, the message passing scheme presented above may give good results. In contrast to the undirected algorithm, the computation of λ messages, (31), involves a summation over an exponentially large set of states. Hence, straightforward application of the algorithm results in a method which scales exponentially with the with the number of parents of single visible nodes X_i. For the case of the GVQ model, the special form of the conditional probabilities, $p(x_i|\mathbf{s}) = e^{-\frac{1}{2\sigma^2}\left[x_i - \sum_k f_{ik}s_k\right]^2}$, allows us to use a 'trick' with which the summation Eq.(31) can be computed tractably. The trick is to remove the quadratic interactions in the exponent using the identity

$$\sqrt{\frac{\pi}{a}} e^{b^2/(4a)} = \int_{-\infty}^{\infty} dy e^{-ay^2+by} \tag{34}$$

where $i = \sqrt{-1}$ is the unit imaginary number. Application of (34) in (31) results in the following expression for the λ messages

$$\lambda_{X_iS_j}(s_j) \propto \int_{-\infty}^{\infty} dy \prod_k \left\{\rho_{S_kX_i}(s_k = 0) + \rho_{S_kX_i}(s_k = 1)e^{iyf_{ki}/\sigma}\right\} e^{-y^2+iy(x_i - f_j s_j)/\sigma}. \tag{35}$$

The integration can be done efficiently with Gaussian quadratures and its complexity scales (only) linearly with the number of parents connected to node X_i.

Our implementation starts with the initialisation of λ and ρ messages. Then a single iteration of the algorithm consists of the following steps:

1. For each parent node S_j compute the incoming $\lambda_{X_iS_j}$ messages from all visible nodes X_i with (35);

2. For each child node X_i, compute the incoming $\rho_{X_jS_k}$ messages from all the connected parent nodes S_k with (32);

3. Reduce the noise according to $\sigma \leftarrow \sigma/\alpha$.

In our experiments we use $\alpha = 2$.

E. Belief Revision in the Directed Graph

Instead of slowly reducing the noise σ while running the belief propagation algorithm of appendix (D) we now formulate the algorithm directly for infinitesimal σ. In the limit $\sigma \to 0$ a single state will start to dominate the summation (31). Hence, in the limit $\sigma \to 0$ the expression for the λ messages are

$$\lambda_{X_iS_j}(s_j) \propto \max_{\mathbf{s}' \in \{\mathrm{Pa}(X_i)\backslash S_j\}} p(x_i|s_j, \mathbf{s}') \prod_{S_k \in \mathrm{Pa}(X_i)\backslash S_j} \rho_{S_kX_i}(s'_k). \tag{36}$$

In this case we can no longer apply the integral 'trick' as in (35) since maximization can not be interchanged with integration. Hence, the computational complexity of (36) scales exponentially with number of parents in $\mathrm{Pa}(X_i)$.

The update equations (32) for the ρ-messages do not change in the limit $\sigma \to 0$. Finally, the belief revision solution for the minimising state $\mathbf{s}^*$ of (5) is

$$s_i^* = \operatorname*{argmax}_{s_i} \prod_{X_k \in \mathrm{Ch}(S_i)} \lambda_{X_k S_i}(s_i) \tag{37}$$

Note that this solution does not depend on the parameter σ in $p(x_i|s_j, \mathbf{s}')$. In the implementation we use $\sigma = 1$.