Generative Vector Quantisation

Machiel Westerdijk*, David Barber and Wim Wiegerinck*

Department of Medical Physics and Biophysics, University of Nijmegen†

The Netherlands

Abstract **– Based on the assumption that a pattern is constructed out of features which are either fully present or absent, we propose a vector quantisation method which constructs patterns using binary combinations of features. For this model there exists an efficient EM-like learning algorithm which learns a set of representative codebook vectors. In terms of a generative model, the collection of allowed binary states 'generates' the set of codebook vectors. The method, therefore, provides not only a compact description of the data in terms of clusters, but also an explanation of the individual clusters in terms of common elementary features. Preliminary results on image compression and handwritten digit analysis indicate that our approach is an interesting and computationally inexpensive alternative to more complex probabilistic generative graphical models.**

1 Introduction

In recent years, a number of methods have been proposed which seek for a description of data in terms of elementary features. It is expected that these methods give rise to more natural and (therefore) more compact models which may lead to better generalisation. Furthermore, because the description is in terms of features these models are easy to interpret, which may be helpful in understanding the hidden data generating process.

Advanced non-linear probability models have recently been promoted by several authors in this context [1, 2, 7]. However, to our knowledge all these models scale badly on larger real world problems, and most applications have been rather limited. Non-probabilistic methods such as vector quantisation (VQ) [5] are able to capture, in principle, any structure, linear or non-linear, using considerably less computational resources. The drawback of VQ however is that it does not have a feature representation. We propose a method, Generative Vector Quantisation (GVQ), which combines the advantages of a description in terms of elementary features with the ability to learn complicated non-linear structures at modest computational expense.

We present an efficient EM-like learning algorithm in section 2. In section 3 we show that Generative Vector Quantisation gives a good performance in large scale real world tasks like image compression and handwritten digit analysis, including problems with up to 400 data dimensions.

2 Generative Vector Quantisation (GVQ)

The aim of vector quantisation is to represent a dataset by a smaller set of codebook vectors. In GVQ, we consider a set of basic feature vectors $\{\mathbf{f}^1, \ldots, \mathbf{f}^{n_f}\}$ which exist in the same N-dimensional space as the data. Codebook vectors are then formed by binary combinations of these feature vectors,

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

*Supported by the Technology Foundation STW, applied science division of NWO

†http://www.mbfys.kun.nl/~{machiel, davidb, wimw}

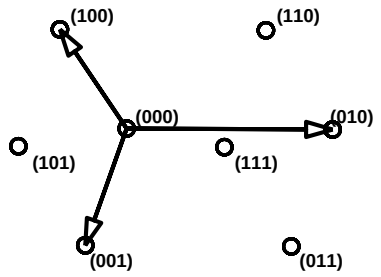

Figure 1: Codebook vectors (circles) are generated by a small set of basic features $\mathbf{f}^1, \mathbf{f}^2$ and $\mathbf{f}^3$, which correspond to the states $(100)^T$, $(010)^T$ and $(001)^T$, respectively.

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

An example of a set of codebook vectors generated by 3 features in a 2-dimensional space is shown in figure 1. The circles represent the generated codebook vectors which correspond to the 8 states $(000)^T$, $(100)^T$, $\cdots$, $(111)^T$.

Note that the number of features n_f is not related to the dimensionality of the data space. Hence, considered as a basis, the feature set may be under or over complete.

2.1 GVQ Learning

In GVQ, as in standard VQ, each data point $\mathbf{x}^\mu$ is associated with a particular codebook vector indexed by c_μ. The squared Euclidian distance between the 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

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

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 describe 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 closest codebook vector and a minimisation step (2) which finds the optimal feature configuration for the given association:

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

$$c_\mu \leftarrow \underset{j}{\text{argmin}} \|\mathbf{x}^\mu - F\mathbf{s}^j\|^2, \qquad (2)$$

2.

$$F \leftarrow \underset{\tilde{F}}{\text{argmin}} \sum_{\mu} \|\mathbf{x}^\mu - \tilde{F}\mathbf{s}^{c_\mu}\|^2 \qquad (3)$$

The quadratic form in (3) can be minimised efficiently by use of Singular Value Decomposition.

Although the GVQ algorithm is convergent, care needs to be taken to avoid local minima. As in standard VQ, the quality of the solution is highly dependent on the choice of the initial codebook vectors, or in this case an initial choice for the features F. Random initialisation of all the features typically results in acceptable but sub-optimal solutions. Superior results on the problems tested so far were obtained by sequentially increasing the number of features. At the n^{th} stage, the procedure starts with the solution obtained for $n-1$ features and includes an extra (small randomly oriented) n^{th} feature. This gradual increase of complexity leads to a more uniform distribution of codebook vectors over the data, resulting in smaller errors.

2.2 Illustration on a 2-D problem

In figure 2 we apply the GVQ learning algorithm to 200 random uniformly generated data points (small circles) on the unit square. The bold arrows are the found features whose binary combinations generate the eight codebook vectors, represented by large circles. Which codebook vector each data point is associated with is indicated by a thin line. We remind the reader that the GVQ learning algorithm begins with one feature and iterates until convergence. Additional features are included one at a time and at each stage the optimal solution is found in less than 10 iterations.

In the example all of the possible feature combinations have data points associated to them. In general, however, it is possible that

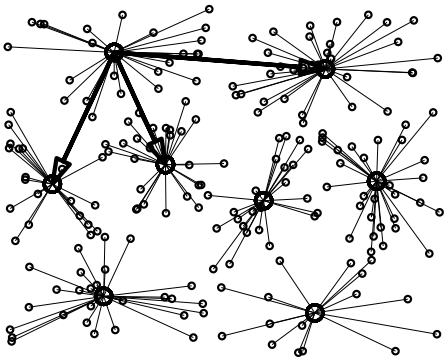

Figure 2: GVQ solution with 3 features for data distributed at random uniformally on the unit square. The bold arrows are the features and the large circles are end points of the codebook vectors. The small dots represent the data and the lines between the data points and codebook vectors represent the associations.

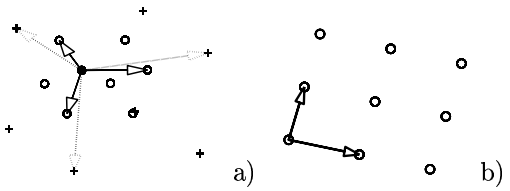

Figure 3: a) Two sets of 3 features and b) Codebook vectors generated by two 3-valued features.

a number of states will move too far out of the data cloud to be associated with any data point. Hence the final codebook may contain far fewer than 2^{n_f} vectors. In the practical problems that we considered, the fraction ϵ of discarded codebook vectors typically remains small if the number of states is smaller than the number of patterns, i.e. $2^{n_f} < P$. Furthermore, if this condition applies then ϵ scales roughly linear with 2^{n_f}. Hence, the frame which is spanned by the binary feature combinations is in general flexible enough to stay close to the data.

2.3 Multiple feature sets and multi-valued features

In many real world problems we can expect that there are different groups of patterns which each are built up of a different unrelated set of features. It is easy to adjust the GVQ learning algorithm for learning multiple feature sets by simply constraining the set of allowable states. An example of this situation is given in figure 3-a. This extra freedom makes it possible to work with a much larger number of features without suffering from high computational costs. In the extreme case of using one feature per set, GVQ is equivalent to standard vector quantisation.

Another extension is to use multi-valued, rather than just binary, feature combinations. This extension can also be incorporated directly into the GVQ algorithm by using $s_i \in \{0, 1, \ldots, K\}$. An example using two three valued features is shown in figure 3-b.

3 Results

3.1 Handwritten digits

We randomly selected 400 training images of handwritten threes and fives from the CEDAR CDROM 1 database [6]. Since the original images contain different numbers of pixels, we rescaled all images to 20×20 pixels. In contrast to [1] and [7] there was no need to reduce the image size further since the algorithm convergences quickly scaling only linearly with the number of input dimensions. We applied our GVQ algorithm to this training set using a single set of $n_f = 8$ features. At each stage of feature inclusion the algorithm reached a local minimum typically after only $n_i = 20$ iterations. On a Pentium 166 MHz computer the CPU-time of our algorithm implemented in Matlab code is $5 \cdot 10^{-6} \times P \times N \times 2^{n_f} \times n_i$ seconds. Learning a codebook with $n_f = 8$ features on a training set with $P = 400$ patterns, each having $N = 400$ dimensions, as in the example, takes 1.1 hours.

A typical sample of training set images together with their closest codebook vectors obtained after training is shown in figure 4. The reproduced data are seen to be faithful representations of the original data. The features which were obtained are shown in figure 5-b1 and -b2. The origin of the feature basis (an extra feature which is learned but which is always switched on) is shown in figure 5-a. The image in the left of figure 5-c is a combination of features 4, 6 and 8

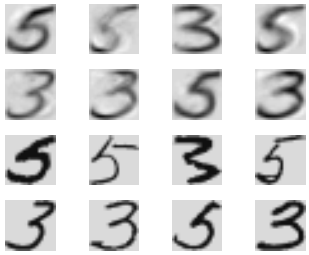

Figure 4: Original handwritten digits (the eight digits shown in the lower half of the figure) and their corresponding nearest codebook vectors (upper half).

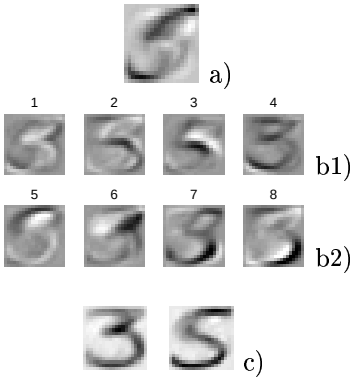

Figure 5: a) The origin of the feature set. b) The complete feature set consisting of eight features. c) The codebook vectors representing a 3 and a 5 are combinations of features 4, 6, 8 and 3,5,7,8, respectively.

and the image in the right is a combination of features 3, 5, 7 and 8.

Using the same preprocessed 8×8 pixel images as used in [7] and [1], we tested a naive classification method. For each digit $0, \cdots, 9$ we used GVQ to learn a separate codebook from 400 examples using 10 features. After learning, the codebooks were joined in one big set and the test patterns (700 for each digit) were classified with the class label of their closest codebook vector. The classification error was 6.0%, slightly worse than the result reported by [7] and [1], 4.6% and 4.8%, respectively, but better than the result of nearest neighbour classification, 6.7% . Our results, using a modest amount of CPU time, are encouraging. We are currently looking into ways of improving the classification accuracy of the method.

3.2 Image Compression

A well known application of vector quantisation is image compression. We used GVQ to compress the image[1] in figure 6-a and compared the result with standard vector quantisation.

If a codebook vector in GVQ is constructed out of a set of n_f features then we need maximally n_f bits to determine a codebook vector. In general the number of bits will be less since some feature combinations are never used. Therefore, if the number of used feature combinations is N_{gvq} we need $\log N_{gvq} < n_f$ to determine a codebook vector in GVQ. Similarly, in standard VQ we need $\log N_{vc}$ bits to determine a codebook vector if N_{vc} is the number of learned codebook vectors.

Consider the case that the image to be compressed is particular in the sense that we can not use features (or codebook vectors) which were used to encode previously encountered images. In that situation we need to consider the information to describe the features (codebook vectors) themselves. This information is proportional to the number of pixels n_p used in a feature and the information 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, then the information in the image after compression with standard VQ is

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

and with GVQ it is

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

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 segments of 16×16 pixels. For the standard VQ algorithm we used $N_v = 16$ codebooks to compress the image into a $I_{VQ} = 74$ kbit image, figure 6-b. We then applied our GVQ algorithm using $N_f = 8$ features to compress

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

Figure 6: a) The Vermeer image prior to compression consists of 865 kbits. After compression: b) With standard VQ $I_v = 74$ kbits, c) with GVQ using 1 set of 8 features $I_{GVQ} = 56$ kbits.

the image in a string of $I_{GVQ} = 56$ kbits, figure 6-c. While containing 8 kbits of information less, the GVQ compressed image is without doubt a superior representation of the original image.

4 Relation to other work

There is a close similarity between the GVQ learning algorithm and the EM-algorithm which is used for learning probabilistic models. The E-step which makes a probabilistic association between a data point and a hidden state $\mathbf{s}$ is replaced with a hard 0-1 association step (step 2 in the algorithm). The M-step, in which the likelihood is maximised, is replaced with the distance minimisation step (step 3 in the algorithm). In fact, in the limit $\sigma \to 0$, the GVQ algorithm is equivalent to the EM-algorithm applied to the following mixture model

$$p(\mathbf{x}^\mu) \propto \sum_{\mathbf{s}} p(\mathbf{s}) e^{-\frac{1}{2\sigma^2}\|\mathbf{x}^\mu - F\mathbf{s}\|^2}, \qquad (4)$$

where the state probabilities $p(\mathbf{s})$ are either 0 (corresponding to codebook vectors which moved out of the data cloud) or equal to a constant value. In the limit $\sigma \to 0$ the data point $\mathbf{x}^\mu$ will be associated with a single state $\mathbf{s}$, namely the one for which $F\mathbf{s}$ has the smallest Euclidian distance to $\mathbf{x}^\mu$. The M-step then corresponds to minimising the sum of these distances with respect to F.

Viewed as the limiting case of the probability model (4), some interesting connections can be made to the work of [4] and also [3] which has recently come to our attention. The main difference between the models these authors propose and Eq. (4) is in the assumption for the hidden state distribution $p(\mathbf{s})$. Both consider *continuous* hidden state variables. Attias [3] considers a product of Gaussian mixture distributions (independent factors) for the hidden states $\mathbf{s}$. If, in that model, the number of mixtures for each hidden state variable s_i is set to 2 (bi-modal distribution), the generated patterns are essentially binary feature combinations as in GVQ. Olshausen and Field [4], on the other hand, 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. A similar property can be incorporated in our method by extending the basic constant GVQ prior $p(\mathbf{s})$ in Eq. (4) with a soft prior $p(\mathbf{s}) \propto \exp(-\lambda \mathbf{s}^T\mathbf{s})$. 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.

5 Discussion

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. In terms of Vector Quantisation, this means that data points are associated with only the nearest codebook vector. Unlike a probabilistic model, GVQ constructs a competition between alternative explanations for a data point, in which there can be only one winning explanation. We saw that in handwritten character recognition, the features learned were able to generate good representations of the data. Similarly, image compression was markedly improved by using GVQ over standard VQ.

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

References

[1] B. Sallans, G.E. Hinton, and Z. Ghahramani. A hierchical community of experts. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, 269–284. Springer-Verlag, 1998.

[2] Z. Ghahramani and G. E. Hinton. Hierarchical non-linear factor analysis and topographic maps. In *NIPS*, 10. MIT Press, 1998.

[3] H. Attias Independent Factor Analysis *Neural Computation*, 11(4):803–851, 1999.

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

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

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

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