GTM: The Generative Topographic Mapping

Johan Fredrik Markus Svensén

Doctor Of Philosophy

Aston University

April 1998

Aston University

GTM: The Generative Topographic Mapping

Johan Fredrik Markus Svensén

Doctor Of Philosophy, 1998

Thesis Summary

This thesis describes the *Generative Topographic Mapping* (GTM) — a non-linear latent variable model, intended for modelling continuous, intrinsically low-dimensional probability distributions, embedded in high-dimensional spaces. It can be seen as a non-linear form of principal component analysis or factor analysis. It also provides a principled alternative to the self-organizing map — a widely established neural network model for unsupervised learning — resolving many of its associated theoretical problems.

An important, potential application of the GTM is visualization of high-dimensional data. Since the GTM is non-linear, the relationship between data and its visual representation may be far from trivial, but a better understanding of this relationship can be gained by computing the so-called *magnification factor*. In essence, the magnification factor relates the distances between data points, as they appear when visualized, to the actual distances between those data points.

There are two principal limitations of the basic GTM model. The computational effort required will grow exponentially with the intrinsic dimensionality of the density model. However, if the intended application is visualization, this will typically not be a problem. The other limitation is the inherent structure of the GTM, which makes it most suitable for modelling moderately curved probability distributions of approximately rectangular shape. When the target distribution is very different to that, the aim of maintaining an 'interpretable' structure, suitable for visualizing data, may come in conflict with the aim of providing a good density model.

The fact that the GTM is a probabilistic model means that results from probability theory and statistics can be used to address problems such as model complexity. Furthermore, this framework provides solid ground for extending the GTM to wider contexts than that of this thesis.

Keywords: latent variable model, visualization, magnification factor, self-organizing map, principal component analysis

Acknowledgements

I would like to thank my supervisors Prof. Chris Bishop and Dr Chris Williams for inspiring, guiding and supporting me through my PhD studies. Several of the key ideas of the GTM were developed in collaboration with them, as reflected by references to appropriate publications (for a list, see section 1.2.1). Their supervision has made my time as a PhD student a very rewarding experience, with many challenges and hard work, but also a lot of fun.

Many of the members of the Neural Computing Research Group (NCRG) at Aston University during the years 1995–1997 gave me help and support, ranging from solving problems with computers and software, to sharing their deep insigths into complex issues with me. Maybe most important, they gave me their time, listening to my (probably incomprehensible) explanations of problems I had, and thereby helped me to understand these problems better myself. Some of them also gave concrete advice and ideas on how my work could be improved. In particular, I would like to thank Dr Mike Tipping (who also provided the LaTeX style used to produce this thesis), Iain Strachan, Dr Ian Nabney and Dr Cazhaow Qazaz.

I would also like to thank the SANS group of the Department of Numerical Analysis and Computer Science at the Royal Institute of Technology in Stockholm, who provided a hospitable 'home' with an Internet connection for me, during my summer visits in Sweden, 1995–1997. Especially, I would like to thank Anders Holst and Anders Lansner.

Figures 1.2 and 3.1 were provided by Prof. Chris Bishop. (Figure 3.1 was modified by me.) The surface plots in figure 5.2 were produced using the equisurf package for MATLAB®, produced by Lennart Bengtsson, Chalmers University of Technology, Sweden.

During my time at Aston, many people have supported me on a more personal level. I would like to thank my parents and my sister for always encouraging me and supporting me, not to mention all practical help they have provided. I would also like to thank my aunt Kina in Gothenburg for the many times I was staying with her, on my trips between Sweden and England.

My life in England would not have been half as fun and many times a lot more difficult, if it had not been for the many friends I met 'along the way'. In particular I would like to thank Chris Murphy, Lisa Loveridge, Rosie Stephenson, Emma Young, Heidi Volp, Michał Morciniec, Ansgar West, Ian Baxter, Tony Walton and Lily Xanthou. This list should really be much, much longer, including flatmates, members of the NCRG that were much more than just colleagues, the members of the 'God Squad' and the members of the infamous *Aston University Mountaineering Club* (AUMC).

Finally, I would like to thank Anna, my girlfriend, for all her love, support, patience and understanding, and for all the fun we have together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The amount of data being recorded and stored throughout society is steadily growing. This is largely due to the increased use of modern technology in general and computers in particular. Satellites in orbit are generating vast amounts of data in terms of imagery and geodesic data. Transactions on the global financial markets, via computerized systems, generate complex time series data. With increased competition, companies are building sophisticated customer databases in attempts to analyze current and future markets. In micro-biology, we now access the large quantity of data stored in the DNA of living organisms.

However, without means and methods that can aid analysis, much data becomes useless. Human observers often find it hard spotting regularities when looking at raw data, e.g. tables of numbers and symbols or large numbers of similar images. We therefore need computers to aid us, not only in the gathering and storing of data, but also in the analysis and processing of it. In particular, if the computer can be used to summarize data visually, humans are often capable of interpreting such graphical summaries intelligently.

1.1 Scope of this thesis

This thesis is concerned with computational methods for finding 'interesting' structures in sets of data, with little or no need of human intervention or guidance. A number of such methods has been known for quite some time. A key feature of many of them is that they involve some sort of dimensionality reduction, from the, typically high-dimensional, *data space* to a low-dimensional *model space* defined by the method used. When visualization is the ultimate aim, the model space is typically chosen to be two-dimensional. In this thesis, both the data space and the model space are taken to be subsets of $\Re^\infty$. Moreover, we will restrict our interest to global structures, i.e. continuous low-dimensional manifolds embedded in high-dimensional continuous spaces. For a long time, models with this scope were restricted to model only linear structures, i.e. hyper-planes, in the data space. We will direct our interest to models where the relationship between model and data space is non-linear, as illustrated in the right half of figure 1.2.

An important reason why the linear models for long were dominating is their computational efficiency. However, the arrival of fast, inexpensive computers has, in the last two decades, changed the picture dramatically. This has combined with discoveries of new computational algorithms and today we are tackling problems which twenty years ago would have been considered untractable. Many of these new algorithms have been inspired by models of the processing going on in the human brain.

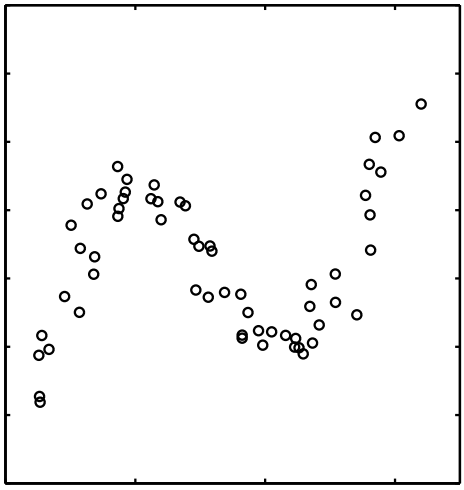


Figure 1.1: An example of data which, although living in a 2-dimensional space, is approximately 1-dimensional. Any good method for reducing the dimensionality of this data must be able to deal with the fact that it is *non-linearly* embedded in the 2-dimensional space.

In particular, there has been a lot of interest in generic algorithms that can 'learn' an underlying structure from a finite set of examples, in a fashion similar to human learning. For many problems this is highly desirable, since a human observer may easily discover regularities in a set of examples, but will find it much harder to describe *how* he or she made this discovery. Consider, for example, the set of points shown if figure 1.1: most human observers would, when asked to comment on this data, immediately point out that the points appear to be distributed, approximately, along a curved line. However, it is unlikely that anyone of them would be able to provide a description of how they arrived at this conclusion, which would be sufficiently exact to translate into computational algorithm.

This has motivated the development of algorithms that, to a certain extent, try to mimic the processes that takes place in the human brain, in terms of discovering and exploiting underlying structures of finite data sets. These algorithms have become known under the common name of *neural networks* [Haykin, 1994]. In parallel, similar algorithms have grown out of the research into statistical pattern recognition [Duda and Hart, 1973], and the strong links between these two fields are today widely appreciated [Bishop, 1995, Ripley, 1996].

To return the focus to the problems we are interested in, our underlying assumption is that although we observe a large number (D) of data variables, these are being generated from a smaller number (L) of hidden, or latent, variables, as illustrated by figure 1.2. Models based on this assumption are called *latent variable models* [Bartholomew, 1987, Everitt, 1984] and have evolved, initially from psychology, to become established statistical models for data analysis. When both latent and observed variables are real valued and the relationship between them is linear, the resulting model is traditionally known as factor analysis [Bartholomew, 1987, Lawley and Maxwell, 1971] and will be further discussed in the next chapter. To allow the relationship between the latent and the data space to be non-linear, we consider a non-linear, parameterized mapping from the latent space to the data space. This will map every point in the latent space to a corresponding point in the data space. If we assume that the mapping is smooth, these points will be confined to an L-dimensional, curved manifold in the D-dimensional data space. If we then define a distribution over the latent space, this will induce a corresponding distribution over the manifold the data space, establishing a probabilistic relationship

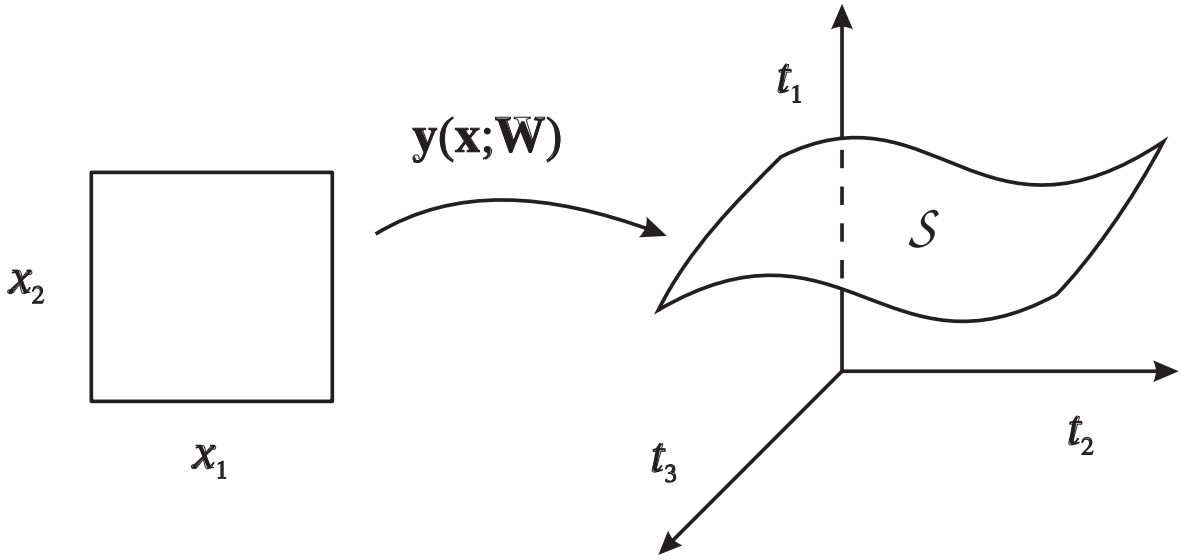


Figure 1.2: A non-linear latent variable model, where the latent variables, x_1 and x_2 are mapped, by the non-linear, parameterized mapping $\mathbf{y}(\mathbf{x}, \mathbf{W})$, to a low-dimensional, curved manifold, $\mathcal{S}$, in the data space defined by t_1, t_2 and t_3.

between the two spaces. The challenge will be to adapt the parameterized mapping so as to fit the density model in the data space to a set of training data.

Assuming we have been able to fit our model, we can then relate points in the data space to points on the curved manifold, which in turn correspond to points in the latent space. This way, we can give points in the high dimensional data space a representation in the low-dimensional latent space, and provided the data space has no more than three dimensions, we can visualize this representation.

The introduction of non-linearity will offer the possibility of modelling a much richer 'family' of structures, compared to what we can achieve with linear models. However non-linearity also brings potential problems. Since real data is typically corrupted by noise, there is a risk that a non-linear model captures not only systematic non-linearities in a set of data, but also random artifacts due to noise. Another problem, if we consider visualization of high-dimensional data, is that the relationship between the data and its 'representation' in the model is not as straightforward when using a non-linear model as when using a linear model. Although these potential problems should not stop us from making use of non-linear models, we must be aware of their presence and try to resolve them, as far as possible.

1.2 Overview of this thesis

In chapter 2, we review a number of models proposed for finding and exploiting structure in high-dimensional data sets. These models are divided into three categories:

- projection models, which are based on the notion of 'projecting' points in the data space onto the model space,
- generative models, where points in the data space are considered to have been generated by points in the model space, and
- other models, which do not belong to either of the above categories.

The most important of the projection models is linear principal component analysis (PCA) [Jolliffe, 1986]; the other models discussed in that category can all, in some sense, be regarded as non-linear variants of PCA. Correspondingly, the section on generative models is headed by factor analysis (FA), which, traditionally has been regarded as the generative counterpart of PCA, followed by non-linear generative models. The final section on other models is primarily concerned with the *self-organizing map* (SOM) [Kohonen, 1995], a widely researched neural network model which is strongly related to the *generative topographic mapping* (GTM), introduced in chapter 3.

The chapter on the GTM describes its architecture, the associated training algorithm and how it can be used for visualization of data. It goes on to discuss the relationship to some of the other models from chapter 2, in particular the SOM.

In chapter 4, we try to investigate the relationship between latent space and data space, defined by the nonlinear mapping, by evaluating the 'stretching' of the manifold forming the image of the latent space in the data space. This will allow us to extract additional information from our model, in terms of how points in the latent space are related to the corresponding points in the data space, information which can be merged with visualized data. A method along these lines has been proposed for the SOM, but has been restricted by the fact that the original SOM model does not define an explicit manifold in the data space. However, we will see how the method proposed can also be used with certain, more recent versions of the SOM, provided certain conditions are met.

Chapter 5 addresses the issue of parameter selection and its relationship to model complexity. Two principal methods for finding suitable parameter values are discussed: cross-validation and approximate Bayesian maximum a-posteriori, the latter of which offers different variants. While both methods can be used *offline*, by simply evaluating the appropriate score for different parameter values and then choose those with the best score, the Bayesian methods can also, to some extent, be used in an *online* setting, where appropriate parameter values are found during training, thereby offering dramatic savings in terms of computation. The methods are evaluated using synthetic data.

Directions for future work are suggested in chapter 6, in some cases accompanied by some provisional results. These include potential ways of dealing with known limitations of the GTM model as it stands today, as well as possibilities for it to be used in contexts different to that of this thesis, e.g. data with categorical variables, incomplete data, and mixtures of GTM models. Finally, chapter 7 gives a concluding discussion.

The reader who is only interested in the GTM model can skip chapter 2, and go directly to chapter 3, skipping section 3.4. Subsequent chapters are independent of each other, but assume that the reader has read the chapter on the GTM. Section 4.4 discusses magnification factors for the batch version of the SOM model (BSOM); for readers unfamiliar with this model, it is described in section 2.3.1 and further discussed in section 3.4.1.

1.2.1 Publications on the GTM

This thesis gathers and complements the material in earlier publications on the GTM:

- *EM Optimization of Latent-Variable Density Models*, presented at Neural Information Processing Systems (NIPS), Denver, Colorado, 1995, chapter 3,
- *GTM: A Principled Alternative to the Self-Organizing Map*, presented at the International Conference on Artificial Neural Networks (ICANN), Bochum, 1996, chapter 3,
- *GTM: A Principled Alternative to the Self-Organizing Map*, presented at NIPS, 1996, chapter 3,

- *Magnification Factors for the SOM and GTM Algorithms*, presented at the Workshop on Self-Organizing Maps (WSOM), Helsinki, 1997, chapter 4,
- *Magnification Factors for the GTM Algorithm*, presented at the IEE International Conference on Artificial Neural Networks, Cambridge, 1997, chapter 4,
- *GTM: The Generative Topographic Mapping*, published in Neural Computation, 1998, chapter 3, and
- *Developments of the GTM Algorithm*, to appear in Neurocomputing, chapter 6.

The chapter numbers given refer to the chapter of this thesis where the main content of the corresponding paper can be found. These papers are all authored by C. M. Bishop, M. Svensén and C. K. I. Williams, and are also listed with further details in the bibliography.

Before moving on, we now introduce some notation and conventions used throughout the rest of this thesis.

1.3 Notation and conventions

In the mathematical notation, the convention will be that an italic typeface indicates scalar values, e.g. t_{nd}, x, ϕ, while bold typeface indicates vectors and matrices, the former using lower case symbols, e.g. $\mathbf{t}, \boldsymbol{\psi}_k$, and the latter using upper case symbols, e.g. $\mathbf{X}, \boldsymbol{\Phi}$. Note, however, that exceptions to this convention do appear.

Our aim is to build a model of a probability distribution in $\Re^D$, based on a finite set of independently drawn samples from this distribution, $\mathbf{t}_1, \ldots, \mathbf{t}_n, \ldots, \mathbf{t}_N$. We will denote this data set $\mathbf{T}$; typically, we organize the samples into a $N \times D$ matrix, where row n contains sample $\mathbf{t}_n$, and $\mathbf{T}$ will also be used to denote this matrix. Individual elements in this matrix or, equivalently, elements of sample $\mathbf{t}_n$, will be denoted t_{nd}. As we will see, a key assumption about $\mathbf{T}$ is that the samples are *independent, identically distributed*, commonly abbreviated *i.i.d.*

Also in the low-dimensional model space ($\Re^L$) we will be working with a finite set of points, $\mathbf{x}_1, \ldots, \mathbf{x}_k, \ldots, \mathbf{x}_K$, which may or may not be in one-to-one correspondence with the points in $\mathbf{T}$. We use $\mathbf{X}$ to denote this set of points as well as the corresponding $K \times L$ matrix. These points will map to a corresponding set of points, $\mathbf{y}_1, \ldots, \mathbf{y}_k, \ldots, \mathbf{y}_K$ in the data space, denoted $\mathbf{Y}$, which also denotes the corresponding $K \times D$ matrix.

Thus, D denotes the dimensionality of the data space, L the dimensionality of the latent space, N the number of data points and K the number of latent points. As far as it is possible, without compromising clarity, matching indices will be used, so that d is used as index over the dimensions of the data space, k is used as index over latent points, etc.

At various occasions we will make use of the identity matrix $\mathbf{I}$ (a diagonal, square matrix with ones along diagonal and zeros elsewhere). Normally, the size of $\mathbf{I}$ will be given implicitly by the context, so in the equation

$$\mathbf{A} = \mathbf{B} + \mathbf{I}$$

where $\mathbf{A}$ and $\mathbf{B}$ are $M \times M$ matrices, $\mathbf{I}$ is understood to also be $M \times M$.

Unless indicated otherwise, summations will start from 1, and we will use the abbreviation

$$\sum_{n,k}^{N,K} = \sum_{n}^{N} \sum_{k}^{K} .$$

We will also use abbreviations 1-D, 2-D, etc. for 1-dimensional, 2-dimensional, etc.

Chapter 2

Modelling Low-dimensional Structure

The problem of finding low-dimensional representations of high-dimensional data is not new and a considerable number of models have been suggested in the literature. The rest of this chapter will review some of those models, broadly categorized into

- projection models,
- generative models and
- other models.

Projection models are, loosely speaking, based on 'projecting' the data, e.g. by orthogonal projection, on the model — fitting those models corresponds to minimizing the distances between data and its projection. Generative models try to model the distribution of the data, by defining a density model with low intrinsic dimensionality in the data space. The borders between the three categories are not clear cut and, as will be seen in the following sections, there are models that fit in more than one category.

2.1 Projection models

The traditional meaning of projection is the orthogonal projection of a point in $\Re^D$, onto a hyperplane, $\Re^L \subseteq \Re^D$, where $L \leq D$. This is also the method of projection used in *principal components analysis* (PCA), the most commonly used of the projection models described here. The fact that PCA defines a linear, orthogonal model space gives it favourable computational properties, but it is also its main limitation. Therefore, a number of models have been suggested that allow for non-linearity, either by using a combination of locally linear models, which together form a non-linear structure, or through the use of a globally non-linear model. However, before coming to these models we first consider standard linear PCA.

2.1.1 Principal component analysis

Principal components analysis [Jolliffe, 1986] takes a data set, $\{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_N\}$, in a given orthonormal basis in $\Re^D$ and finds a new orthonormal basis, $\{\mathbf{u}_1, \ldots, \mathbf{u}_D\}$, with its axes ordered. This new basis is

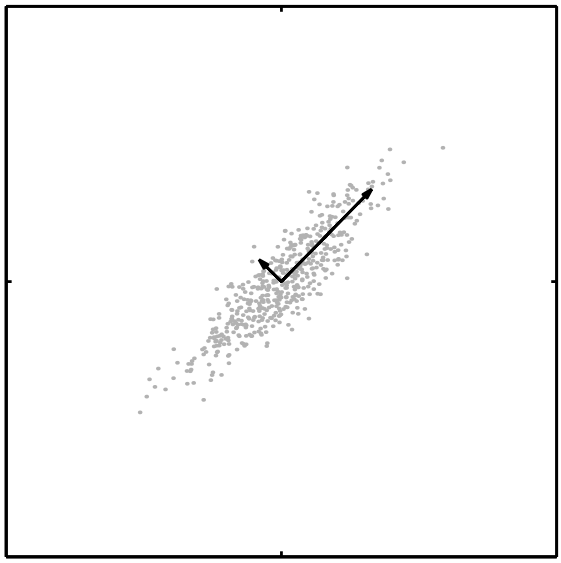


Figure 2.1: The plot shows a data set of 500 points, plotted as shaded dots, $\cdot$, drawn at random from a correlated Gaussian distribution. The two arrows represent the principal components, scaled by the square root of the corresponding variance times two — $2\mathbf{u}_i\sqrt{\lambda_i}$, $i = 1, 2$, in the terminology of equation (2.1).

rotated in such a way that the first axis is oriented along the direction in which the data has its highest variance. The second axis is oriented along the direction of maximal variance in the data, orthogonal to the first axis. Similarly, subsequent axes are oriented so as to account for as much as possible of the variance in the data, subject to the constraint that they must be orthogonal to preceeding axes. Consequently, these axes have associated decreasing 'indices', λ_d, $d = 1, \ldots, D$, corresponding to the variance of the data set when projected on the axes, which we hence refer to as *variances*. The principal components are the new basis vectors, ordered by their corresponding variances, with the vectors with the largest variance corresponding to the first principal component[1]. Figure 2.1 shows an example for a 2-dimensional data set.

By projecting the original data set on the L first principal components, with $L < D$, a new data set with lower dimensionality can be obtained. If the principal components are first scaled by the corresponding inverse variances, the variables of the new data set will all have unit variance — a procedure known as whitening or sphering [Fukunaga, 1990, Ripley, 1996].

The traditional way of computing the principal components is to compute the sample covariance matrix of the data set,

$$\mathbf{S} = \frac{1}{N-1}\sum_n^N (\mathbf{t}_n - \overline{\mathbf{t}})(\mathbf{t}_n - \overline{\mathbf{t}})^{\mathrm{T}}, \qquad \overline{\mathbf{t}} = \frac{1}{N}\sum_n^N \mathbf{t}_n,$$

and then find its eigen-structure

$$\mathbf{SU} = \mathbf{U}\mathbf{\Lambda}. \tag{2.1}$$

$\mathbf{U}$ is a $D \times D$ matrix which has the unit length eigenvectors, $\mathbf{u}_1, \ldots, \mathbf{u}_D$, as its columns and $\mathbf{\Lambda}$ is

[1] There seems to be some disagreement regarding the terminology in the literature — sometimes it is the new variables obtained by projecting the data set on the new (scaled) basis vectors that are referred to as the principal components, and there are also examples where it is the variances.

diagonal matrix with the corresponding eigenvalues, $\lambda_1, \ldots, \lambda_D$, along the diagonal. The eigenvectors are the principal components and the eigenvalues are the corresponding variances.

An alternative method for computing the principal components, which is claimed to be more robust [Ripley, 1996], is to compute the singular value decomposition (SVD) [Strang, 1988, Press et al., 1992] of the $N \times D$ matrix, $\mathbf{T}$, containing the data set, so that

$$\mathbf{T} = \mathbf{V}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}, \tag{2.2}$$

where $\mathbf{V}$ is a $N \times D$ matrix with orthogonal columns, $\mathbf{U}$ is a $D \times D$ orthogonal matrix and $\mathbf{\Lambda}$ is a diagonal matrix with the singular values of $\mathbf{T}$ along the diagonal. As the notation suggests, $\mathbf{U}$ and $\mathbf{\Lambda}$ have the same values in (2.1) and (2.2).

An important property of the principal components is that they constitute the unique set of vectors, up to scaling, that minimizes the *reconstruction error*,

$$E_L = \sum_{n}^{N} \|\mathbf{t}_n^{\mathrm{T}} - (\mathbf{t}_n^{\mathrm{T}}\hat{\mathbf{U}}_L)\check{\mathbf{U}}_L^{\mathrm{T}}\|^2, \tag{2.3}$$

where $\hat{\mathbf{U}}_L$ and $\check{\mathbf{U}}_L$ are $D \times L$ matrices with the (scaled) principal components $\mathbf{u}_1, \cdots, \mathbf{u}_L$ $(1 \leq L \leq D)$ as their columns, such that $\check{\mathbf{U}}_L^{\mathrm{T}}\hat{\mathbf{U}}_L = \mathbf{I}$. E_L is the sum of the squared distances between the data points and their projections on the L principal components, summed over the data set. Thus, it is a decreasing function of L, equal to zero when $L = D$. Under this formulation, PCA is known as the Karuhnen-Loéve transform, and it suggests an alternative way of finding the principal components, by minimizing (2.3). This approach has formed the basis for non-linear extensions, known as auto-associative networks or auto-encoders, discussed in section 2.1.3 below.

Mixtures of PCA

Since PCA only defines a linear subspace, it will be sub-optimal when the underlying structure in the data is non-linear. However, even if we have reasons to assume that the data we are dealing with is not overall linear, it may still be reasonable to assume that in local regions of the data space, a linear approximation is sufficient. How good such an approximation will be, will depend how strong the non-linearity in the data is and how small we choose our local regions. Based on this assumption, there has been a number of suggestions for combining a number of local PCA models, to approximate a globally non-linear structure. Kambhatla and Leen [1994] partitions the data space using vector quantization and then performs 'local' PCA on the data points assigned to each vector quantizer. Bregler and Omohundro [1994] takes a more elaborate approach, finding an initial model using K-means and local PCA, which is then refined using the EM-algorithm [Dempster et al., 1977] and gradient descent. Hinton et al. [1995a] suggest an iterative scheme where data points are assigned, initially e.g. by using K-means, to the PCA component which reconstructs them best, local PCA is performed and then points are re-assigned. This is repeated until no points have their assignment changed. They also suggest a 'soft' version of this algorithm, further discussed in [Hinton et al., 1997], where data points are 'softly' assigned to PCA components, based on the corresponding reconstruction errors. However, all these algorithms have some degree of arbitrariness associated with them.

Recently, a new, probabilistic formulation of PCA has been proposed by Tipping and Bishop [1997a]. It derives PCA as a latent variable model, and can be regarded as a special case of factor analysis. It defines a generative model, which allows for mixtures of PCA to be constructed with in the framework of probability theory. Further discussion of this model is deferred to section 2.2.2, following the introduction of the factor analysis model.

2.1.2 Principal curves and surfaces

Principal curves and surfaces have been suggested as non-linear generalizations of principal component analysis. In contrast to the mixture models discussed above, principal curves and surfaces represent single, global models.

Principal curves

Intuitively, a principal curve [Hastie and Stuetzle, 1989, Hastie, 1984] is a smooth, one-dimensional curve that passes through the 'middle' of a probability distribution, or a cloud of data points, in a D-dimensional space. More formally, a principal curve is a parameterized curve, $\mathbf{f}(x)$, that respects the definition of *self-consistency*,

$$\mathbf{f}(x) = E[\mathbf{t}|\lambda_{\mathbf{f}}(\mathbf{t}) = x], \tag{2.4}$$

where $\mathbf{t}$ is a random variable in the data space and $\lambda_{\mathbf{f}}(\mathbf{t})$ is the 'projection' of $\mathbf{t}$ on the curve defined by $\mathbf{f}(\cdot)$. This says that for any point x, $\mathbf{f}(x)$ equals the average of the probability mass that is projected on x under the projection index $\lambda_{\mathbf{f}}(\cdot)$, which in the principal curve and surface models is the orthogonal projection.

For a finite data set, this definition must be modified, so Hastie and Stuetzle [1989] use a scatter-plot smoothing procedure, which replaces the averaging of continuous probability mass with a smoothed average over data points projecting in the same region on $\mathbf{f}(\cdot)$. This leads to the following procedure for finding principal curves:

1. Set the initial principal curve, $\mathbf{f}$, equal to the first principal component and for each data point, $\mathbf{t}_n$, $n = 1, \ldots, N$, compute its orthogonal projection on $\mathbf{f}$, x_n.
2. Compute a new value for each point on the curve, $\mathbf{f}(x_n)$, by a smoothed average over data points projecting in the neighbourhood of x_n.
3. Project the data points (numerically) on the new curve.
4. Repeat steps 2 and 3 until convergence.

As noted by Hastie and Stuetzle, the size of the neighbourhood used in step 2 can have a significant impact on the final shape of the curve. In essence, the size of the neighbourhood controls the smoothness of the curve. Hastie and Stuetzle set the neighbourhood, measured by the number of neighbouring data points it includes, to an initial size which is then gradually decreased during the iterative fitting procedure until it reaches a desired value. Similar use of weighted averaging over a shrinking neighbourhood appears in the fitting procedure for the self-organizing map, discussed in section 2.3.1. The algorithm above is the first example of the iterative two-step fitting procedures associated with many of the non-linear models that will be discussed below.

Hastie and Stuetzle [1989] show that $\mathbf{f}(\cdot)$, restricted to come from a smooth class of curves, minimizes

$$E[\|\mathbf{t} - \lambda_{\mathbf{f}}(\mathbf{t})\|^2],$$

which is the expected distance between $\mathbf{t}$ and its projection on the curve $\lambda_{\mathbf{f}}(\mathbf{t})$, taken over the distribution of $\mathbf{t}$. This is analogous with the property (2.3) of principal components, emphasizing the relationship between the two models.

Webb [1996] proposes an alternative model for doing non-linear PCA which is based on the self-consistency condition (2.4), but where the orthogonal projection on the curve[2] has been replaced by a more general mapping. Thus, this model forms a link between the principal curve model and the auto-associative neural network model discussed in section 2.1.3. Tibshirani [1992] proposed an alternative definition of the principal curve, transforming it into a generative model, which will be further discussed in section 2.2.4.

Principal surfaces

Principal surfaces are discussed by Hastie and Stuetzle [1989] as an extension of the principal curve, which allows the model space to be two-dimensional. The definition is based on self-consistency, analogous with the one for principal curves, and Hastie and Stuetzle [1989] report that they have implemented a corresponding principal surface algorithm, using two-dimensional surface-smoothers in place of the scatter-plot smoothers. However, many of the theoretical results obtained for the principal curve model no longer hold in the case of principal surfaces.

An alternative definition of principal surfaces is proposed by LeBlanc and Tibshirani [1994], which combines ideas of principal curves and multi-adaptive regression splines (MARS) [Friedman, 1991]. The resulting model defines a low-dimensional, piecewise linear structure, which is built in a incremental fashion. An important difference compared to many of the other models considered here is that dimensionality of the model is determined as part of the procedure fitting the model to data. This is based on minimizing the distance between data points and their projections onto the principal surface and consists of two main steps: the first which grows the model, the second which prunes it. A part of both these steps is what LeBlanc and Tibshirani call model re-fitting, which is similar to the fitting procedure for principal curves, alternating between projection on and adaption of the surface.

2.1.3 Auto-associative feed-forward neural networks

An alternative and rather different approach to non-linear PCA is the use of auto-associative networks [Kramer, 1991], also known as auto-encoders. These are feed-forward neural networks which are trained to implement the identity function, i.e. to map a vector to itself, through a 'bottleneck', encouraging it to find and exploit an underlying, simpler structure in the training data. Figure 2.2 shows a schematic picture of an auto-associative network, where a D-dimensional data vector is fed as input *and* target, but the mapping goes via an L-dimensional space (the model space), with $L < D$.

If all the units in this network are taken to be linear, in which case any intermediary layers between inputs and targets and the bottleneck layer can be removed, and the network is trained using the sum-of-squares error function, this training corresponds to the minimization of the reconstruction error in equation (2.3). This will result in the network performing standard PCA with L principal components [Baldi and Hornik, 1989]. In fact, it can be shown that this will also be the case for a network with a single bottleneck layer of non-linear units [Bourlard and Kamp, 1988]. However, if we instead use a network with intermediary layers of non-linear units before and after the bottleneck layer, this allows the network to find non-linear structures in the data, which we can interpret as a form of non-linear PCA. Another interesting implication is that if we allow the number of units in the intermediary layers to exceed D, we could also consider letting $L > D$ and find more principal components than there are dimensions in the data.

[2] This model readily extends to more than one non-linear component, but for simplicity we refer to the resulting low-dimensional structure as a curve.

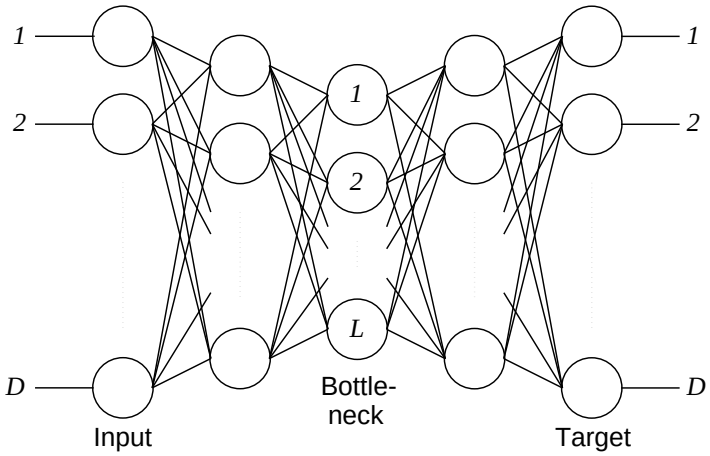


Figure 2.2: An auto-associative network that maps a D-dimensional input vector to itself, through a L-dimensional bottleneck layer, $L < D$, possibly preceded and followed by layers of non-linear units.

Zemel and Hinton [1994] develop an alternative formalism in the context of auto-encoders, where each hidden unit instead represents a quantization vector, with the auto-encoder implementing a vector quantizer. In this model, there are no intermediate layers between input and target layers and the bottleneck layer, and the units in the bottleneck layer, of which there are typically many more than there are inputs, form their activations as soft-max transformations [Bridle, 1990] of their net input. However, each hidden unit also has a adjustable location in a low-dimensional *implicit* space. During training, the parameters of the model are adjusted to reconstruct the input vector on the targets, while driving the activations of the hidden units towards forming a Gaussian bump in the implicit space. This means adjusting the position of each hidden unit, both in the input space and in the implicit space, so that units which have nearby locations in the implicit space respond to nearby vectors in the input space.

2.1.4 Kernel based PCA

Kernel based methods, most prominently represented by the non-linear support vector machine [Cortes and Vapnik, 1995], offer promise for non-linear extensions to many methods based on the inner product, or dot product, of vectors. Using the kernel based methods, we can consider mapping two vectors in the input space to a high (maybe even infinite) dimensional feature space and then compute the inner product of the resulting vectors in the feature space. The relationship between vectors in the input space and their images in the feature space need not be linear. The important point, however, is that the image vectors are actually never computed explicitly; all that is computed is their inner product, using a so-called kernel function.

Schölkopf et al. [1996] describe a method for doing non-linear, kernel based PCA. It corresponds to doing ordinary linear PCA, but doing it in the implicitly defined feature space. As with auto-encoders, this has the interesting implication that we can find more principal components than observed variables. Another implication, which makes this method less interesting compared with the other models discussed here, is that, in general, we cannot get hold of the actual principal components. We can compute the projection of a new point in the data space onto the principal components, but it is difficult to explore the relationship in the other direction, i.e. how variance along the principal components is reflected in the input space.

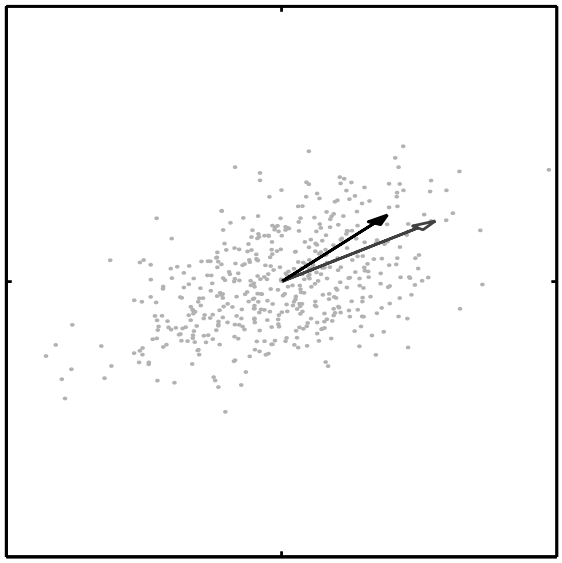


Figure 2.3: The plot shows the data set from figure 2.1, but here the noise on the horizontal variable has been scaled up. Consequently, the the first principal component, shown as the lower grey arrow with hollow head, is approaching the horizontal axis, while the vector corresponding to one-factor FA model, although affected by the increasing noise, stay closer to the 45-degree line, representing the covariance between the data variables.

2.2 Generative models

The projection models discussed in the previous section aim at finding low-dimensional manifolds in the space of the data, such that the distance between data and its projection on the manifold is small. The generative models that are to be discussed in the sections to come, try to model the density function that is assumed to have generated the data, under a set of constraints that restricts the set of possible models to those with a low intrinsic dimensionality.

2.2.1 Factor analysis

Traditionally, factor analysis (FA) [Bartholomew, 1987, Lawley and Maxwell, 1971] has been the 'generative cousin' of PCA; in fact, the two techniques are sometimes confused. The key difference is that where PCA is focusing on variance, FA focus on *co*variance. Covariance between a set of observed variables is seen as an indication that these variables are, if only to a certain extent, functions of a common latent factor. The effect of this difference becomes apparent when the observed variables are subject to significantly different noise levels. While PCA will try to capture all variance in the data, including variance due to noise affecting only individual variables, FA will focus on the covariance, regarding additional variability in the observed variables as noise. Figure 2.3 illustrates this for a two-dimensional data set.

FA has, just like PCA, a long history, dating back to the beginning of the century. It was developed by psychologists with the aim to explain results from cognitive tests in terms the underlying organization of mental abilities. Since then, a number of variants have been suggested, differing primarily in their estimation procedures for the model parameters. Due to this diversity, and maybe also to its origin, FA was for a long time looked at with some scepticism, as lacking a solid statistical foundation. A method for maximum likelihood estimation of the parameters in the FA model was

proposed by Lawley [1940], but this method had a number of practical disadvantages and it was not until when Jöreskog [1967] proposed an alternative maximum likelihood method that FA got a wider acknowledgment as a useful statistical tool.

Rubin and Thayer [1982] developed an *Expectation-Maximization* (EM) algorithm [Dempster et al., 1977] for parameter estimation in the FA model, and recently a variant of the Wake-Sleep algorithm [Hinton et al., 1995b, Dayan et al., 1995], was proposed for FA [Neal and Dayan, 1997]. This algorithm, which shares some features with the EM algorithm of Rubin and Thayer [1982], is motivated by localized learning, in turn motivated by neuro-biological considerations.

The factor analysis model

Factor analysis represents an observed D-dimensional continuous variable, $\mathbf{t}$, as a linear function of an L-dimensional continuous latent variable and an independent Gaussian noise process,

$$\mathbf{t} = \mathbf{W}\mathbf{x} + \mathbf{e} + \boldsymbol{\mu}. \tag{2.5}$$

Here $\mathbf{W}$ is a D-by-L matrix defining the linear function, $\mathbf{e}$ is a D-dimensional vector representing the noise or individual variability associated with each of the D observed variables, and $\boldsymbol{\mu}$ is a D-dimensional vector representing the mean of the distribution of $\mathbf{t}$. To keep the notation simple we will assume, without any loss of generality, that the data sets we consider have zero mean, so that $\boldsymbol{\mu}$ can be omitted.

We also assume that $\mathbf{x}$ has a zero mean Gaussian distribution and, from the notational point of view, it also is convenient to assume that the latent variables are all independent and have unit variance,

$$p(\mathbf{x}) = \left(\frac{1}{2\pi}\right)^{L/2} \exp\left(-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{x}\right). \tag{2.6}$$

Finally also assuming that $\mathbf{x}$ and $\mathbf{e}$ are uncorrelated results in a conditional distribution over $\mathbf{t}$ which is also Gaussian,

$$p(\mathbf{t}|\mathbf{x}, \mathbf{W}, \boldsymbol{\Psi}) = \left(\prod_{d}^{D} 2\pi\Psi_{dd}\right)^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{t} - \mathbf{W}\mathbf{x})^{\mathrm{T}}\boldsymbol{\Psi}^{-1}(\mathbf{t} - \mathbf{W}\mathbf{x})\right), \tag{2.7}$$

where $\boldsymbol{\Psi}$ is a D-by-D diagonal matrix, with element Ψ_{dd} representing the variance of e_d, the individual variance of t_d.

From (2.5) and (2.6) follows that,

$$E[\mathbf{t}\mathbf{t}^{\mathrm{T}}] = \mathbf{W}\mathbf{W}^{\mathrm{T}} + \boldsymbol{\Psi}. \tag{2.8}$$

This manifests the fundamental assumption of many latent variable models, that the conditional distribution of the observed variables given the latent variables is independent, i.e. the dependence on the common latent variables explain all covariance between observed variables.

Equation (2.8) is the starting point for many of the algorithms proposed for parameter estimation in the FA model. For a given set of training data, $\{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_N\}$, we compute its sample covariance matrix,

$$\mathbf{S} = \frac{1}{N}\sum_{n}^{N} \mathbf{t}_n \mathbf{t}_n^{\mathrm{T}}, \tag{2.9}$$

where we have assumed that the data set has zero mean, and then seek parameters, $\mathbf{W}$ and $\mathbf{\Psi}$, that satisfy the equation

$$\mathbf{S} = \mathbf{W}\mathbf{W}^{\mathrm{T}} + \mathbf{\Psi}.$$

Here, however, we will review the EM algorithm of Rubin and Thayer [1982], to make a connection to the EM-algorithm derived for the non-linear generative model described in the next chapter.

An EM algorithm for factor analysis

The Expectation-Maximization (EM) algorithm [Dempster et al., 1977] is a general algorithm for maximum likelihood estimation in parameterized models from incomplete data. It works in two steps: in the E-step, it computes expected values of the missing parts of the data or the sufficient statistics thereof, given the observed data and a current value of the model parameters. In the M-step, it uses these expectations for the missing data to estimate new values for the parameters. It has been proved that, alternating between these two steps is guaranteed to increase the likelihood unless already at a (local) maximum [Dempster et al., 1977, Bishop, 1995].

In our case, we are given a set of observed data, $\{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_N\}$, but we are missing the corresponding $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, from which the data set is assumed to have been generated by (2.7); if these were known, estimation of $\mathbf{W}$ and $\mathbf{\Psi}$ would be straightforward. However, by Bayes' theorem, using (2.6) and (2.7), we can write the *posterior* distribution over $\mathbf{x}$ given a data point $\mathbf{t}_n$ as

$$p(\mathbf{x}|\mathbf{t}_n, \mathbf{W}, \mathbf{\Psi}) = (2\pi)^{-L/2}|\mathbf{M}|^{1/2} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Psi}^{-1}\mathbf{t}_n)^{\mathrm{T}}\mathbf{M}(\mathbf{x} - \mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Psi}^{-1}\mathbf{t}_n)\right), \quad (2.10)$$

which is an L-variate normal distribution with posterior covariance matrix

$$\mathbf{M}^{-1} = (\mathbf{I} + \mathbf{W}^{\mathrm{T}}\mathbf{\Psi}^{-1}\mathbf{W})^{-1}.$$

Now, assume for a moment that we know $\mathbf{x}_n$, $n = 1, \ldots, N$, and further that the data points in $\mathbf{T}$ have been drawn independently. We can then write the complete-data log-likelihood, using (2.6) and (2.7), as

$$\begin{aligned}\ell &= \sum_n^N \ln p(\mathbf{t}_n, \mathbf{x}_n) \\ &= -\frac{N}{2}\ln|\mathbf{\Psi}| - \frac{1}{2}\sum_n^N (\mathrm{tr}[\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}] + \mathrm{tr}[\mathbf{\Psi}^{-1}(\mathbf{t}\mathbf{t}^{\mathrm{T}} - 2\mathbf{W}\mathbf{x}_n\mathbf{t}^{\mathrm{T}} + \mathbf{W}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}\mathbf{W}^{\mathrm{T}})]) + \text{constant terms.}\end{aligned} \quad (2.11)$$

We do not know $\mathbf{x}_n$ and $\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}$, but using (2.10) we can compute their corresponding expectations (the E-step),

$$\langle\mathbf{x}_n\rangle = \mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Psi}^{-1}\mathbf{t}_n \text{ and} \quad (2.12)$$

$$\langle\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}\rangle = \mathbf{M}^{-1} + \langle\mathbf{x}_n\rangle\langle\mathbf{x}_n\rangle^{\mathrm{T}}, \quad (2.13)$$

resulting in an *expected* complete-data log-likelihood,

$$\begin{aligned}\langle\ell\rangle = -\frac{N}{2}\ln|\mathbf{\Psi}| - \frac{1}{2}\sum_n^N (\mathrm{tr}[\langle\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}\rangle] + \\ \mathrm{tr}[\mathbf{\Psi}^{-1}(\mathbf{t}\mathbf{t}^{\mathrm{T}} - 2\mathbf{W}\langle\mathbf{x}_n\rangle\mathbf{t}^{\mathrm{T}} + \mathbf{W}\langle\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}\rangle\mathbf{W}^{\mathrm{T}})]) + \text{constant terms.}\end{aligned} \quad (2.14)$$

The $\langle \mathbf{x}_n \rangle$, $n = 1, \ldots, N$, are referred to as *factor scores*, although there also other definitions of this term [Mardia et al., 1979].

We can maximise $\langle \ell \rangle$ with respect to $\mathbf{W}$ and $\mathbf{\Psi}$ (the M-step) by computing the corresponding derivatives, using results on matrix derivatives [Fukunaga, 1990, appendix A], and substituting for $\langle \mathbf{x}_n \rangle$ and $\langle \mathbf{x}_n \mathbf{x}_n^{\mathrm{T}} \rangle$, yielding the update formulae

$$\widetilde{\mathbf{W}} = \mathbf{S}\mathbf{\Psi}^{-1}\mathbf{W}(\mathbf{I} + \mathbf{W}^{\mathrm{T}}\mathbf{\Psi}^{-1}\mathbf{S}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}^{-1})^{-1} \text{ and} \tag{2.15}$$

$$\widetilde{\mathbf{\Psi}} = \mathrm{diag}(\mathbf{S} - \widetilde{\mathbf{W}}\mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Psi}^{-1}\mathbf{S}), \tag{2.16}$$

where $\mathbf{S}$ is defined in (2.9). Note that we use the old values, $\mathbf{W}$ and $\mathbf{\Psi}$, to compute $\langle \mathbf{x}_n \rangle$ and $\langle \mathbf{x}_n \mathbf{x}_n^{\mathrm{T}} \rangle$, while we use the updated value, $\widetilde{\mathbf{W}}$, in (2.14), when deriving (2.16).

If we study (2.15), we see that it can be re-written as follows:

$$\begin{aligned}
\widetilde{\mathbf{W}} &= \mathbf{S}\mathbf{\Psi}^{-1}\mathbf{W}(\mathbf{I} + \mathbf{W}^{\mathrm{T}}\mathbf{\Psi}^{-1}\mathbf{S}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}^{-1})^{-1} \\
&= \frac{1}{N}\mathbf{T}\mathbf{T}^{\mathrm{T}}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}^{-1}(\mathbf{M}^{-1} + \mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{\Psi}^{-1}\mathbf{S}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}^{-1})^{-1} \\
&= \frac{1}{N}\mathbf{T}\langle \mathbf{X} \rangle^{\mathrm{T}} \langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle^{-1},
\end{aligned} \tag{2.17}$$

where $\mathbf{T}$ is a $D \times N$ matrix containing the data points $\mathbf{t}_n$, $n = 1, \ldots, N$, as its columns, $\langle \mathbf{X} \rangle$ is the $L \times N$ matrix containing the corresponding posterior mean estimates from (2.12), and

$$\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle = \frac{1}{N} \sum_{n}^{N} \langle \mathbf{x}_n \mathbf{x}_n^{\mathrm{T}} \rangle,$$

with $\langle \mathbf{x}_n \mathbf{x}_n^{\mathrm{T}} \rangle$ defined in (2.13).

We can compare (2.17) with the least squares solution of the linear equations

$$\widetilde{\mathbf{W}} \langle \mathbf{X} \rangle = \mathbf{T}$$

for $\widetilde{\mathbf{W}}$,

$$\widetilde{\mathbf{W}} = \mathbf{T}\langle \mathbf{X} \rangle^{\mathrm{T}} (\langle \mathbf{X} \rangle \langle \mathbf{X} \rangle^{\mathrm{T}})^{-1}.$$

This solution ignores the covariance structure of the posterior distribution over $\mathbf{x}$ and is therefore incorrect, but it highlights the intuitive idea. We are alternating between estimating posterior mean points, for a given $\mathbf{W}$, and then estimating $\widetilde{\mathbf{W}}$ to map these back to the corresponding data points.

2.2.2 Principal components revisited

Recently, Tipping and Bishop [1997b] proposed a probabilistic formulation of PCA (PPCA), in the form of a FA model with an isotropic noise model, i.e. $\mathbf{\Psi} = \sigma^2\mathbf{I}$. They formulate an EM algorithm, similar to the one of reviewed above, and show that the maximum-likelihood estimate of $\mathbf{W}$ corresponds to (an arbitrary permutation of) the L principal eigenvectors of the covariance matrix, $\mathbf{S}$, scaled by their corresponding eigenvalues[3]

This brings PCA into the family of generative models, which in turn opens up a whole range of possibilities. In particular, Tipping and Bishop [1997a] show how to construct mixtures of principal component analyzers, which are fitted to data using a simple extension of the EM algorithm for basic probabilistic PCA.

[3] To be precise, they show that $\mathbf{W} = \mathbf{U}_L(\mathbf{\Lambda}_L - \sigma^2\mathbf{I})^{1/2}\mathbf{R}$, where $\mathbf{\Lambda}_L$ is a diagonal matrix containing the L largest eigenvalues of the sample covariance matrix $\mathbf{S}$, $\mathbf{U}_L$ contains the corresponding eigenvectors and $\mathbf{R}$ is an arbitrary $L \times L$ orthogonal rotation matrix.

2.2.3 Non-linear factor analysis

In some sense, all generative non-linear models presented in the following sections can be seen as variants of non-linear factor analysis. However, it is only the model by Etezadi-Amoli and McDonald [1983] which is explicitly proposed as a non-linear factor analysis model; it follows earlier models by McDonald [1967, 1962].

This model treats the observed variables as lower order polynomial functions of the latent variables,

$$\mathbf{T} = \mathbf{\Phi}(\mathbf{X})\mathbf{W} + \mathbf{E},$$

where $\mathbf{X}$ is an $N \times L$ matrix which rows corresponds to factor scores, $\mathbf{W}$ is an $L \times D$ matrix of adaptable weight parameters, $\mathbf{E}$ is an $N \times D$ matrix of residuals, and, for a two-factor model ($L = 2$) using second order polynomial,

$$\boldsymbol{\phi}_n = [x_{n1}, x_{n2}, x_{n1}^2, x_{n2}^2, x_{n1}x_{n2}],$$

where $\boldsymbol{\phi}_n$ denotes the nth row of $\mathbf{\Phi}(\mathbf{X})$; this can be extended to more factors and higher order polynomials.

As with the EM algorithm for FA presented in section 2.2.1, computing $\mathbf{W}$ and $\mathbf{E}$ would be straightforward if the elements of $\mathbf{X}$ where known. As this is not the case, McDonald [1979] also adopts an iterative scheme, which is a direct extension of a fitting method for linear factor analysis [McDonald, 1979], alternating between estimating $\mathbf{W}$ and $\mathbf{E}$ and adapting elements of $\mathbf{X}$, using gradient descent. However, Etezadi-Amoli and McDonald does not define any prior or posterior distribution over $\mathbf{x}$ at any stage so this is not generative model.

2.2.4 Principal curves revisited

There also exists a generative variant of the principal curve model [Tibshirani, 1992], which was motivated by the observation of Hastie and Stuetzle [1989] that the original principal curve model is not a generative model, in the sense that if

$$\mathbf{t} = \mathbf{f}(x) + \mathbf{e},$$

where $\mathbf{e}$ represents Gaussian noise and x is uniform on some closed interval, then $\mathbf{f}(\cdot)$ is generally *not* a principal curve for $p(\mathbf{t})$. The reason for this becomes clear from the illustration in figure 2.4 — the original principal curve should pass through middle of the data that projects *orthogonally* onto it, but when the generating curve is bent, the corresponding probability mass (shaded in the figure) is going be greater on the outside than the inside of the generative curve, so the resulting principal curve ends up with a wider radius than the generating curve.

Tibshirani defines a principal curve as a triplet $\langle p(x), p(\mathbf{t}|x), \mathbf{f}(x)\rangle$, where $\int p(\mathbf{t}|x)p(x)dx = p(\mathbf{t})$, and $\mathbf{f}(x)$ is a curve, parameterized over a closed interval, which satisfies the self-consistency property, (2.4). $p(\mathbf{t}|x)$ is defined $\mathcal{N}(\mathbf{f}(x), \boldsymbol{\sigma}(x))$[4], where $\boldsymbol{\sigma}(x)$ is a D-dimensional vector representing independent variances, whereas $p(x)$ is left unspecified. Given a set of i.i.d. data points, the resulting log-likelihood function becomes

$$\ell = \sum_n^N \log \int p(\mathbf{t}_n|x)p(x)\, dx. \tag{2.18}$$

[4]To be precise, Tibshirani defines $p(\mathbf{t}|x)$ to come from a parametric family, but only discusses the concrete case where it is Gaussian, which is also the case which is relevant in the context of this thesis.

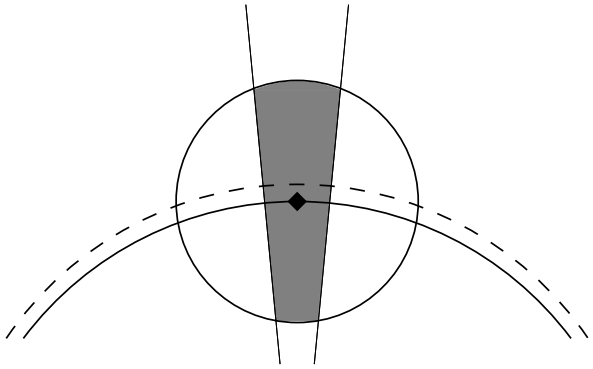


Figure 2.4: The solid curve represent a generating curve which is convolved with a Gaussian noise distribution, represented by the circle, which has its mean on the curve, represented by ⧫. The dashed curve represents the corresponding principal curve, which falls outside the generating curve.

Drawing on a theorem by Lindsay [1983], Tibshirani states that, with $\mathbf{f}(x)$ and $\sigma(x)$ kept fixed, the maximum likelihood estimate for the mixing density $p(x)$ is discrete with at most N points of support; we denote these points, $x_1, x_2, \ldots, x_K$, with $K \leq N$. Then, (2.18) becomes

$$\ell = \sum_{n}^{N} \log \sum_{k}^{K} p(\mathbf{t}_n|x)\pi_k, \tag{2.19}$$

where $\pi_k = p(x_k)$. This is just the log-likelihood function for a Gaussian mixture with centres $\mathbf{f}(x_k)$, variances $\sigma_{dk} = \sigma_d(x_k)$ and mixing coefficients π_k, $k = 1, \ldots, K$, which can be maximized using (e.g.) the EM algorithm. As noted by Tibshirani [1992], this function has its maxima, equal to infinity, when there is a one to one matching between centres and data point and the variances are driven to zero; moreover, any curve passing through all data points will reach this maximum, and hence there is a uniqueness problem.

To force the centres of the Gaussian mixture to follow a smooth 1-dimensional curve, Tibshirani introduces a cubic spline smoother, which plays the role of a regularizer of the log-likelihood function,

$$\ell = \sum_{n}^{N} \log \sum_{k}^{K} p(\mathbf{t}_n|x)\pi_k + \lambda \int (\mathbf{f}''(x_k))^2 \, dx. \tag{2.20}$$

The resulting model is a *regularized* Gaussian mixture, where λ controls the degree of regularization, which can be trained using the EM algorithm, with a modified M-step. This will also have to involve finding new positions for the support points of the discrete mixing distribution, $x_1, x_2, \ldots, x_K$, for which Tibshirani uses a one-step Newton-Raphson procedure. This will not maximise, but increase the log-likelihood, and so it corresponds to a generalised EM (GEM) algorithm [Dempster et al., 1977].

2.2.5 Density networks

'Density Networks' [MacKay and Gibbs, 1997, MacKay, 1995] is the label attached to a fairly general framework, proposed to extend the applicability of feed-forward neural networks [Bishop, 1995], such as the multi-layer perceptron, to the domain of *unconditional* density modelling. These have already proved highly successful for conditional density modelling, e.g. in pattern classifiers and non-linear regression models. As such, they have been trained using methods known as 'supervised learning', where data is split into inputs and targets; for each input datum there is a corresponding target that the model should try to match. In unconditional density modelling, there is no such division of data

— the model tries to model the joint distribution of all data variables and must by itself discover any structure in the data that can aid the modelling. The associated training procedures are therefore know as *unsupervised learning*.

MacKay [1995] merges the theory of feed-forward neural networks with that of latent variable models by regarding the inputs of the network, $\mathbf{x}$, as latent variables, for which he prescribes a prior distribution $p(\mathbf{x})$. This results in a corresponding distribution over the outputs of the network, the nature of which depends on the network. With a simple linear network with linear outputs, a Gaussian distribution over the latent variables and an axis-aligned Gaussian noise model in the target space, this simply becomes a factor analysis model. If the outputs are fed through a soft-max function, in which case the resulting variables can be interpreted as conditional probabilities of class membership, we have obtained what is known as a *latent trait* model [Lazarsfeld and Henry, 1968]. It captures the idea of a sparse distribution in a categorical space dependent on a continuous underlying variable, which is manifested in correlations between the categorical variables. MacKay [1995, 1996] shows how such models can be used for discovering structure in protein data. Using more complex networks in these models, with non-linear units between inputs and outputs, will allow more complex structures to be discovered.

To fit these models to data, MacKay [1995] employs a conjugate-gradient optimization routine [see e.g. Press et al., 1992], where the gradient is computed by averaging over the posterior distribution over the latent space given the data.

2.2.6 The Elastic net

The elastic net algorithm was originally proposed by Durbin and Willshaw [1987] as a heuristic method for finding good approximate solutions to the travelling salesman problem (TSP). The TSP consists of finding a tour of minimal length that makes a single visit to each of the cities in a given set; it is known to be NP-complete [see e.g. Papadimitriou and Steiglitz, 1982]. The elastic net algorithm takes a geometrical approach, starting with a set of points, $\mathbf{Y} = \{\mathbf{y}_k\}, k = 1, \ldots, K$, distributed evenly on a loop, initially shaped as a circle and centered on the mean of the set of points, $\mathbf{T} = \mathbf{t}_n, n = 1, \ldots, N$, representing the cities under consideration. The points on the loop are then moved in steps towards 'cities' to which they are close, while trying to minimize the distances to their nearest neighbours on the circle, as illustrated in figure 2.5. Gradually, the trade-off between these two forces is shifted so that closeness of some point on the loop to each point representing a city becomes dominating.

Durbin et al. [1989] reformulated the algorithm and showed that it can be interpreted as a *maximum a posteriori* (MAP) estimate over the distribution of possible tours, specified by a prior favouring short tours,

$$p(\mathbf{Y}) = \prod_{k}^{K} \exp\left\{-\frac{\gamma}{V}\|\mathbf{y}_k - \mathbf{y}_{k+1}\|^2\right\}, \tag{2.21}$$

where the indices of the $\mathbf{y}$-points are counted modulo K, and a likelihood factor computed from the data,

$$p(\mathbf{T}|\mathbf{Y}) = \prod_{n}^{N} \frac{1}{K} \sum_{k}^{K} \left(\frac{1}{2\pi V^2}\right)^{D/2} \exp\left\{-\frac{1}{2V^2}\|\mathbf{t}_n - \mathbf{y}_k\|^2\right\}. \tag{2.22}$$

The prior, (2.21), is a K-dimensional, correlated Gaussian, encouraging the points in $\mathbf{Y}$ to follow a locally 1-D structure. The likelihood factor, (2.22), is a product of N independent distributions, each consisting of a mixture of K Gaussians, with centres $\mathbf{y}_k$ and common variance V. γ, in (2.21), controls the trade-off between the prior and likelihood factors.

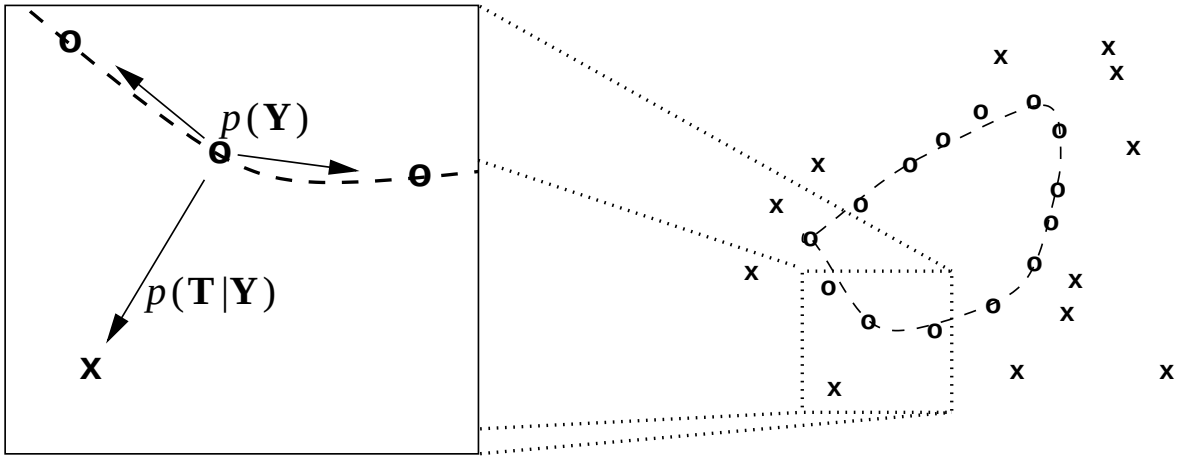


Figure 2.5: The right half of the figure shows a schematic illustration of an elastic net model, drawn as a dashed loop with $\circ$ representing the points $\mathbf{y}_k$, and data points ('cities', $\mathbf{t}_n$) plotted as $\times$. To the left is a blow-up, where arrows represent the forces acting on the $\mathbf{y}_k$-points, which arises from the prior $p(\mathbf{Y})$ (2.21) and likelihood $p(\mathbf{T}|\mathbf{Y})$ (2.22).

Utsugi [1997, 1996] discusses a generalization of the elastic net model, with a prior that imposes a 2-D structure on the Gaussian mixture model, and relates it to the self-organizing map, discussed below. This model has some similarities with the model proposed in the next chapter, and will be further discussed there (section 3.4.2).

2.3 Other models

2.3.1 The Self-organizing map

The Self-Organizing Map (SOM) [Kohonen, 1995] is a neural network architecture for unsupervised learning which shares many features with the models discussed so far, despite having rather different motivation. Since it was proposed by Kohonen [1982], it has had considerable success in a wide range of applications, and has been the subject of significant research efforts. Nevertheless, the SOM is still lacking a sound theoretical foundation and is generally motivated by heuristic arguments.

The inspiration for the SOM came from observations of self-organization taking place in the sensory cortex of the human brain. Bilateral connections between nearby neurons encourage spatial ordering of sensory input to be reflected in the 2-D spatial ordering of neurons — neighbouring neurons will typically be activated by similar stimuli. A typical SOM model is depicted in figure 2.6; it consists of a set of nodes (sometimes referred to as 'neurons') arranged in a regular lattice in a (typically) 2-D space; associated with each node, k, is a so called *reference vector*, $\mathbf{w}_k$, which lives in a D-dimensional space. Given a D-dimensional set of data, the SOM is trained using the following algorithm:

1. Initialize the reference vectors, $\mathbf{w}_k^0$, e.g. setting them equal to random samples drawn from the data.

2. For each iteration, i, select a data point, $\mathbf{t}_n$, either at random or cycling through the data points, and find the node with the closest reference vector; that is, find node k_n such that

$$k_n = \underset{k}{\operatorname{argmin}} \, \|\mathbf{t}_n - \mathbf{w}_k^{(i-1)}\|^2.$$

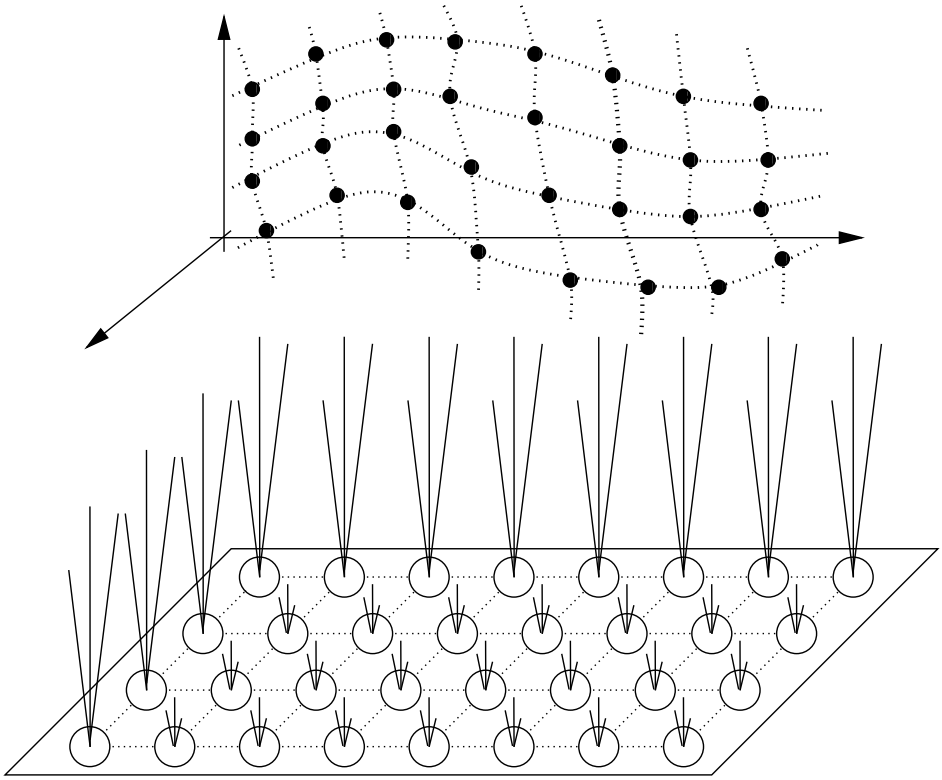


Figure 2.6: A schematic illustration of the self-organizing map — the lower part of the figure shows the nodes, drawn as circles, arranged in a rectangular 2-D lattice. Each node is mapped to a corresponding reference vector in the data space, illustrated as black discs in the upper part of the figure. As indicated, the ordering of the reference vectors should reflect the ordering of the nodes.

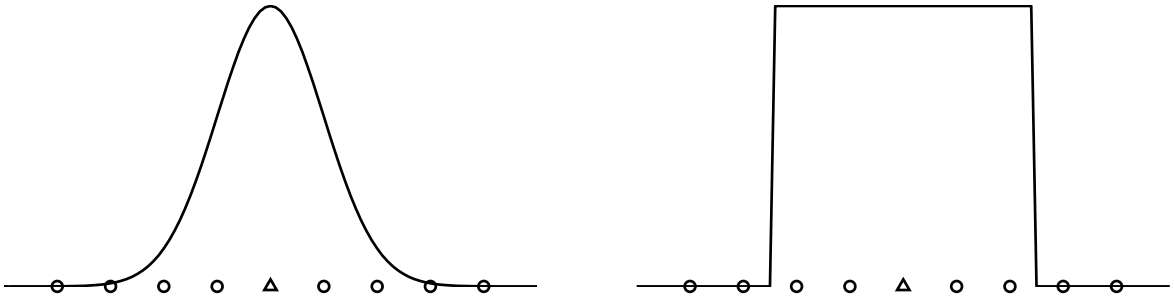


Figure 2.7: The left and right plots show a Gaussian and 'top hat' neighbourhood function, respectively, for a 1-D SOM. Nodes are plotted as $\circ$, except for the node on which the neighbourhood function is centred (k_n), which is plotted as $\triangle$.

3. Update the reference vectors so that

$$\mathbf{w}_k^{(i)} = \mathbf{w}_k^{(i-1)} + \eta^{(i)} h^{(i)}(k, k_n)(\mathbf{t}_n - \mathbf{w}_{k_n}^{(i-1)})$$

   where $\eta^{(i)}$ is a learning rate and $h^{(i)}(\cdot)$ is the *neighbourhood function*.

4. Repeat step 2 and 3 while decreasing the value of η and the width of the neighbourhood function.

The neighbourhood function $h^{(i)}(k, k_n)$ typically take values between 0 and 1, is unimodal, symmetric and centred on k_n; common choices are the unnormalized Gaussian and the 'top-hat' function[5], illustrated in figure 2.7. The intended effect of the neighbourhood function is encourage the reference vectors of nodes which are near each other on the map, to be near each other in the data space. As the width of the neighbourhood function gradually decreases, so does the influence nodes have on their neighbours.

There is no theoretical framework for how to choose starting values and decrementing schedules for $\eta^{(i)}$ and $h^{(i)}(\cdot)$, but there are simple rules of thumb which usually give reasonable results [Kohonen et al., 1995].

The Batch SOM

Most of the training algorithms discussed so far are *batch* algorithms, meaning that each update of the model parameters is based on all data points, whereas the original version of the SOM is a so called *online* algorithm, which makes a separate update for each data point, taken one at a time. There is also a batch version of the SOM algorithm (BSOM):

Initialize the reference vectors, $\mathbf{w}_k$ e.g. using random samples from the data.
repeat
  for each data point, $\mathbf{t}_n$, **do**
    Find node k_n such that $k_n = \underset{k}{\operatorname{argmin}} \|\mathbf{t}_n - \mathbf{w}_k\|^2$.
  end for
  Update all the reference vectors using:

$$\mathbf{w}_k^{(i)} = \sum_n^N \frac{h^{(i)}(k, k_n)\mathbf{t}_n}{\sum_{n'}^N h^{(i)}(k, k_{n'})}. \tag{2.23}$$

until convergence

[5] Also called the 'bubble' neighbourhood function.

Note that the learning rate parameter η is no longer present. Obviously, the **for**-loop and the subsequent update in the batch algorithm will be computationally more intensive than their online counterparts, steps 2 and 3, but this is usually compensated by a much faster convergence, counted in number of iterations.

Problems with the SOM

Although the SOM has been subject of a considerable amount of research and applied to a wide range of tasks, there are still a number of problems that remain unresolved [Kohonen, 1995].

1. The SOM does not define a density model in the data space. Attempts has been made to formalize the relationship between the distribution of reference vectors and the distribution of the data, but has only succeeded under very restricted conditions [Ritter and Schulten, 1986, 1988].
2. The training algorithm does not optimize an objective function — in fact, it has been proved [Erwin et al., 1992] that such an objective function cannot exist.
3. There is no general guarantee the training algorithm will converge.
4. There is no theoretical framework based on which appropriate values for the model parameters can be chosen, e.g. initial value for the learning rate and width of the neighbourhood functions, and subsequent rate of decrease and shrinkage, respectively.
5. It is not obvious how SOM models should be compared to other SOM models or to models with different architectures.
6. The mapping from the topographic space to the data space in the original SOM is only defined at the locations of the nodes.

Probabilistic versions of the SOM

Points 2–5 above, all stem from the first point and would largely be resolved in a probabilistic setting. This has inspired the search for re-formulations of the SOM within the framework of probability theory and statistics. Indeed, the model presented in the next chapter has been proposed as a principled alternative to the SOM [Bishop et al., 1997b, 1996b], and a related model based on the elastic net has also been proposed along those lines [Utsugi, 1997, 1996]. A rather different approach is taken by Luttrell [1994], who derives the SOM as a special case in a more general framework based on folded Markov chains.

Here we review a latent variable based approximation to the SOM, developed for modelling radar range profile data [Luttrell, 1995]. The data is assumed to follow a low-dimensional manifold (1-D for radar range profile data), so Luttrell devises the following probabilistic model:

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y}(x))p(x)\,dx, \tag{2.24}$$

where $p(\mathbf{t}|\mathbf{y}(x))$ is assumed to be Gaussian with mean $\mathbf{y}(x)$ and the prior distribution over the latent variable, $p(x)$, is assumed to be uniform over a finite interval X.

The model is fitted using maximum-likelihood, by gradient ascent. The gradient of the log-likelihood function involves the term, $p(x|\mathbf{t})$, which, using Bayes' theorem, can be written as

$$p(x|\mathbf{t}) = \frac{p(\mathbf{t}|\mathbf{y}(x))p(x)}{p(\mathbf{t})}. \tag{2.25}$$

Luttrell approximates (2.25) by the formula

$$p(x|\mathbf{t}) \approx \pi(x - x(\mathbf{t})),$$

where $x(\mathbf{t})$ is the point on X minimizing $\|\mathbf{t} - \mathbf{y}(x)\|$ (c.f. the projection index of principal curves) and $\pi(\cdot)$ is chosen based on prior knowledge of the data.

He then suggests the following training algorithm:

1. Select a random data point, $\mathbf{t}_n$, and find the value $x(\mathbf{t}_n)$ that minimizes $\|\mathbf{t}_n - \mathbf{y}(x)\|$
2. Adjust $\mathbf{y}(x)$ so that

$$\mathbf{y}(x) \to \mathbf{y}(x) + \eta\pi(x - x(\mathbf{t}))(\mathbf{t}_n - \mathbf{y}(x)).$$

3. Repeat step 1 and 2 till convergence.

Since the necessary calculations cannot be done analytically, x is quantized into a discrete set of non-overlapping 'bins' over the interval X, each with its own $\mathbf{y}(x)$. Following this 'discretization', steps 1 and 2 above will correspond approximately to steps 2 and 3 of the SOM algorithm, with $\pi(x - x(\mathbf{t}))$ playing the role of the neighbourhood function. Thus, the training algorithm of the SOM can be seen as an approximation to maximum-likelihood training of a latent variable model.

2.3.2 Multidimensional scaling

Given an $N \times N$ matrix of 'distances', $\mathbf{D}$, between N points, multidimensional scaling (MDS) [Mardia et al., 1979, Ripley, 1996] gives a corresponding set of N points, $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, in an L-dimensional space, such that the distances between points in $\mathbf{X}$ reflect those given in $\mathbf{D}$. The 'distances' need not be Euclidean distances, but can be more general, e.g. distance measures for categorical variables or subjective measures of similarity, in which case they are often called *dissimilarities*. These dissimilarities are the only information about the data that is required, so indeed the data does not even need to have an explicit form. However, in the context that we are interested in, where the data has an explicit representation as a set of points in $\Re^D$, for which the Euclidean distance is the obvious dissimilarity measure, it can be shown that MDS corresponds to PCA. More precisely, the set of points found by MDS, $\mathbf{X}$, corresponds (up to scaling and rotation) to the projection of the data on its first L principal components. In this form, MDS is known as *principal coordinate analysis*.

The Sammon mapping

The Sammon mapping [Sammon, 1969] represents a particular form of MDS [Ripley, 1996] — the basic idea is the same, but the Sammon mapping pays more attention to smaller distances, thereby achieving a varying resolution in the new representation of the data. Regions with a dense population of data points, between which distances are small, will be 'magnified' in the new representation. To formalize, given a set of 'distances' between N data points[6], the Sammon mapping tries to find the set of points $\{\mathbf{x}_n\}$, $n = 1, \ldots, N$, in $\Re^L$ that minimizes

$$\sum_{j<i}^{N} = \frac{\left(d_{ij}^{\mathbf{t}} - d_{ij}^{\mathbf{x}}\right)^2}{d_{ij}^{\mathbf{t}}}, \tag{2.26}$$

[6] We assume that these distances are symmetric and that the distance from a point to itself is zero.

where $d_{ij}^{\mathbf{t}}$ denotes the distance between $\mathbf{t}_i$ and $\mathbf{t}_j$ and $d_{ij}^{\mathbf{x}}$ is the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. This is a non-linear problem so iterative, numerical optimizations schemes must be used.

The name 'mapping' is somewhat misleading, since the Sammon mapping does not provide any mapping that can be utilized to find a point in the model space corresponding to a new point in the data space. This has led to the use of parametric neural network models to learn the mapping from the data space to the low-dimensional space, using (2.26) as an error function [Lowe and Tipping, 1996, Kraaijveld et al., 1995].

2.4 Discussion

This chapter has reviewed a number of models intended for capturing low-dimensional structure in data living in high-dimensional spaces, or at least provide a low-dimensional representation of this data. A striking fact is that three of the 'non-generative' models that we have considered — PCA and the original versions of the principal curve and the SOM — has been re-interpreted or reformulated for the purpose of bringing them into the family of generative models. The attraction of this type of model stems from the fact it fits into the much wider framework of probability theory and statistics. They can therefore directly make use well-founded theory for fitting models to data, combining models, treatment of incomplete data, etc.

In the next chapter we propose a generative latent variable model for modelling non-linear, continuous probability distributions with low intrinsic dimensionality, embedded in high-dimensional spaces. Although similar models have been discussed in this chapter, these differ in scope or suffer practical or theoretical limitations.

- In the generative principal curve model, the number of latent points, K, depends on the number of points in the data set used for training, N; in practice, it will almost always be the case that $K = N$. This is likely to cause computational difficulties when tackling larger data sets. Moreover, it is difficult to see how this model could be extended to online learning.

- The density network model has been proposed in fairly general terms, but in practice it has so far only been applied to categorical data.

- The original elastic net model was proposed for finding good, heuristic solutions to the travelling salesman problem. This is reflected in the structure of the model and that there are typically many more mixture components than there are data points ('cities'). The data is assumed to be free of noise and so, in a successfully trained model, there is a mixture component positioned at each data point. A generalised elastic net model will be discussed in the next chapter.

- In the latent variable model proposed by Luttrell [1995] to give a probabilistic interpretation of the self-organizing map, the posterior distribution over the latent variables is approximated using a function which is based on prior knowledge of the data, but such prior knowledge may not always be available. As we will see, this approximation is in fact not necessary.

Chapter 3

The Generative Topographic Mapping

This chapter presents the *generative topographic mapping* (GTM) — a novel non-linear latent variable model — along with examples that illustrate how the GTM works and its potential applications. There is also a discussion on the relationship between GTM and some of models presented in the previous chapter, in particular the self-organizing map.

The underlying idea of the GTM is the same as that of factor analysis and probabilistic PCA — we are seeking an 'explanation' to the behaviour of a number of observed variables (data variables), in terms of a smaller number of hidden, or latent, variables. In contrast to FA and PPCA, the GTM allows for a *non-linear* relationship between latent and observed variables.

3.1 The GTM Model

The GTM defines a non-linear, parametric mapping $\mathbf{y}(\mathbf{x}, \mathbf{W})$ from an L-dimensional latent space ($\mathbf{x} \in \Re^L$) to a D-dimensional data space ($\mathbf{y} \in \Re^D$) where normally $L < D$. $\mathbf{y}(\mathbf{x}, \mathbf{W})$ could e.g. be a multi-layer perceptron [Bishop, 1995], in which case $\mathbf{W}$ would denote its weights and biases; as we shall see later, by making a careful choice of how we implement $\mathbf{y}(\mathbf{x}, \mathbf{W})$, significant savings can be made in terms of computation. For now, we just define it to be continuous and differentiable. $\mathbf{y}(\mathbf{x}, \mathbf{W})$ maps every point in the latent space to a point in the data space. Since the latent space is L-dimensional, these points will be confined to an L-dimensional manifold non-linearly embedded in the D-dimensional data space. Figure 1.2 showed a schematic illustration where a 2-dimensional latent space was mapped to a 3-dimensional data space.

If we define a probability distribution over the latent space, $p(\mathbf{x})$, this will induce a corresponding probability distribution in the data space. Strictly confined to the L-dimensional manifold, this distribution would be singular, so we convolve it with an isotropic Gaussian noise distribution, given by

$$\begin{aligned} p(\mathbf{t}|\mathbf{x}, \mathbf{W}, \beta) &= \mathcal{N}(\mathbf{y}(\mathbf{x}, \mathbf{W}), \beta) \\ &= \left(\frac{\beta}{2\pi}\right)^{-D/2} \exp\left\{-\frac{\beta}{2}\sum_{d}^{D}(t_d - y_d(\mathbf{x}, \mathbf{W}))^2\right\} \end{aligned} \tag{3.1}$$

where $\mathbf{t}$ is a point in the data space and β^{-1} denotes the noise variance. This can be thought of as smearing out the manifold, giving it a bit of volume, and corresponds to the residual variance of

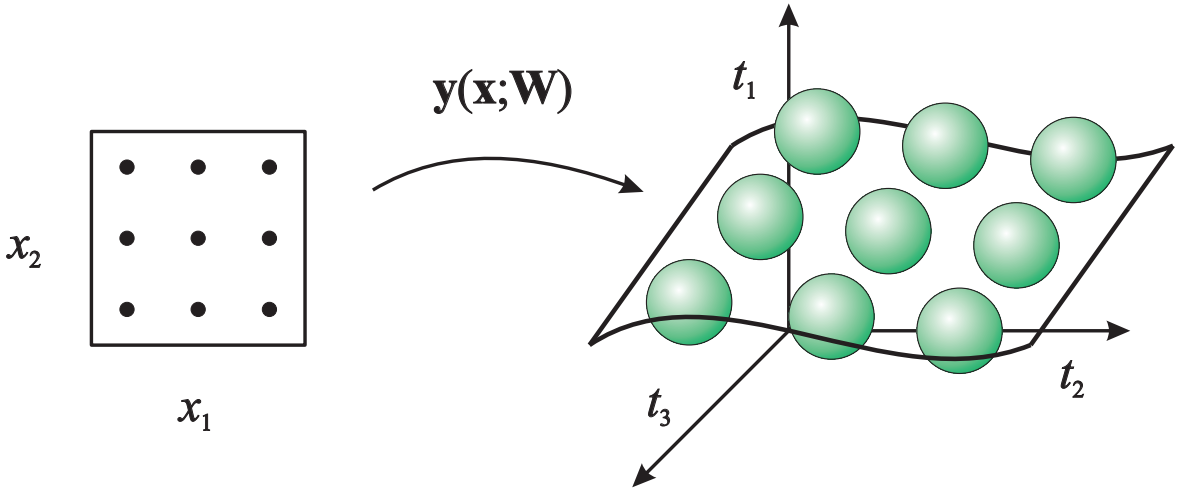


Figure 3.1: The basic idea of the GTM — points on a regular grid in the low-dimensional latent space (left) are mapped, using a parameterised, non-linear mapping $\mathbf{y}(\mathbf{x}, \mathbf{W})$, to corresponding centres of Gaussians (right). These centres will lie in the low-dimensional manifold, defined by the mapping $\mathbf{y}(\mathbf{x}, \mathbf{W})$, embedded in the (potentially) high-dimensional data space.

the PPCA model (section 2.2.2) — it allows for some variance in the observed variables that is not explained by the latent variables.

By integrating out the latent variable, we get the probability distribution in the data space expressed as a function of the parameters β and $\mathbf{W}$,

$$p(\mathbf{t}|\mathbf{W}, \beta) = \int p(\mathbf{t}|\mathbf{x}, \mathbf{W}, \beta)\, p(\mathbf{x})\, d\mathbf{x}. \tag{3.2}$$

This integral is generally not analytically tractable. However, by choosing $p(\mathbf{x})$ to have a particular form, a set of K equally weighted delta functions on a regular grid,

$$p(\mathbf{x}) = \frac{1}{K} \sum_{k}^{K} \delta(\mathbf{x} - \mathbf{x}_k), \tag{3.3}$$

the integral in (3.2) turns into a sum,

$$p(\mathbf{t}|\mathbf{W}, \beta) = \frac{1}{K} \sum_{k}^{K} p(\mathbf{t}|\mathbf{x}_k, \mathbf{W}, \beta). \tag{3.4}$$

An alternative approach, used by Bishop et al. [1996a] and MacKay [1995], is to approximate $p(\mathbf{x})$ with a Monte Carlo sample. If $p(\mathbf{x})$ is taken to be uniform over a finite interval, this becomes similar to (3.4).

Now we have a model where each delta function centre (we will from now on refer to these as latent points) maps to the centre of a Gaussian which lies in the manifold embedded in the data space, as illustrated in figure 3.1. Note that as long as $\mathbf{y}(\mathbf{x}, \mathbf{W})$ is continuous, the ordering of the latent points will be reflected in the ordering of the centres of Gaussians in the data space. What we have is a *constrained* mixture of Gaussians [Hinton et al., 1992, Williams, 1994], since the centres of the mixture components can not move independently of each other, but all depend on the mapping $\mathbf{y}(\mathbf{x}, \mathbf{W})$. Moreover, all components of the mixture share the same variance, β^{-1}, and the mixing coefficients are all fixed to $1/K$.

Given a finite set of i.i.d. data points, $\{\mathbf{t}_1, \ldots, \mathbf{t}_N\}$, we can write down the likelihood function for

this model,

$$\mathcal{L} = \prod_n^N p(\mathbf{t}|\mathbf{W}, \beta) = \prod_n^N \left[\frac{1}{K} \sum_k^K p(\mathbf{t}_n|\mathbf{x}_k, \mathbf{W}, \beta) \right], \tag{3.5}$$

and maximise it with respect to $\mathbf{W}$ and β. However, it is normally more convenient to work with the log-likelihood function,

$$\ell = \sum_n^N \ln \left(\frac{1}{K} \sum_k^K p(\mathbf{t}_n|\mathbf{x}_k, \mathbf{W}, \beta) \right). \tag{3.6}$$

We could employ any standard non-linear optimization technique [see e.g. Press et al., 1992] for the maximization, but having noted that we are working with a mixture of Gaussians, we may instead use the EM algorithm [Dempster et al., 1977, Bishop, 1995]. In the last chapter, section 2.2.1, we saw how an EM-algorithm could be used to fit a factor analysis model to a data set, where the key step was to compute the expectations of sufficient statistics of the latent variables, the values of which were missing. When fitting a mixture of Gaussians, which is maybe the most common example of the application of the EM-algorithm [see e.g. Bishop, 1995], the problem would be easily solved if we knew which data point was generated by which mixture component; unfortunately, this is usually not the case and so we treat these 'labels' as missing variables.

3.2 An EM algorithm for the GTM

Given some initial values for $\mathbf{W}$ and β, the E-step for the GTM is the same as for a general Gaussian mixture model, computing the *responsibilities*,

$$r_{kn} = p(\mathbf{x}_k|\mathbf{t}_n, \mathbf{W}, \beta) = \frac{p(\mathbf{t}_n|\mathbf{x}_k, \mathbf{W}, \beta)p(\mathbf{x}_k)}{\sum_{k'} p(\mathbf{t}_n|\mathbf{x}_{k'}, \mathbf{W}, \beta)p(\mathbf{x}_{k'})}, \tag{3.7}$$

assumed by the kth component of the Gaussian mixture for the nth data point, for each possible pair of k and n. r_{kn} corresponds to the *posterior* probability that the nth data point was generated by the kth component. As the prior probabilities, $p(\mathbf{x}_k)$, were defined to be fixed and equal $(1/K)$ in (3.3), these will cancel in (3.7). Note that, since the mixture components correspond to points in the latent space, the distribution of responsibilities over mixture components correspond to a distribution over the latent space, forming a connection to the EM-algorithm for FA. In the M-step, these responsibilities will act as weights in the update equations for $\mathbf{W}$ and β. In essence, we will try to move each component of the mixture towards data points for which it is most responsible.

So far, we have not specified the form for $\mathbf{y}(\mathbf{x}, \mathbf{W})$, but only stated that it could be any parametric, non-linear model. For the GTM, we normally choose a generalised linear regression model, where $\mathbf{y}$ is a linear combination of a set of fixed basis functions,

$$y_d(\mathbf{x}, \mathbf{W}) = \sum_m^M \phi_m(\mathbf{x}) w_{md}, \tag{3.8}$$

We could consider a wide range of basis function, but for the rest of this thesis, we will use a combination of

- M_{NL} non-linear basis functions, in the form of non-normalised, Gaussian basis functions,
- L linear basis functions, for capturing linear trends in the data, and

- one fixed basis function, that allows the corresponding weights to act as biases.

Thus, we get

$$\phi_m(\mathbf{x}) = \begin{cases} \exp\left\{-\frac{\|\mathbf{x}-\boldsymbol{\mu}_m\|^2}{2\sigma^2}\right\} & \text{if } m \leq M_{\mathrm{NL}}, \\ x^l & \text{if } m = M_{\mathrm{NL}} + l,\ l = 1, \ldots, L \\ 1 & \text{if } m = M_{\mathrm{NL}} + L + 1 = M, \end{cases} \tag{3.9}$$

where $\boldsymbol{\mu}_m$, $m = 1, \ldots, M_{\mathrm{NL}}$, denotes the centres of the Gaussian basis functions and σ their common width, and x^l denotes the lth element of $\mathbf{x}$. Note that, throughout the rest of the this thesis, the GTM models used in experiments are understood to have linear and bias basis functions, and these will not be explicitly mentioned. It will be convenient to write (3.8) in matrix form as

$$\mathbf{Y} = \boldsymbol{\Phi}\mathbf{W}, \tag{3.10}$$

where $\mathbf{Y}$ is a $K \times D$ matrix of mixture component centres, $\boldsymbol{\Phi}$ is a $K \times M$ matrix with elements $\Phi_{km} = \phi_m(\mathbf{x}_k)$, and $\mathbf{W}$ is a $M \times D$ matrix containing the weight and bias parameters.

We now derive the M-step for this model as follows: using (3.1), (3.7) and (3.8), we can calculate the derivatives of (3.6) with respect to w_{md}, yielding

$$\frac{\partial \ell}{w_{md}} = \sum_{n,k}^{N,K} r_{kn}\beta \left(\sum_{m'}^{M} \phi_{m'}(\mathbf{x}_k) w_{m'd} - t_{nd} \right) \phi_m(\mathbf{x}_k), \tag{3.11}$$

where r_{kn} are the responsibilities computed in the preceding E-step, and setting these derivatives to zero we obtain an update formula for $\mathbf{W}$. A detailed derivation is found in appendix A. Similarly, calculating the derivatives of (3.6) with respect to β and setting these to zero, we obtain

$$\frac{1}{\beta} = \frac{1}{ND} \sum_{n}^{N} \sum_{k}^{K} r_{kn} \|\mathbf{y}(\mathbf{x}_k, \widetilde{\mathbf{W}}) - \mathbf{t}_n\|^2. \tag{3.12}$$

Here, $\widetilde{\mathbf{W}}$ corresponds to the updated weights, which means that we must first maximise with respect to the weights, then with respect to β. The update formula for β is the same as for general Gaussian mixtures and has an intuitive meaning. We set β^{-1}, which is the common variance of the Gaussian mixture, to the average weighted distance between mixture components and data points, where the weights are given by the responsibilities.

Using (3.10), the M-step for $\mathbf{W}$ can be written on matrix form as

$$\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{G}\boldsymbol{\Phi}\mathbf{W} = \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}\mathbf{T} \tag{3.13}$$

where $\mathbf{T}$ is the $N \times D$ matrix containing the data points, $\mathbf{R}$ is the $K \times N$ responsibility matrix with elements defined in (3.7), and $\mathbf{G}$ is an $K \times K$ diagonal matrix with entries

$$g_{kk} = \sum_{n}^{N} r_{kn}. \tag{3.14}$$

(3.13) can be seen as a form of generalised least squares [Mardia et al., 1979]. To draw the parallel with the M-step for the factor analysis model in (2.17), we are setting $\mathbf{W}$ to map the weighted, non-linear representation of the latent variables, $\mathbf{G}\boldsymbol{\Phi}$, to the targets formed by the weighted combination of data points, $\mathbf{R}\mathbf{T}$.

We can now also see the advantages of having chosen a generalized linear regression model, as this part of the M-step is reduced to a matrix inversion and a few matrix multiplications. A different

model, where the log-likelihood depended non-quadratically on the adjustable parameters, would have required non-linear, iterative maximization, at each iteration computing a new log-likelihood, which is generally the most costly part of the algorithm[1]. Note that, since $\mathbf{\Phi}^{\mathrm{T}}\mathbf{G}\mathbf{\Phi}$ is symmetric and often positive definite, we can utilize fast Cholesky decomposition for the matrix inversion, with the option of resorting to singular value decomposition (SVD) [Press et al., 1992, Strang, 1988], if the matrix proves to be singular. There are two possible ways this can happen: $\mathbf{G}$ may contain one or more zeros along its diagonal, which means that the corresponding mixture components take no responsibility at all. This is very unlikely to happen as long there are significantly less mixture components than data points. The second possible cause is rank deficiency in $\mathbf{\Phi}$, which may occur if we choose the basis functions very broad or very narrow, or use more basis functions than latent points. Normally, there will be no difficulty avoiding such choices of basis functions and the rank of $\mathbf{\Phi}$ can be checked prior to fitting the GTM to data.

In addition, we could impose a degree of weight regularization, leading to the equation

$$(\mathbf{\Phi}^{\mathrm{T}}\mathbf{G}\mathbf{\Phi} + \lambda\mathbf{I})\mathbf{W} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{R}\mathbf{T} \tag{3.15}$$

where λ is the regularization parameter and $\mathbf{I}$ is an identity matrix of the same dimensions as $\mathbf{\Phi}^{\mathrm{T}}\mathbf{G}\mathbf{\Phi}$. This correspond to specifying an isotropic Gaussian prior distribution over $\mathbf{W}$,

$$p(\mathbf{W}) = \left(\frac{\alpha}{2\pi}\right)^{W/2} \exp\left(-\frac{\alpha}{2}\|\mathbf{W}\|^2\right), \tag{3.16}$$

with zero mean and variance α^{-1}, where W denotes the total number of elements in $\mathbf{W}$. From (3.11) and (3.16), it follows that $\lambda = \alpha/\beta$. Apart from ensuring a fast matrix inversion, the use of weight regularization gives us one handle on the model complexity through the real valued parameter α. The issue of model complexity and parameter selection will be further discussed in chapter 5.

3.2.1 Initialization

The only remaining issue is to choose appropriate initial values for $\mathbf{W}$ and β. For $\mathbf{W}$, one possibility is to use random samples drawn from a Gaussian distribution, $\mathcal{N}(\mathbf{0}, \boldsymbol{\varsigma})$, where $\boldsymbol{\varsigma}$ is chosen so that the expected variance over $\mathbf{y}$ equals the variance of the training data. An alternative, which is often better, is to initialize the weights so that the L latent variables map to the L-dimensional hyper-plane spanned by the L first principal components of the data set we are trying to model. A PCA initialization only requires the weight of the linear basis functions, so weights of the non-linear basis functions can be set to zero, or alternatively, to very small random values, resulting in a 'semi-random' initialization. Whether we use random or PCA-based initialization, it is reasonable to initialize the weight vector corresponding to the bias basis function so as to match the mean of the training data. For β, our choice to some extent depends on how we choose $\mathbf{W}$. If $\mathbf{W}$ is initialized randomly β is set to the reciprocal of the average squared distance between the centres of the resulting Gaussian mixture and the points in our data set, which correspond to the update formula in (3.12) with all responsibilities being equal. If, on the other hand, $\mathbf{W}$ is initialized using PCA, β is set so that its inverse (the variance in the data space) equals the larger of

- the length of the $(L+1)$th principal component, i.e. the largest variance orthogonal to the L-dimensional hyper-plane to which the Gaussian mixture is initially mapped,
- half the average minimal distance between the mixture components.

[1] In such a case it might be better to only a partial M-step, increasing, but not necessarily maximising the likelihood, corresponding to a generalised EM (GEM) algorithm [Dempster et al., 1977]

This is motivated by the idea that the initial β should be small enough to explain the variance orthogonal to, as well as the variance within, the initial manifold.

3.2.2 Summary of the GTM algorithm

We now summarize the sequence of steps for constructing a GTM model:

Generate the grid of latent points $\{\mathbf{x}_k\}$, $k = 1, \ldots, K$.
Generate the grid of basis function centres $\{\boldsymbol{\mu}_m\}$, $m = 1, \ldots, M$.
Select the basis function width σ.
Compute the matrix of basis function activations, $\mathbf{\Phi}$, from (3.9).
Initialize $\mathbf{W}$, randomly or using PCA.
Initialize β.
If desired, select a value for α.
Compute $\mathbf{\Delta}$, $\Delta_{kn} = \|\mathbf{t}_n - \mathbf{\Phi}_k \mathbf{W}\|^2$.
repeat
  Compute $\mathbf{R}$ from (3.7) using $\mathbf{\Delta}$ and β.
  Compute $\mathbf{G}$ from (3.14) using $\mathbf{R}$.
  } $E - step$
  $\mathbf{W} = (\mathbf{\Phi}^{\mathrm{T}}\mathbf{G}\mathbf{\Phi} + \lambda\mathbf{I})^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{R}\mathbf{T}$, where λ may be zero.
  Compute $\mathbf{\Delta}$, $\Delta_{kn} = \|\mathbf{t}_n - \mathbf{\Phi}_k \mathbf{W}\|^2$.
  Update β according to (3.12), using $\mathbf{R}$ and $\mathbf{\Delta}$.
  } $M - step$
until convergence

Note how the squared distances required to update β in the M-step gets 're-used' when calculating the responsibilities in the following E-step. Next, we look at an example of how this algorithm works.

Example 3.1 (Curved line in 2-D) *Figure 3.2 shows how a GTM with a 1-dimensional latent variable 'learns' to model a data set which is intrinsically 1-dimensional but has been non-linearly embedded in a 2-dimensional data space. The data set was generated by picking 59 equidistant points in the interval* $[0.15, 3.05]$ *as the x-coordinates. The y-coordinates were then computed as the function* $y = x + 1.25\sin(2x)$*. Finally, spherical Gaussian noise with standard deviation* 0.1 *was added to the data. As can be seen in figure 3.2, this results in a data set with a distribution which is more dense around the bends of the curve and sparser towards the ends, as expected. The initial configuration for the GTM was found using principal components.*

3.3 Visualization

An important potential application for the GTM is visualization. To see how this works, note that a GTM, for which we have found suitable parameter values $\mathbf{W}^*$ and β^*, by (3.1) and (3.3), defines a probability distribution in the data space conditioned on the latent variable, $p(\mathbf{t}|\mathbf{x}_k)$, $k = 1, \ldots, K$. We can therefore use Bayes' theorem, in conjunction with the prior distribution over latent variable, $p(\mathbf{x})$, given in (3.3), to compute the corresponding posterior distribution in latent space for any given point in data space, $\mathbf{t}$, as

$$p(\mathbf{x}_k|\mathbf{t}) = \frac{p(\mathbf{t}|\mathbf{x}_k, \mathbf{W}^*, \beta^*)p(\mathbf{x}_k)}{\sum_{k'} p(\mathbf{t}_n|\mathbf{x}_{k'}, \mathbf{W}^*, \beta^*)p(\mathbf{x}_{k'})}.$$

As can be seen, this is exactly the calculation of responsibilities in (3.7), where again the $p(\mathbf{x}_k)$ cancel.

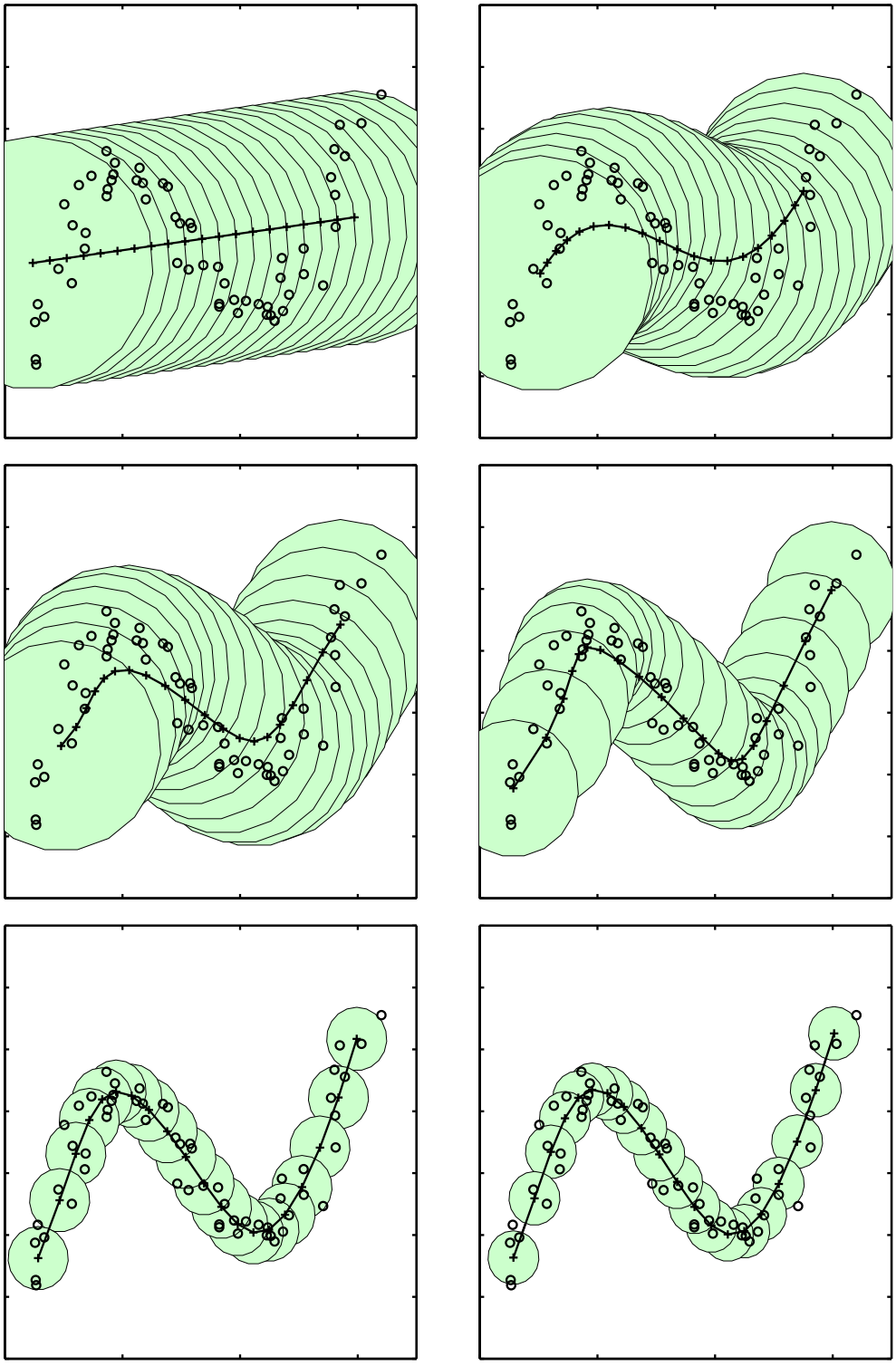


Figure 3.2: The GTM learning process — the plots show the density model in data space at iteration 0 (the initial configuration), 1, 2, 4, 8 and 15. The data points are plotted as ∘ while the centres of the Gaussian mixture are plotted as '+'. The centres are joined by a line according to their ordering in the latent space. The discs surrounding each '+'-sign represent two standard deviations' width of the noise model ($2\sqrt{\beta^{-1}}$). Note that the final density model reflects the distribution of the training data.

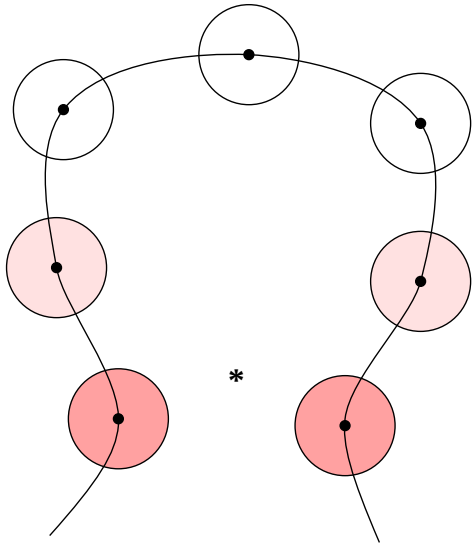


Figure 3.3: The figure shows a schematic illustration of a 1-D GTM in a 2-D data space, together with a data point, plotted as *. The manifold of the GTM is bent, so that the two mixture components that are closest to the data point are not close to each other on the manifold. This result in bi-modal distribution of responsibilities over the mixture components, illustrated in the figure with a shading of mixture components 'proportional' to the responsibility they take.

Provided that the latent space has no more than two, or possibly three, dimensions, we can plot $p(\mathbf{x}_k|\mathbf{t})$ against $\mathbf{x}_k$. If we want to visualize whole sets of data, we must resort to less rich descriptions. Two possibilities are, for each data point $\mathbf{t}_n$, to plot

- the mode of the posterior distribution in latent space,

$$\mathbf{x}_n^{\text{mode}} = \underset{\mathbf{x}_k}{\text{argmax}}\ p(\mathbf{x}_k|\mathbf{t}_n),$$

  which we call the *posterior-mode* projection, or

- the mean of the posterior distribution in latent space,

$$\mathbf{x}_n^{\text{mean}} = \sum_k^K \mathbf{x}_k p(\mathbf{x}_k|\mathbf{t}_n),$$

  consequently called the *posterior-mean* projection.

Whatever we choose, we must bear in mind that summarizing descriptors, such as the mode and the mean, can give misleading results, e.g. in case the posterior distribution is multi-modal. A schematic illustration of how such a situation may arise is given in figure 3.3, for a 1-D GTM. In fact, plotting both the mean and the mode and comparing them can give an indication of multi-modality. Our second example demonstrates how the GTM can be used for visualization of data.

Example 3.2 (3-phase pipe flow data) *In this example we use synthetically generated data, simulating non-intrusive measurements by gamma-densitometry, from a pipeline transporting a mixture of gas, oil and water [Bishop and James, 1993]. The fractions of gas, water and oil vary, and the flow in the pipe takes one of three possible configurations.*

The construction for data collection is illustrated in figure 3.4. Six pairs of γ-beams, where the two beams in each pair have different wave length, are sent through the pipe, and from measurements of

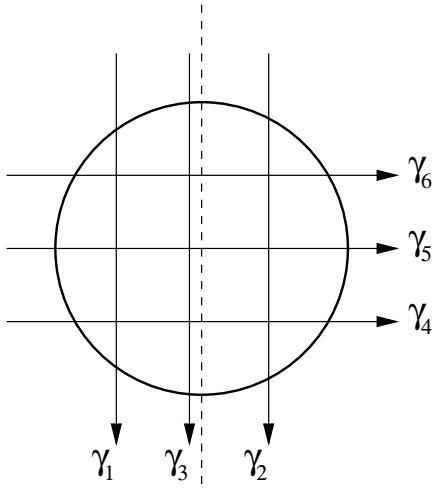


Figure 3.4: A cross-section of the pipe, showing the location of the γ-beams used for collecting the measurements in the 3-phase data. Note that the vertical beams are displaced relative to the centre of the pipe, because all configurations considered are left-right symmetrical. (see figure 3.5).

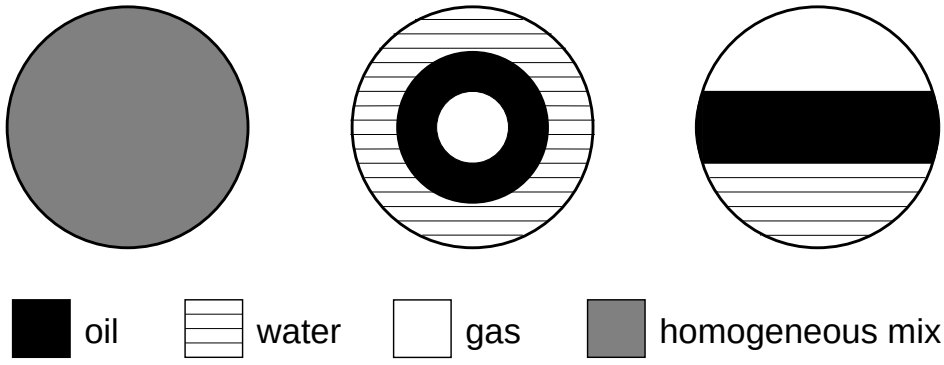


Figure 3.5: A cross-section view of the three different configurations of flow in the pipe, showing, left to right, homogeneous, annular and stratified flow.

their attenuation, the path lengths through water and oil can be computed. With six pairs of beams, this data set is twelve-dimensional. However, for any given flow-configuration, there are only two degrees of freedom in the data: the fractions of oil and water (the fraction of gas being redundant, as the three fractions must sum to one). Hence, even if this data lives in a twelve-dimensional space, it is really confined to a two-dimensional subspace.

The existence of multi-phase flow configurations complicate matters somewhat. The three different configurations of flow are illustrated in figure 3.5. For the homogeneous flow, which is simply a homogeneous mix of oil, water and gas, only one γ-beam would be required to determine the fractions of oil and water — data points collected for this configuration all lives (approximately) in a two-dimensional plane in the data space, and hence the measurements from the different γ-beams provide the same information. For the annular configuration, the relationship between the measurements and the fractions of water and oil is no longer linear, but all data points taken from this flow configuration still lives on a single curved manifold. This not the case with the stratified (or laminar) configuration — as the three fractions change, the vertical γ-beams change from passing only through oil to passing only through water (say). This cause discontinuities, and consequently data points from this configuration

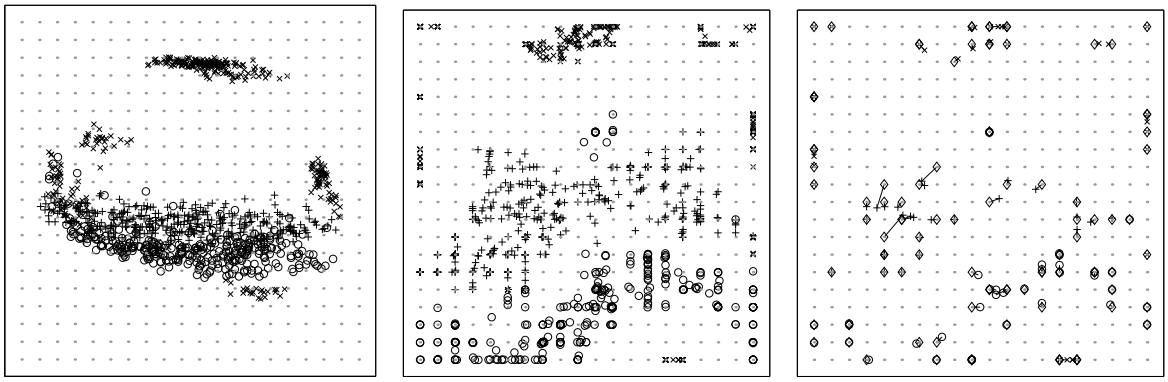


Figure 3.6: Shown, left to right, are the posterior-mean projection of the data in the latent space of the PCA-initialised GTM, prior to training, the corresponding plot after having trained the GTM, and, rightmost, pairs or posterior-mean and -mode projections for the trained GTM, joined by lines, for 100 randomly drawn data points. The three different types of flow are plotted as + (homogeneous), ∘ (annular) and × (stratified). In all the plots, latent points are plotted as shaded ·. In left plot, ⋄ represent posterior mode points, with the class label given by the connected posterior mean point.

are spread over a number of separate two-dimensional manifolds. The three classes join at the three points, corresponding to pure flows of oil, water or gas.

The data generating model includes a noise process, modelling errors in the measurements arising from photon statistics. In a real setting, the noise level would be governed by the time spent to collect each data point, called the integration time. The data set discussed here was generated so as to correspond to an integration time of 10 seconds.

A GTM model was fitted to a set containing samples from all three classes. It had a 20-by-20 square grid of latent points in two-dimensional space. It utilized, apart from bias and linear basis functions, 81 Gaussian basis functions with their centres located on a 9-by-9 square grid in the latent space. Both grids were centred on the origin in the latent space. The basis functions had a common width of 2 times the shortest distance between two neighbouring basis functions. The model was initialized using PCA and trained for 40 iterations of the training algorithm, imposing a Gaussian prior on the weights with inverse variance $\alpha = 0.1$.

The left panel in figure 3.6 shows the posterior-mean projection of the training data in the latent space with the initial configuration found using PCA; the middle panel shows the corresponding plot after training. The separation of the three different classes has increased; in particular, the data points belonging to the laminar class has been distributed over a number of distinct clusters. The right panel shows a plot of posterior-mean and -mode for a few of the data points, with the mean and mode corresponding to the same data point connected by a line. The rather large distances between mean and mode in some cases suggest that the corresponding distributions may be multi-modal, or at least skewed. This is not completely unexpected, as the GTM is modelling a rather complex distribution spread over a number of separated two-dimensional manifolds, some of which are curved, using a single 'elastic' manifold.

3.4 Relationship to other models

In the previous chapter we discussed a number of models which all have a similar aim to the GTM. In this section we discuss the relationship between the GTM and some of these models, giving special

attention to the relationship to the Self-Organizing Map, which has a long-standing position in the area of unsupervised neural networks.

3.4.1 The Self-organizing map

Since the GTM defines a density model in the the data space, many of the problems associated with the SOM, which were discussed in section 2.3.1, are automatically solved [Bishop et al., 1996b, 1997b, 1998b].

- The GTM is trained by optimizing an objective function, namely the log-likelihood function in (3.6).
- The EM-algorithm is guaranteed to converge to a (local) maxima of the log-likelihood function [Dempster et al., 1977, Bishop, 1995]. By appealing to the Robbins-Monro theorem [Robbins and Monro, 1951, Fukunaga, 1990], sequential maximization schemes could also be guaranteed to converge, or we could consider using an online EM-algorithm [Titterington et al., 1985].
- We can invoke the machinery of Bayesian statistics to derive methods for treating the parameters of the model, as will be described in chapter 5.
- The likelihood provides a measure based on which a GTM model can be compared to other generative models.

Another important feature of the GTM is that, if the mapping from the latent space to the data space is taken to be smooth, the topographic ordering in the latent space will be preserved on the manifold in the data space[2]. This is a direct consequence of the fact the GTM defines a continuous manifold in the data space, which is not the case with the original SOM model. To this end, Ritter [1993] suggested the *parameterized* SOM (PSOM) model, where a parametric 'surface' is constructed that passes through the reference vectors of a fitted SOM model, by associating a basis function with each node-reference vector pair. A more elegant solution, however, is the kernel-smoothed interpretation of the BSOM, by Mulier and Cherkassky [1995], which is discussed further in the section on kernel smoothing below. However, both these models still suffer many of the problems of the original version of the SOM, stemming from the fact that they do not define generative models.

We now investigate the relationship between the GTM and the SOM in a little bit more detail.

Soft vs. hard assignment

If we study the training algorithms for the GTM and the SOM, we can discover both similarities and differences. An important dividing line is the way the two models handle the assignment of data points to mixture components or reference vectors. The SOM assigns each data point to a single reference vector, corresponding to 'the winning node', whereas the GTM distributes the responsibility for a data point over a number of mixture components. This difference is analogous to that between the *K-means* algorithm [Linde et al., 1980] and the EM-algorithm for a conventional, K-component Gaussian mixture. A K-means model represents a data set using K mean points, $\boldsymbol{\mu}_k$, which are fitted to given a data set of N data points, $\{\mathbf{t}_n\}$, using the following algorithm[3]:

initialize $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, e.g. using K randomly drawn points from the data set

repeat

[2] Note that this does *not* imply that the GTM is guaranteed to reveal any topographic ordering present in the data.

[3] Note here the similarities with the training algorithm for the batch version of the SOM, discussed in section 2.3.1.

for each data point, $\mathbf{t}_n$, **do**

Find k such that $k = \underset{k'}{\operatorname{argmin}} \|\mathbf{t}_n - \boldsymbol{\mu}_{k'}\|^2$ and assign $\mathbf{t}_n$ to $\boldsymbol{\mu}_k$ — $\mathbf{t}_n \in \mathcal{T}_k$.

end for

re-estimate the the mean points so that

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{\mathbf{t} \in \mathcal{T}_k} \mathbf{t}, \text{ where } N_k \text{ in the number of elements in } \mathcal{T}_k$$

until no data point has its assignment changed.

The assignment of data points to mean points can be seen as a special case of the E-step in (3.7), where $\beta \to \infty$ and all responsibility is assigned to a single mixture component. The re-estimation of the mean points correspond exactly to the M-step for updating a mixture of Gaussians — each mean point or mixture component is set equal to a weighted combination of the data points assigned to it. The difference lies in the weights of this combination. In the K-means case, each data point assigned to $\boldsymbol{\mu}_k$ gets weight $1/N_k$ in the update formula, while all other data points get weight zero. For the Gaussian mixture, the weights are given by the responsibilities, which are typically greater than zero.

The GTM and the SOM differ in exactly the same way, in terms of assignment of data points to latent points or nodes. In terms of the update, however, both the GTM and the SOM differ from the simple weighted averaging used by K-means and Gaussian mixtures, as well as from each other. As already pointed out, the GTM defines a *constrained* Gaussian mixture in the data space, so even though it has the same weighted average of data points used for the general Gaussian mixture as target for its update, it can only try to fit this target as well as possible, while maintaining its overall smooth, low-dimensional structure. The SOM uses the neighbourhood function to allow nodes to influence each other in the update of their corresponding reference vectors — in effect, each node is incorporating data points assigned to other nodes in the weighted average update of its reference vector. The weights assigned to data points of other nodes depends on the distances between the nodes in the *latent* space, and will usually differ from the responsibilities used to calculate the update target for the GTM, which are based on the distances between mixture components and data points in the *data* space. From this perspective, the use of the neighbourhood function in the SOM model can be seen as a way of trying to smooth or distribute the hard assignments of data points to reference vectors. In the GTM, there is no need for such arbitrary smoothing, since it uses soft assignments — responsibilities — calculated under a probabilistic model. Further insights can be gained by studying how the distribution of responsibilities evolve during training of a GTM model. Figure 3.7 shows grey-scale plots of the responsibility distribution over the latent space during different stages of training, for a particular data point. The responsibility distribution starts off being rather wide, to then gradually narrow with training. The effect of this process is similar to that achieved by shrinking the neighbourhood in the SOM model. The important difference is that in the GTM, this results as an *automatic* consequence of the gradually improved fit of the model to the data, whereas the shrinking of the neighbourhood in the SOM model has to be done 'by hand', by the user.

When comparing the GTM and the SOM, it is difficult to describe the precise effects of these different strategies of assignment, as they depend on many factors, only some of which are under the control of the user. Figure 3.8 shows the same plot as in figure 3.7, but for a different data point. From initially having the same characteristics as the distribution in figure 3.7, instead of getting narrower the single mode here splits into two. Whereas in the unimodal case, the hard assignment of the SOM combined with neighbourhood smoothing could possibly be regarded as a reasonable approximation, it is clearly inappropriate in this bi-modal case.

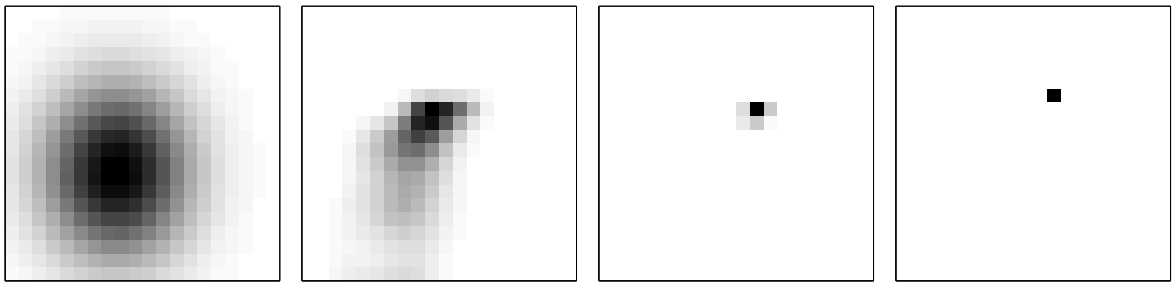


Figure 3.7: The four plots show the distribution of responsibilities over the latent space for a particular data point from the pipe-flow data set described in example 3.2, at the initial configuration, found by PCA, and then after 2, 4 and 40 iterations of training.

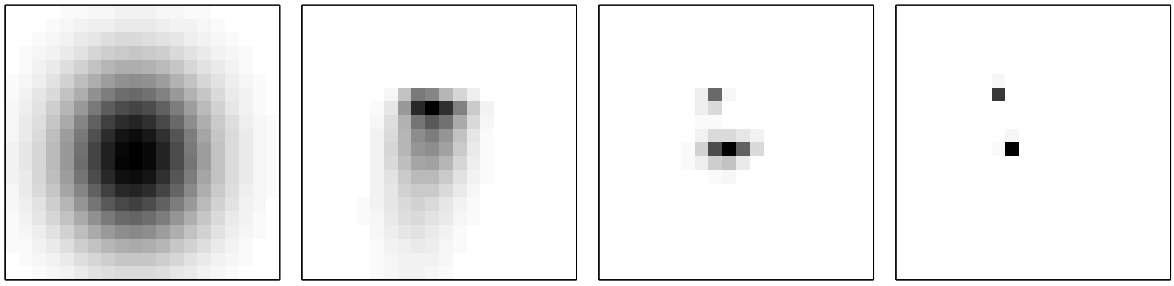


Figure 3.8: The four plots correspond to those shown in figure 3.7, showing the distribution of responsibilities over the latent space at different stages of training, but for a different data point, where the distribution eventually splits over two modes.

Kernel regression

A different framework in which the relationship between the GTM and the SOM can be analyzed is that of kernel regression. As noted by Haan and Egecioglu [1991], the update formula for the batch version of the SOM (BSOM),

$$\mathbf{w}_k = \sum_n^N \frac{h(k, k_n)\mathbf{t}_n}{\sum_{n'}^N h(k, k_{n'})}, \tag{2.23}$$

(where we here have dropped the time-step index (i)) can be written as

$$\mathbf{w}_k = \frac{\sum_{k'}^K N_{k'} h(k, k')\, \mathbf{m}_{k'}}{\sum_j^K N_j h(k, j)},$$

where

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{\mathbf{t} \in \mathcal{T}_k} \mathbf{t}, \tag{3.17}$$

is the mean of the set of data points assigned to reference vector k, denoted by $\mathcal{T}_k$, and N_k denotes the number of data points in $\mathcal{T}_k$.

Mulier and Cherkassky [1995] used this to show that, at any given iteration of the training algorithm, the BSOM model can be expressed using a kernel regression formula

$$\mathbf{y}(\mathbf{x}) = \sum_k^K F(\mathbf{x}, \mathbf{x}_k)\mathbf{m}_k \tag{3.18}$$

with $\mathbf{m}_k$ defined as in (3.17) and

$$F(\mathbf{x}, \mathbf{x}_k) = \frac{N_k h(\mathbf{x}, \mathbf{x}_k)}{\sum_j N_j h(\mathbf{x}, \mathbf{x}_j)}, \tag{3.19}$$

is the kernel function of node $\mathbf{x}_k$, and we have made the neighbourhood functions dependency on location in the topographic space explicit.

Also the GTM model can, at any given iteration of the training algorithm, be written on the form in (3.18), with the kernel functions

$$F(\mathbf{x}, \mathbf{x}_k) = \boldsymbol{\phi}(\mathbf{x})(\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{G}\boldsymbol{\Phi})^{-1}\boldsymbol{\phi}(\mathbf{x}_k)^{\mathrm{T}} g_{kk}, \tag{3.20}$$

where $\boldsymbol{\phi}(\mathbf{x})$ is a $1 \times M$ vector with elements $\phi_m(\mathbf{x})$, $\mathbf{G}$ and g_{kk} are defined in (3.14), and

$$\mathbf{m}_k = g_{kk}^{-1}\mathbf{R}_k\mathbf{T}, \tag{3.21}$$

where $\mathbf{R}_k$ is the kth row of $\mathbf{R}$ — the responsibility matrix with elements defined in (3.7). Figure 3.9 shows examples of this kernel, approximately centred in the latent space, during different stages of training. Note that both kernel functions — (3.19) and (3.20) — sum to one. For (3.19) this follows directly from the formula, while for (3.20), it is easy to see that $\mathbf{F}\boldsymbol{\Phi} = \boldsymbol{\Phi}$, where $\mathbf{F} = [F(\mathbf{x}_i, \mathbf{x}_j)]$, $i, j = 1, \ldots, K$; the result then follows from the fact the Mth column of $\boldsymbol{\Phi}$, which corresponds to the bias basis function, contains only ones (1's).

Formulae (3.17) and (3.21) again reflects the difference between the hard assignment of the SOM and the soft assignment of the GTM. If we study the kernel functions in (3.19) and (3.20), we see that the SOM kernel will gradually get narrower during training, as a consequence of the shrinking neighbourhood function. The GTM kernel, on the other hand, varies only with $\mathbf{G}$, and typically retains its width during training, although peaks and troughs tend to become more pronounced. This is illustrated in figure 3.9, and is a consequence of another important difference between the SOM and the GTM. For the GTM, the stiffness of the manifold, which primarily depends upon the width of the non-linear basis functions, does not change during training. For the SOM model, the 'manifold' starts off being rather stiff[4], to then gradually become more flexible as the neighbourhood function shrinks. This gradual softening, which is essential for the learning in the SOM, unfortunately makes the relationship between the user controlled parameters (e.g. the initial and final width of the neighbourhood, rate of shrinking, etc.) and *a priori* expectations about the resulting model rather obscure. In the GTM, user controlled parameters are de-coupled from the learning process, and their impact on the final model is therefore easier to understand. Figure 3.10 shows examples of 2-D manifolds embedded in a 3-D space, generated from a GTM with a 2-D latent space, by randomly sampling weight parameters from the prior (3.16). The three plots correspond to increasing values of σ; α will only affect the overall scale of the manifold, although we could consider allowing greater variance for the weights of the linear basis functions, which would then consequently result in more 'linear' manifolds.

Computational considerations

Although the rapid development of computer technology has to some extent altered our perception of computational complexity, this issue cannot be ignored. To simplify the comparison, we here only consider the batch version of the SOM (BSOM).

[4] As have already been mentioned, the original SOM model does not define a continuous manifold in the data space, but thinking of the reference vectors as spanning an elastic manifold helps the understanding.

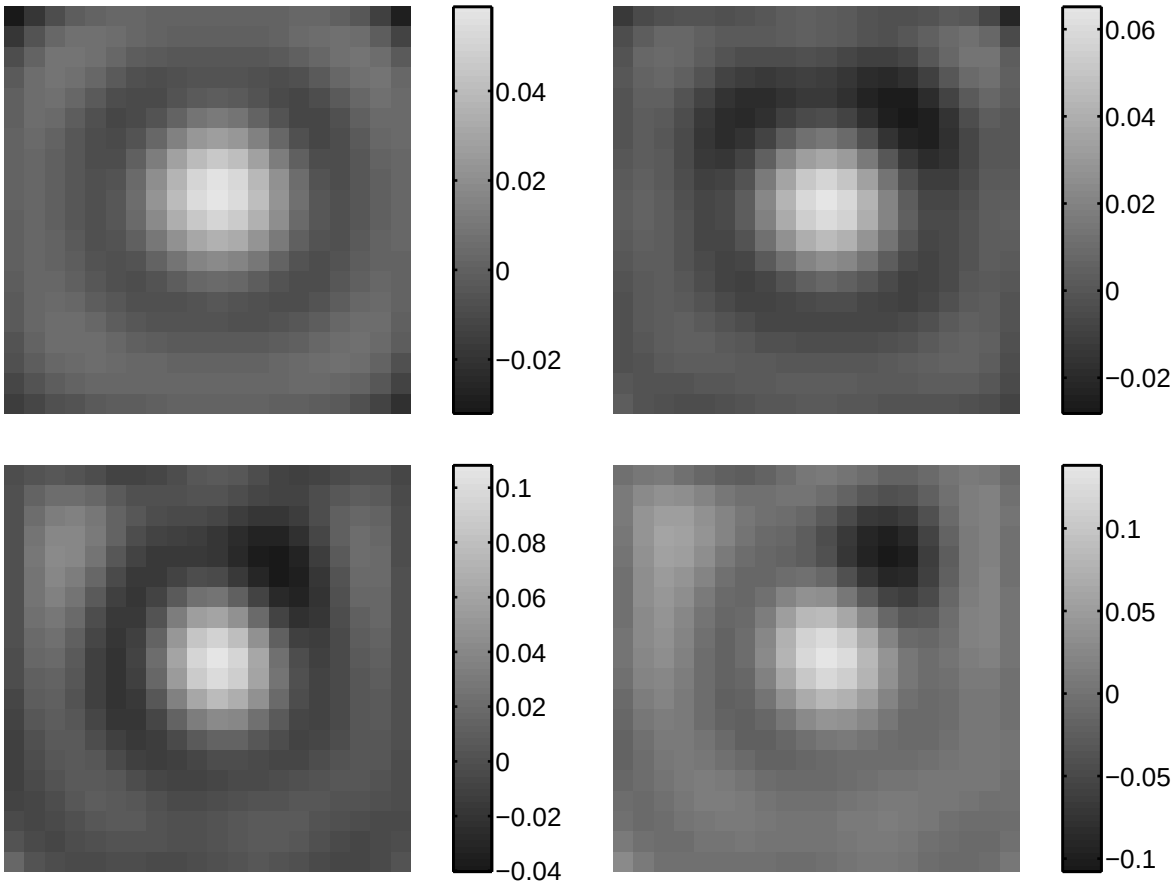


Figure 3.9: The four plots show (right–left, top–down) the kernel (3.20), evaluated over the latent grid after 0 (PCA initialisation), 2, 10 and 40 iterations of training, using the data set from example 3.2, with the centre of the kernel located approximately at the centre of the latent space.

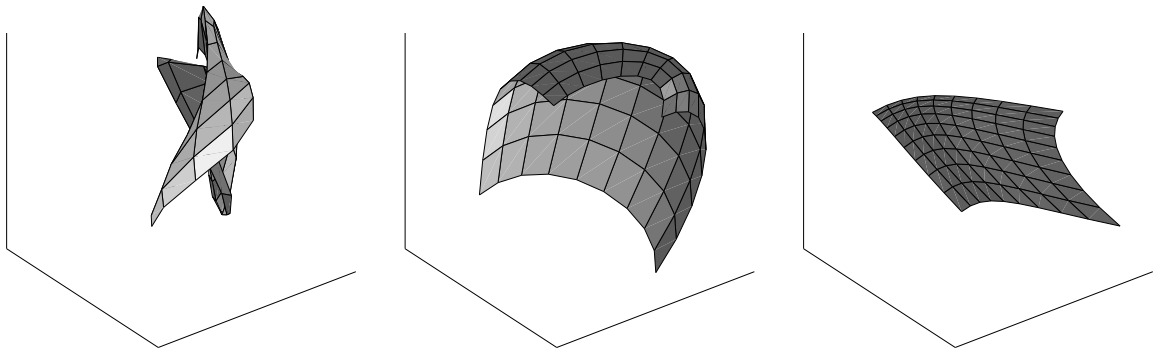


Figure 3.10: Sample manifolds of increasing stiffness. These manifolds were generated by first selecting the relative width for the non-linear basis functions, σ — 0.5, 1.0 and 2.0 for the left, middle and right panel, respectively — and then draw weight parameters randomly from the prior (3.16).

If we study the steps for computing the statistics necessary to update the parameters (winning nodes or responsibilities), we see that the distance calculation between data points and mixture components of reference vectors, respectively, is identical in both training algorithms. On top of that, the GTM has the additional cost of computing the responsibilities from these distances, but as the the dimensionality of the data space increases, the proportional cost of this extra step decreases.

When updating the parameters, the GTM requires a matrix inversion of an $M \times M$ matrix, where M is the number of basis functions, followed by a set of matrix multiplications. The matrix inversion scales as $\mathcal{O}(M^3)$, while the matrix multiplications scales as $\mathcal{O}(KND)$[5]. The update of the SOM depends on the form of the neighbourhood function. If it is continuous on the latent space, then every node will potentially be influenced by all other nodes and so the update will require $\mathcal{O}(K^2ND)$ operations. Every time the width neighbourhood changes, determining the cross-influence between nodes will require another $\mathcal{O}(K^2)$ operations. If, on the other hand, the top-hat neighbourhood function is used, each node will only influence nodes which are within the width of the neighbourhood function, which can result in dramatic savings, especially when the neighbourhood is small. However, updates using the top-hat neighbourhood function is typically much less smooth than those obtained when using e.g. a Gaussian neighbourhood function.

Assuming that the BSOM is using a continuous neighbourhood function, the cost ratio for the respective update calculations will largely depend on the ratio between K and M. Normally, the number of basis functions in the GTM will be much smaller than the number of latent points. When applied to the data used in example 3.2, a BSOM model with a corresponding grid of 20×20 nodes and a Gaussian neighbourhood function converged in roughly the same time as the GTM used in the example. However, using the top-hat neighbourhood function, the same BSOM model converged to a 'comparable' solution (as judged by visually inspecting the resulting winning node plot for the data) in less than a third of that time. An additional factor that must be considered is the number of trials required to find suitable parameter values, and to which extent such trials can be run and assessed without human supervision. As described in chapter 5, there are principled ways in which this can be done automatically for the GTM.

Various techniques could be used in both models to speed up the computations. One potential such technique for the GTM, which retains all its desirable properties, is discussed in section 6.6.

To summarize, with a top-hat neighbourhood function, a BSOM model will normally converge more quickly than the corresponding GTM model (i.e. the number of latent points equals the number of nodes). However, using a Gaussian neighbourhood function with the BSOM model, which typically gives a smoother convergence, the difference in speed of convergence will depend on the ratio between K and M. In practice, we normally chose $M < K/2$ for the GTM, in which case the convergence rates are similar.

3.4.2 A Generalised elastic net model

Recently, a generalization of the elastic net model was proposed by Utsugi [1997, 1996] as a probabilistic formulation of the SOM — a model which is closely related to the GTM. Recall that the elastic net model, as proposed by Durbin et al. [1989] (see section 2.2.6), is a Gaussian mixture with a prior that encourages the mixture components to follow a locally 1-D, globally cyclic structure. This prior can be extended to more general forms; Utsugi uses a discretized Laplacian smoothing prior [O'Sullivan, 1991] that encourage the mixture components to follow a low-dimensional, rectangular structure, and

[5] To be exact, the matrix multiplications scales as $\mathcal{O}(KMD + KND)$, but normally the number of data points, N, exceeds the number of basis functions, M.

which can relatively easily be modified to more complex priors, e.g. to allow for (partial) 'cuts' or 'tears' in the manifold [Utsugi, 1996]. To formalize this, the model consists of a K-component Gaussian mixture with centres, $\mathbf{w}_k$, a common variance β^{-1} and equal mixing coefficients fixed to $1/K$. For the centres we define the prior

$$p(\mathbf{W}|\alpha) = \prod_d^D \left(\frac{\alpha}{2\pi}\right)^{j/2} (|\mathbf{\Delta}^{\mathrm{T}}\mathbf{\Delta}|^+)^{1/2} \exp\left(-\frac{\alpha}{2}\|\mathbf{\Delta}\widehat{\mathbf{w}}_d\|^2\right),$$

where $\mathbf{W}$ is the $K \times D$ matrix holding the centres, $\mathbf{w}_k$, as its rows, $\widehat{\mathbf{w}}_d$ is the dth column of $\mathbf{W}$, $\mathbf{\Delta}$ is a matrix representing a discretized smoothing operator on the latent (topological) space, $|\mathbf{\Delta}^{\mathrm{T}}\mathbf{\Delta}|^+$ denotes the product of the j positive eigenvalues of $\mathbf{\Delta}^{\mathrm{T}}\mathbf{\Delta}$, and α controls the degree of smoothing imposed. Utsugi gives examples using a second order smoother discretized on a lattice in a 1-D latent space,

$$\Delta_{ij} = \begin{cases} -2 & \text{if } |(i-j)+1| = 0, \\ 1 & \text{if } |(i-j)+1| = 1, \\ 0 & \text{otherwise.} \end{cases} \qquad i = 1, \ldots, (K-2); j = 1, \ldots, K,$$

Given a data set, $\{\mathbf{t}_1, \ldots, \mathbf{t}_N\}$, we can write the penalized log-likelihood function as,

$$\ell = \sum_n^N \ln p(\mathbf{t}_n|\mathbf{W}, \beta) + \ln p(\mathbf{W}|\alpha)$$

We can maximise this using an EM-algorithm, where the E-step is identical to that of the GTM, while in the M-step, we are solving

$$\left(\mathbf{G} + \frac{\alpha}{\beta}\mathbf{\Delta}^{\mathrm{T}}\mathbf{\Delta}\right)\mathbf{W} = \mathbf{RT},$$

for $\mathbf{W}$, where $\mathbf{G}$, $\mathbf{R}$ and $\mathbf{T}$ are defined as for the GTM, equation (3.13). Comparing these two equations highlights the key difference between the two models. The GTM consists of a *constrained*, rather than a *regularized*, Gaussian mixture. Alternatively, regularization can be seen as imposing *soft* constraints on the mixture components, in contrast to the *hard* constraints enforced by the GTM. Another important difference, is that this elastic net model, due to its use of a *discretized* smoother, does not define a mapping from the latent space to the data, and hence no explicit manifold in the data space. A new point in the latent space which does not coincide with any point in the lattice of the smoother can therefore not be mapped to the data space, as is the case with the GTM. In section 6.9 we discuss an alternative way of defining the mapping from latent to data space in the GTM, which imposes soft constraints on the mixture components, using a Gaussian process prior.

The relationship of the generalised elastic net to the SOM is largely analogous with that of the GTM, discussed in the previous paragraphs. Utsugi [1997] shows how the Laplacian smoother alternatively can be written in the form of a (discretized) kernel smoother.

3.4.3 Principal curves

The original principal curve algorithm, discussed in section 2.1.2, is in some ways closer to the SOM than the GTM, in that

- each data point is associated with a single point on the curve, namely its projection on the curve, and

- for finite data sets, the conditional estimates of the curve are smoothed over a neighbourhood defined in the parameter space of the curve, corresponding to the neighbourhood function of the SOM.

Note that the projections onto the curve change on a continuous scale in the parameter spaces, as the curve adapts. The re-assignment of data points to the static nodes in the SOM can be seen as a discretization of this process.

The revisited version of the principal curve, discussed in section 2.2.4, is closer to the GTM and the elastic net model discussed in the previous section. It also generates a regularized Gaussian mixture, but uses a cubic spline smoother, and the number of components in the mixture equals the number of data points. Tibshirani [1992] suggests the possible extension of the revisited principal curve model to structures of higher dimensionality, but goes no further.

3.4.4 Density networks

The density networks model [MacKay and Gibbs, 1997, MacKay, 1995] is fairly general and the GTM model proposed here can be seen as a particular instance, with a particular form for the prior distribution in the latent space, given in (3.3), and the mapping from latent to data space being implemented using a generalised linear regression model which is optimized using the EM algorithm. As mentioned at the end of section 2.2.5, MacKay and Gibbs use a conjugate gradient routine for the optimization. The gradient is computed by averaging over the posterior distribution over the latent variables and since MacKay and Gibbs approximates this distribution over a finite sample of points in the latent space, the computation of this distribution will be equivalent to the computation of responsibilities in the GTM.

MacKay and Gibbs [1997] also discuss a hybrid Monte-Carlo approach [Neal, 1992] for modelling the posterior distribution, which holds potential to resolve problems that arise as the dimensionality of the latent space increase.

3.4.5 Auto-associative networks

The most important difference between the auto-encoder and the GTM is that the former does not define a distribution over the latent space (the space of the hidden units) and hence it is not a generative model. However, the auto-encoder has the advantage of effectively dealing with latent spaces of higher dimension, since the E-step of the generative models, which computes a (discretized) distribution over the whole of the latent space, is replaced by a straightforward 'projection' in the latent space (the space of activations from the bottleneck layer), which is a single point, computed by the forward propagation from the input layer to the bottleneck layer. These projections are then mapped, by the second half of the auto-encoder (bottleneck to targets), which corresponds to the mapping from latent to data space in the GTM.

This 2-stage view of the auto-encoder has provided inspiration for developments of generative latent variable models, in which a *recognition* model, analogous with the input-to-bottleneck mapping, is used to (approximately) model the conditional distribution over a set of latent variables, given a data set. This distribution is then mapped to the data space, in analogy with the bottleneck-to-targets mapping, resulting in a *generative* model in the data space. These ideas were first developed for latent class models, where the observed data consist of binary vectors, e.g. binary images [Dayan et al., 1995]. More recently, also models for non-linear factor analysis and topographic maps have been suggested within this framework [Ghahramani and Hinton, 1998, Hinton and Ghahramani, 1997].

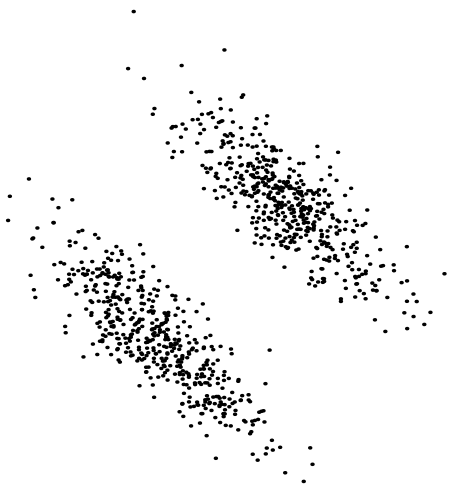


Figure 3.11: A toy data set consisting of 1000 data points drawn at random from a distribution defined by two correlated Gaussians in 2-D.

3.5 Discussion

This chapter has introduced the basic GTM model. There are a number of ways in which this model can be generalized, extended or adapted. The important point, however, is that any such future developments can be carried out within the framework of probability theory. We have in this chapter left a number of parameters of the GTM model unspecified — in chapter 5, we will see how we can find suitable values for these using Bayesian methods. In chapter 6 several other suggestions will be given on how the GTM model can be extended in a principled manner, providing further evidence of the benefits of using a generative, probabilistic model.

We have also seen how the GTM can be used for visualization of data from the modelled distribution, based on the posterior distribution over the latent space induced by a point in the data space. In chapter 4, we will see how we can use the fact that the GTM defines a continuous manifold in the data space to further enhance its capabilities for visualization by the introduction of the *magnification factor*.

A potential problem with the GTM as presented in this chapter, is that it will be best suited to model continuous, low-dimensional distributions of roughly square shape. When this is not the case, the non-linear mapping will try to adapt in order to match the data as well as possible, but that may in turn raise a conflict between the interpretability and the quality of the density model.

Example 3.3 (2 Gaussians in 2-D) *Consider the data set shown in figure 3.11, consisting of two correlated Gaussians in 2-D. Two GTM models were fitted to this data set, both having a* 10×10 *grid of latent points, but one had a rather flexible mapping, with a* 5×5 *grid of basis functions, whereas the other had a minimal* 2×2 *grid of basis functions; both had* $\sigma = 1.0$.

Both models were trained using 60 iterations of EM, the first without using any weight regularization, the second using $\alpha = 0.1$, *and the resulting manifolds are shown in the top left and right panels of figure 3.12; note that these have been plotted using a* 30×30 *grid of latent points. In the bottom panels the corresponding density models are illustrated, together with a test set consisting of 1000 points drawn independently from the same distribution as the training data. The more flexible manifold has been curled up and folded as the training algorithm has tried to achieve an optimal fit*

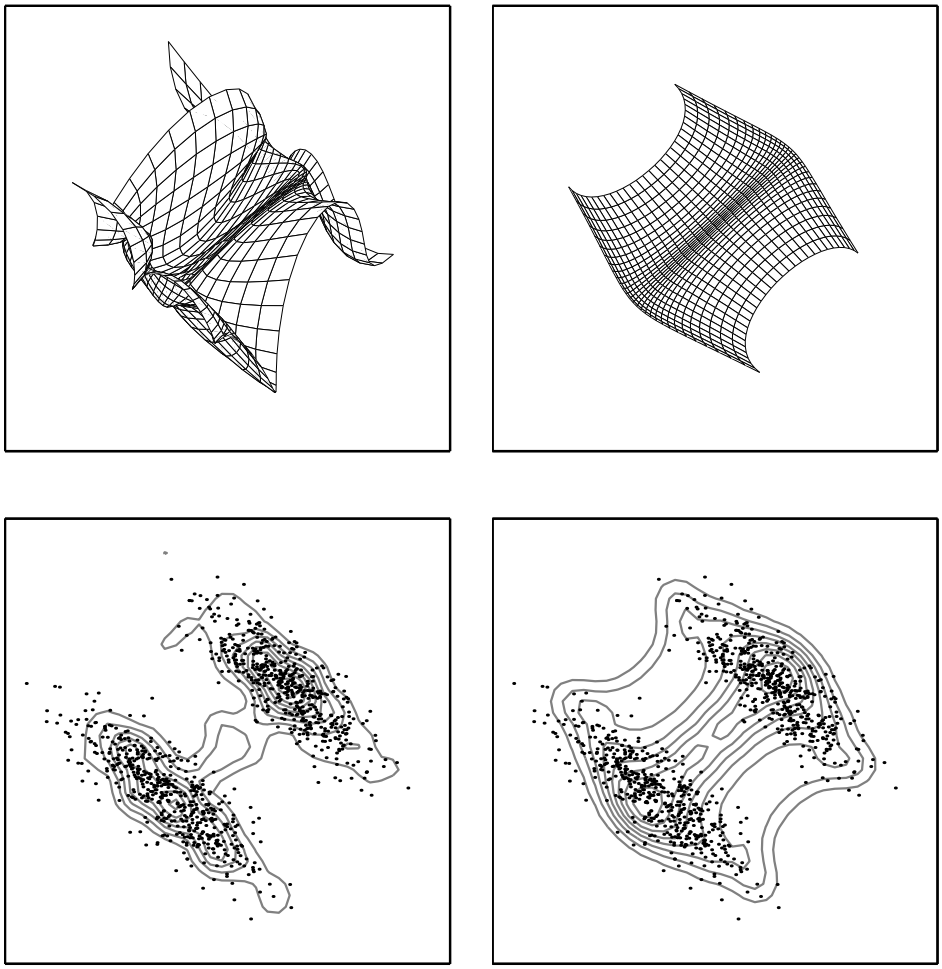


Figure 3.12: The left and right column shows the manifolds (top) and density models (bottom) of the more flexible and the more stiff GTM models, respectively. The manifolds has been plotted using a 30×30 grid of latent points. The density model plots shows contours of constant density, together with a set of independent test data points, plotted as $\cdot$.

to the training data. The stiffer manifold has been too stiff to bend or fold and, as a consequence, the resulting density model is clearly inferior. The log-likelihood scores for the two models are shown in table 3.1.

It should be noted that the experiment in example 3.3 was designed to demonstrate a point. The data consists of two separated clusters with ellipsoid shapes, and since we are fitting a 2-D model to 2-D data, the non-linearity in the GTM will be used entirely to squeeze the single, inherently square shaped manifold to fit two ellipsoid clusters. In this situation, warping the manifold as in the top-left panel of figure 3.12 appears to be the most 'profitable' alternative for the training algorithm, in terms of the trade-off between likelihood and the degrees of freedom available. A second important point to note is that a more flexible model will always fit better to training data, compared to a less flexible one, but this will not necessary generalise to independent test data, a problem known as *overfitting*, which will be further discussed in section 5.2. Indeed, if the flexible model in the example had been even more flexible, or if the training data set had been smaller, the scores in the test data column of table 3.1 may have been reversed.

	Data Set	
Model	Training	Test
Flexible	-2052	-2087
Stiff	-2386	-2358

Table 3.1: Log-likelihood scores on training and test data, for the flexible and stiff GTM models discussed in example 3.3. Both training and test set contained 1000 data points.

A potential solution to problems arising from the fixed, square shaped distribution is to relax the constraint of fixed mixing coefficients, and instead estimating these as part of the training procedure. The training algorithm could then, within given limits, choose the distribution of mixing coefficients that gives the best fit to data, in effect choosing the distribution over the latent space. However, this is likely to have a significant impact on the GTM as a model for visualization. The data in example 3.3, may in such a model project only on the two opposite edges of the latent space. The fixed mixing coefficients, on the other hand, are encouraging the training algorithm to make use of as many mixture components as possible, given the constraints on the flexibility of the mapping.

If the ultimate goal is density modelling and a (single) GTM model indicates clusters in the data, it may be that a better density model can be obtained by using a mixture model, which may have a GTM as one or more of its components.

Chapter 4

Magnification Factors

The concept of magnification factors initially arose in studies of the formation of topological maps in the visual, auditory and somatosensory regions of the cortex of mammalian brains [see e.g. Suga, 1991, Kaas et al., 1981]. It refers to how a region in a sensory space (e.g. a region of the retina in the eye) is being mapped to a, proportionally, much larger region of the cortex; the region in the sensory space is said to be 'magnified'. It was naturally carried over to the biologically inspired SOM model, where it came to represent how the topological map was being (locally) stretched and compressed when embedded in the data space, in order to make the density of reference vectors 'match' the density of the training data. More precisely, Kohonen [1995] uses the term 'magnification factor' to mean "the inverse of the point density" of the reference vectors, and theoretical analysis of the magnification factor, in this sense, was carried out by Ritter and Schulten [1986, 1988]. We will use the term 'magnification factor' to refer to the stretching and compression of the manifold representing the latent space, when embedded in the data space. Since the GTM density model consists of a set of equally weighted Gaussians with a common noise model, which corresponds to the regular grid of points in the latent space, the stretching and compression of the manifold will be driven by the objective of the training algorithm, to make density model match the distribution of the training data.

Since for the original version of the SOM, the topological map is represented in the data space only in term of a discrete set of reference vectors, the magnification factor, according to the definition used here, will only be available in a discretized form, as the ratio of distances between reference vectors in the data space and distances between the corresponding distances between nodes on the map. A method, called the U-matrix method, was proposed by Ultsch and Siemon [1990], which visualizes distances between reference vectors on the topological map; this method will be further discussed in section 4.2.

The GTM, by contrast, defines a continuous manifold embedded in the data space, which allows us to derive methods for computing the magnification factor as a continuous function over manifold (and hence over the latent space), as will be discussed in section 4.3. As will be described in section 4.4, this method is also applicable to the batch version of the SOM, provided certain conditions are met. First, however, we give further motivation for the use of magnification factors.

4.1 Motivation

What does the locations of two points in the latent space tell us about the locations of the corresponding two points in the data space? Given the discussion in the previous chapter on the topology

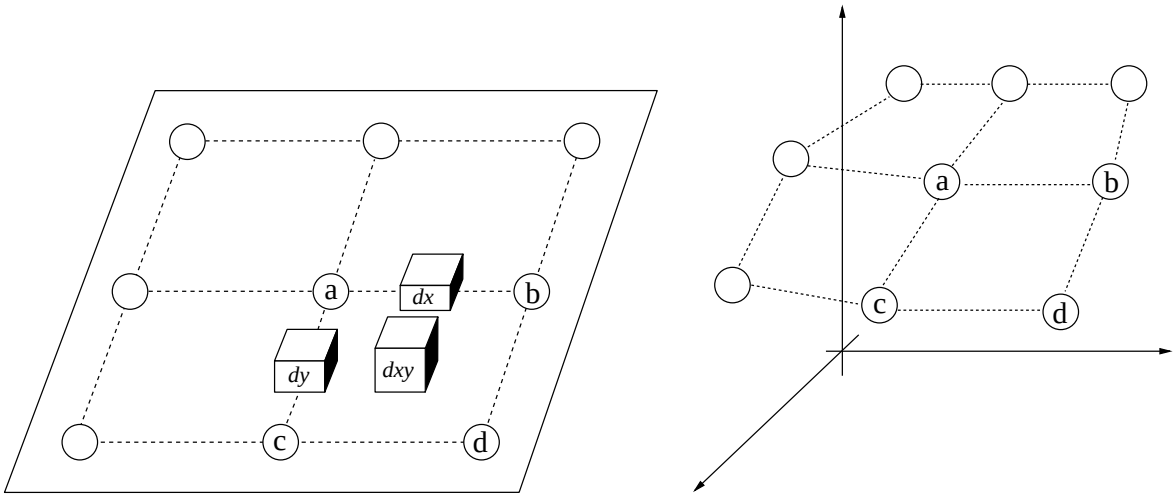


Figure 4.1: An illustration of the U-matrix method. To the left is the topographic map, where the circles represents nodes and the 3-D bars represent the distances between the corresponding reference vectors, shown as circles in the data space to the right.

preserving properties of the the GTM, the answer may seem obvious: nearby points in the latent space map to nearby points in the data space. But how near is 'nearby' and will this value be constant over the latent space? The answer to this last question is generally no. Since the mapping between the latent space and the data space is non-linear, what is nearby will vary over the latent space. In fact, we have already seen an example of this — the toy data set used to demonstrate learning in the GTM in example 3.1 is not uniformly distributed over the curve that it follows. Consequently, the GTM trained on this data will stretch the manifold in regions of low data density and compress it in regions of high density. This is reflected in the bottom right plot of of figure 3.2, showing the converged model; the mixture components, that correspond to a uniform grid in the 1-D latent space, are spread out towards the end of the manifold and compressed together in the bends.

Thus, it is clear that 'nearby' is something relative which varies over the latent space, and this can have important implications on how data is visualized in our model. Clusters of data which are well separated in the data space may appear much closer in when visualized in the latent space. However, if we find out how the manifold in the data space is being stretched or compressed, locally, that should give an idea of what nearby means at different positions in the latent space. This could reveal boundaries between clusters as regions where the manifold in the data space undergoes high stretch.

4.2 The U-matrix method

The unified distance matrix (U-matrix) method [Ultsch and Siemon, 1990, Ultsch, 1993] provides an estimate of the magnification factor for the SOM by visualizing the distances between reference vectors in the data space on the topographic map. The method is illustrated in figure 4.1; the dx bar on the map represents the square distance between reference vectors a and b; similarly, the dy bar represents the square distance between reference vectors a and c; the dxy bar, finally, represents the averaged squared distances between reference vector pairs a–d and b–c. Instead of using 3-D bars, distances can be visualized using grey-scale or colour coding, as will be shown in the examples in section 4.5.

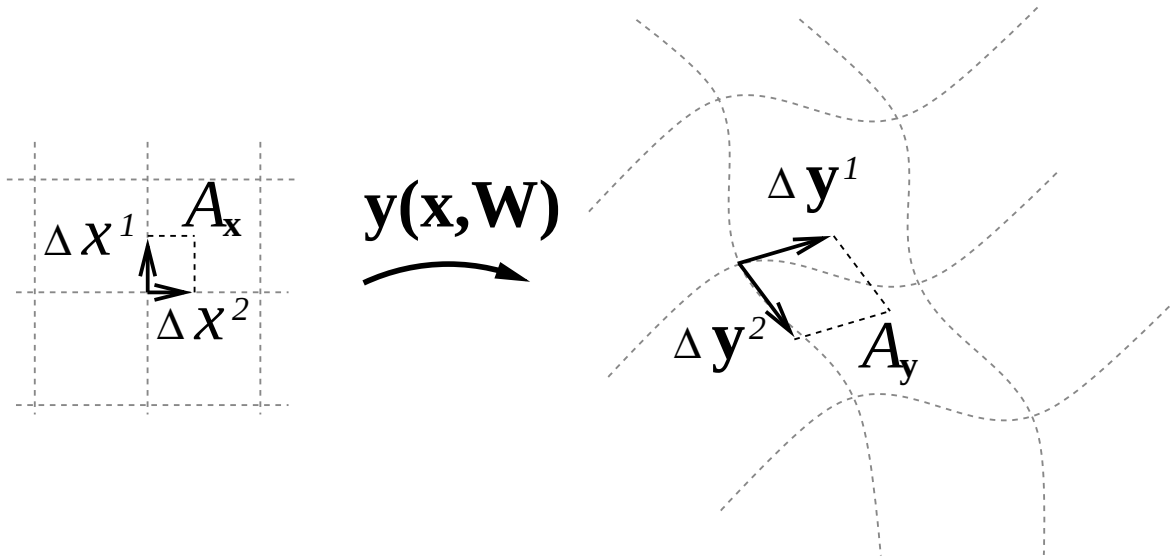


Figure 4.2: An illustration of the magnification factor for a 2-D latent space — the vectors Δx^1 and Δx^2, forming a rectangle in the latent space (left) with area $A_{\mathbf{x}}$, are mapped to $\Delta \mathbf{y}^1$ and $\Delta \mathbf{y}^2$, respectively, forming a parallelogram with area $A_{\mathbf{y}}$ in the data space (right)

4.3 Continuous magnification factors

We introduce the method for computing the magnification factor as a continuous function over latent space by first looking at the special case were the latent space is two-dimensional. This is partly because this is the by far most common case, especially when the ultimate aim is visualization of data. More importantly, the two-dimensional case provides an intuitive understanding of the general treatment.

We are interested in how a region in the latent space is being stretched (or compressed) when mapped into the data space. More precisely, we want to find the areal ratio between an infinitesimal rectangle in the latent space with area $A_{\mathbf{x}}$, and its 'image' in the data space with area $A_{\mathbf{y}}$, as shown in figure 4.2.

As the Δx's in the latent space go to zero, we can treat the mapping as linear around the point of interest, and we get the $1 \times D$ vectors $\Delta \mathbf{y}^1$ and $\Delta \mathbf{y}^2$ in terms of the partial derivatives of the mapping with respect to the first and second dimension of the latent space, respectively.

$$\Delta \mathbf{y}^1 = \frac{\partial \mathbf{y}}{\partial x^1} \Delta x^1 \tag{4.1}$$

$$\Delta \mathbf{y}^2 = \frac{\partial \mathbf{y}}{\partial x^2} \Delta x^2 \tag{4.2}$$

By standard geometrical arguments, the square of the area $A_{\mathbf{y}}$ can be written as

$$A_{\mathbf{y}}^2 = \|\Delta \mathbf{y}^1\|^2 \|\Delta \mathbf{y}^2\|^2 - (\Delta \mathbf{y}^1 \Delta \mathbf{y}^{2^{\mathrm{T}}})^2 \tag{4.3}$$

The last term in (4.3) expresses the fact that we must consider the correlation in direction between $\Delta \mathbf{y}^1$ and $\Delta \mathbf{y}^2$ — if they were orthogonal to each other, this term would be zero, whereas if they were parallel, it would equal the first term and $A_{\mathbf{y}}$ would be zero (the parallelogram folding to a line).

Now, by using (4.1) and (4.2), and after some re-arranging, we get the magnification factor as

$$\frac{dA_{\mathbf{y}}}{dA_{\mathbf{x}}} = \sqrt{\left\|\frac{\partial \mathbf{y}}{\partial x^1}\right\|^2 \left\|\frac{\partial \mathbf{y}}{\partial x^2}\right\|^2 - \left(\frac{\partial \mathbf{y}}{\partial x^1}\frac{\partial \mathbf{y}}{\partial x^2}^{\mathrm{T}}\right)^2}. \tag{4.4}$$

The partial derivatives of the mapping $\mathbf{y}(\mathbf{x}, \mathbf{W})$, with respect to the latent variable, $\mathbf{x}$, are easily obtained from (3.8), yielding

$$\frac{\partial \mathbf{y}}{\partial x^l} = \boldsymbol{\psi}_l \mathbf{W} \tag{4.5}$$

where $\boldsymbol{\psi}_l$ is an $1 \times M$ vector, containing the partial derivatives of the basis functions with respect to x^l, which we get from (3.9) as

$$\psi_{lm} = \begin{cases} -\phi_m(\mathbf{x})(x^l - \mu_m^l)\sigma^{-2} & \text{if } m \leq M_{\mathrm{NL}}, \\ 1 & \text{if } m = M_{\mathrm{NL}} + l, \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

4.3.1 The General case

The results obtained in the 2-D case can be extended to latent spaces of higher dimensionality, by considering the volumetric ratio between an infinitesimal, L-dimensional hypercuboid in the latent space, with volume $V_{\mathbf{x}}$, and its image in the data space, with volume $V_{\mathbf{y}}$. Again, with the sides of the hypercuboid in the latent space going to zero, we can regard the manifold embedded in the data space as locally linear, and so $V_{\mathbf{y}}$ is contained in an L-dimensional parallelepiped, as illustrated in figure 4.3, the volume of which is given by the determinant of the matrix containing the sides of the parallelepiped as its rows. This matrix, which we denote with $\mathbf{J}$, is the Jacobian of the mapping $\mathbf{y}(\mathbf{x}, \mathbf{W})$, i.e. the partial derivatives of $\mathbf{y}$ with respect to $\mathbf{x}$,

$$J_{ld} = \frac{\partial y^d}{\partial x^l} \tag{4.7}$$

Using (4.5) and (4.6), we can write $\mathbf{J}$ as

$$\mathbf{J} = \boldsymbol{\Psi}\mathbf{W}, \tag{4.8}$$

where $\boldsymbol{\Psi}$ is an $L \times M$ matrix with elements ψ_{lm}, as defined in (4.6).

In general, $\mathbf{J}$ is not square but $L \times D$, reflecting the fact that the L-dimensional parallelepiped lies embedded in the D-dimensional data space. In this form, the determinant of $\mathbf{J}$ is undefined, but we can resolve this by finding[1] a $D \times L$ matrix, $\widehat{\mathbf{M}}$, with orthonormal columns that span the row-space of $\mathbf{J}$ and then compute

$$\widehat{\mathbf{J}} = \mathbf{J}\widehat{\mathbf{M}}.$$

Since the columns of $\widehat{\mathbf{M}}$ are orthonormal and span the row-space of $\mathbf{J}$, the lengths of and angles between the row vectors of $\mathbf{J}$ and $\widehat{\mathbf{J}}$ are identical, and thus we would get the volume $V_{\mathbf{y}}$ by computing the determinant of $\widehat{\mathbf{J}}$, which is $L \times L$.

Eventually, we can compute this volume in a more efficient way; since lengths and mutual angles of the row vectors in $\mathbf{J}$ and $\widehat{\mathbf{J}}$ would be identical, it follows that

$$\widehat{\mathbf{J}}\widehat{\mathbf{J}}^{\mathrm{T}} = \mathbf{J}\mathbf{J}^{\mathrm{T}}.$$

[1] Using Gram-Schmidt orthogonalization [see e.g. Strang, 1988]

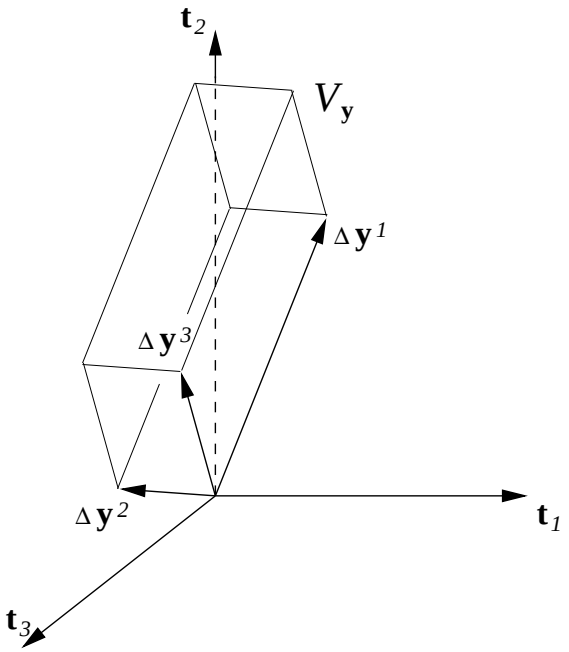


Figure 4.3: An L-dimensional ($L = 3$) parallelepiped whose sides, $\Delta\mathbf{y}^1, \ldots, \Delta\mathbf{y}^L$, are given by the partial derivatives of the mapping from latent to data space, with respect to the latent variables.

From this and the properties of the determinant, we have

$$\begin{aligned}(\det(\widehat{\mathbf{J}}))^2 &= \det(\widehat{\mathbf{J}})\det(\widehat{\mathbf{J}}) \\ &= \det(\widehat{\mathbf{J}})\det(\widehat{\mathbf{J}}^{\mathrm{T}}) \\ &= \det(\widehat{\mathbf{J}}\widehat{\mathbf{J}}^{\mathrm{T}}) \\ &= \det(\mathbf{J}\mathbf{J}^{\mathrm{T}}),\end{aligned}$$

and thus

$$\frac{dV_{\mathbf{y}}}{dV_{\mathbf{x}}} = \sqrt{\det(\mathbf{J}\mathbf{J}^{\mathrm{T}})} = \sqrt{\det(\mathbf{\Psi}\mathbf{W}\mathbf{W}^{\mathrm{T}}\mathbf{\Psi}^{\mathrm{T}})}. \tag{4.10}$$

It is easy to verify that this formula equals (4.4) if $L = 2$.

An alternative derivation

These results can alternatively be derived using the theory of differential geometry [Bishop et al., 1997c,d]. Throughout this section, we will adopt the convention from differential geometry of summing over repeated raised and lowered indices. In this approach, we regard the Cartesian coordinate system defined on the latent space, x^i, to be mapped to a corresponding curvilinear coordinate system, ξ^i, defined on the manifold embedded in the data space. We then consider the transformation from ξ^i, at a point $P_{\mathbf{y}}$ in the manifold, to an L-dimensional Cartesian coordinate system, $\zeta^\mu = \zeta^\mu(\boldsymbol{\xi})$. The

squared length element in these coordinates is then given by

$$\begin{aligned} ds^2 &= \delta_{\mu\nu} d\zeta^\mu d\zeta^\nu \\ &= \delta_{\mu\nu} \frac{\partial \zeta^\mu}{\partial \xi^i} \frac{\partial \zeta^\nu}{\partial \xi^j} d\xi^i d\xi^j \\ &= g_{ij} d\xi^i d\xi^j, \end{aligned}$$

where we have introduced the metric tensor, $\mathbf{g}$, whose components are given by

$$g_{ij} = \delta_{\mu\nu} \frac{\partial \zeta^\mu}{\partial \xi^i} \frac{\partial \zeta^\nu}{\partial \xi^j}. \tag{4.11}$$

The volume element $dV_{\mathbf{x}}$ in the latent space can be related to the corresponding element in the data space $dV_{\mathbf{y}}$, through the determinant of the Jacobian of the transformation $\xi \to \zeta$,

$$\begin{aligned} dV_{\mathbf{y}} &= \prod_{\mu}^{L} d\zeta^\mu = \det(\widehat{\mathbf{J}}) \prod_{i}^{L} d\xi^i \\ &= \det(\widehat{\mathbf{J}}) \prod_{i}^{L} dx^i = \det(\widehat{\mathbf{J}}) dV_{\mathbf{x}}, \end{aligned} \tag{4.12}$$

where the Jacobian, $\widehat{\mathbf{J}}$, is given by

$$\widehat{J}_{\mu i} = \frac{\partial \zeta^\mu}{\partial \xi^i}. \tag{4.13}$$

If we study (4.11) and (4.13), we see that

$$\mathbf{g} = \widehat{\mathbf{J}} \widehat{\mathbf{J}}^{\mathrm{T}},$$

so by the properties of the determinant,

$$\det(\widehat{\mathbf{J}}) = \sqrt{\det(\mathbf{g})},$$

and, from (4.12),

$$\frac{dV_{\mathbf{y}}}{dV_{\mathbf{x}}} = \sqrt{\det(\mathbf{g})} \tag{4.14}$$

We therefore seek an an expression for the metric tensor, in terms of the non-linear mapping from latent to data space. Again we consider the squared length element ds^2 but this time in the D-dimensional Cartesian coordinate system of the data space, where we get

$$\begin{aligned} ds^2 &= \delta_{pq} dy^p dy^q \\ &= \delta_{pq} \frac{\partial y^p}{\partial x^i} \frac{\partial dy^q}{\partial x^j} dx^i dx^j \\ &= g_{ij} dx^i dx^j, \end{aligned}$$

and so we get the metric tensor $\mathbf{g}$ as

$$g_{ij} = \delta_{pq} \frac{\partial y^p}{\partial x^i} \frac{\partial dy^q}{\partial x^j},$$

Using this, (4.14) and (4.7), we get

$$\begin{aligned} \frac{dV_{\mathbf{y}}}{dV_{\mathbf{x}}} &= \sqrt{\det(\mathbf{g})} \\ &= \det\left(\delta_{pq} \frac{\partial y^p}{\partial x^i} \frac{\partial dy^q}{\partial x^j} \right)^{1/2} \\ &= \sqrt{\det(\mathbf{J}\mathbf{J}^{\mathrm{T}})}, \end{aligned}$$

and so we have recovered (4.10).

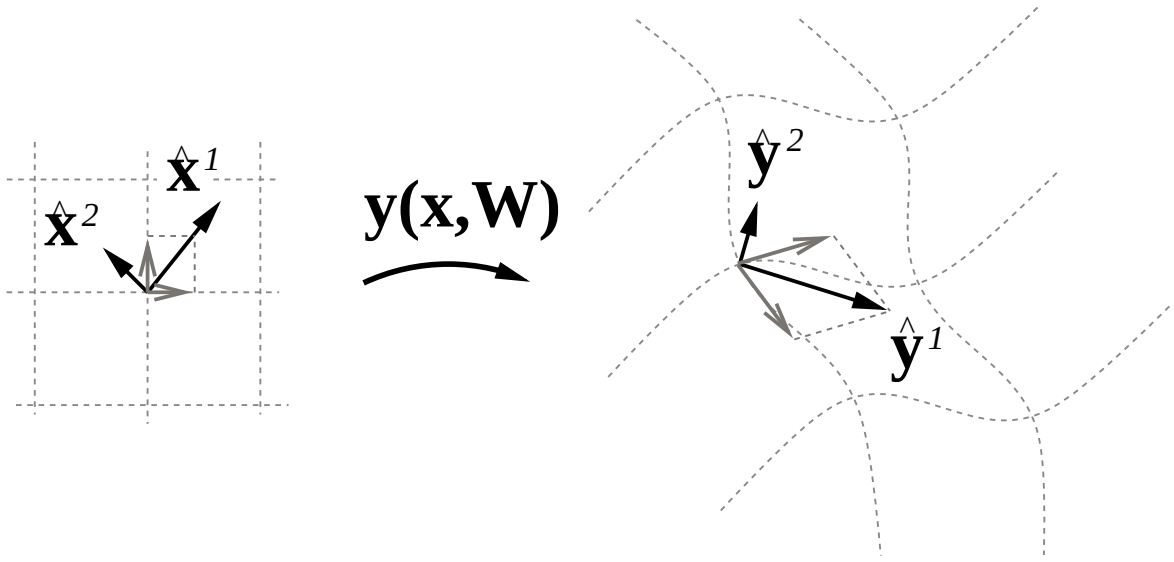


Figure 4.4: A 2-D illustration of the directions of stretch, given by vectors $\hat{\mathbf{y}}^1$ and $\hat{\mathbf{y}}^2$ in the data space (right), with corresponding vectors $\hat{\mathbf{x}}^1$ and $\hat{\mathbf{x}}^2$ in the latent space (left).

4.3.2 The Direction of stretch

So far, we have only considered how to compute the areal[2] magnification factor over the embedded manifold. However, when the manifold is more than one dimensional, stretching in one direction can be offset by compression in another direction — a 2-by-2 rectangle has an area of 4, but then so has a 1-by-4 rectangle. We would therefore like to find not only the degree of areal magnification, but also the direction of any compression or stretching.

Intuitively, we want to decompose the stretching of the manifold into its 'principal directions', as illustrated in figure 4.4. This involves finding the single direction in the latent space along which we find the largest magnitude of the partial derivatives, and then repeat this procedure until we have spanned the latent space, with the additional constraint that each new direction must be orthogonal to all directions found so far.

Put more formally, to find the directions of stretch at a point $P_{\mathbf{y}}$ in the manifold, corresponding to the point $P_{\mathbf{x}}$ in the latent space, we want to find the eigenvalues and eigenvectors of the outer product matrix of $\partial\mathbf{y}/\partial x_1|_{P_{\mathbf{y}}}, \ldots, \partial\mathbf{y}/\partial x_L|_{P_{\mathbf{y}}}$, defined as

$$\sum_{l}^{L} \left.\frac{\partial \mathbf{y}}{\partial x_l}\right|_{P_{\mathbf{y}}}^{\mathrm{T}} \left.\frac{\partial \mathbf{y}}{\partial x_l}\right|_{P_{\mathbf{y}}}.$$

Note that this a $D \times D$ matrix, but it has rank L. All subsequent calculations in this sub-section are understood to be relative to the points $P_{\mathbf{y}}$ and $P_{\mathbf{x}}$, so in the interest of clarity these indices will be dropped.

As discussed in the previous section, the Jacobian $\mathbf{J}$, defined in (4.8), has $\partial\mathbf{y}/\partial x_1, \ldots, \partial\mathbf{y}/\partial x_L$ as it rows, so our desired outer product matrix can be expressed as $\mathbf{J}^{\mathrm{T}}\mathbf{J}$. We can identify eigenvectors $\hat{\mathbf{y}}^l$ and eigenvalues λ_l, $l = 1, 2, \ldots, L$, such that

$$\hat{\mathbf{y}}^l \mathbf{J}^{\mathrm{T}}\mathbf{J} = \lambda_l \hat{\mathbf{y}}^l. \tag{4.15}$$

[2]We will keep using terms like 'area' and 'areal', since a 2-D latent space is by far the most common case. However, the technique described also applies to cases where $L > 2$.

However, what we are really interested in are the corresponding vectors in the latent space, but since $\mathbf{y} = \mathbf{xJ}$ around the point of interest, we can write (4.15) as

$$\hat{\mathbf{x}}^l \mathbf{J}\mathbf{J}^{\mathrm{T}}\mathbf{J} = \lambda_l \hat{\mathbf{x}}^l \mathbf{J}, \tag{4.16}$$

and since $\mathbf{J}$ is $L \times D$ and has rank L, it has got a right-inverse, and hence

$$\hat{\mathbf{x}}^l \mathbf{J}\mathbf{J}^{\mathrm{T}} = \lambda_l \hat{\mathbf{x}}^l. \tag{4.17}$$

Thus we have identified the directions and magnitudes of stretch in the latent space with the eigenvectors and eigenvalues of $\mathbf{J}\mathbf{J}^{\mathrm{T}}$ or, equivalently, the metric tensor $\mathbf{g}$.

4.4 Magnification factors for the BSOM

The techniques presented in the previous sections can also be applied to the batch version of the self-organizing map (BSOM), provided that the neighbourhood function used is continuous over the topographic space. As discussed in section 3.4.1, the update formula for the reference vectors in the training algorithm for the BSOM can be re-written as a kernel regression formula

$$\mathbf{y}^{k'} = \sum_{k}^{K} F(\mathbf{x}_{k'}, \mathbf{x}_k)\mathbf{m}_k \tag{3.18}$$

where

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{\mathbf{t} \in \mathcal{T}_k} \mathbf{t}. \tag{3.17}$$

and

$$F(\mathbf{x}, \mathbf{x}_k) = \frac{N_k h(\mathbf{x}, \mathbf{x}_k)}{\sum_j N_j h(\mathbf{x}, \mathbf{x}_j)}. \tag{3.19}$$

If the neighbourhood function, $h(\cdot)$, is defined to be continuously differentiable — the non-normalized Gaussian,

$$h(\mathbf{x}, \mathbf{x}_k) = \exp\left\{-\frac{\|\mathbf{x} - \mathbf{x}_k\|^2}{2\sigma^2}\right\}, \tag{4.18}$$

will be used in the examples presented below — formulae (3.18)–(3.19) define a continuous mapping from the topographic space to the data space. The BSOM model therefore, just like the GTM, defines a continuous manifold in the data space, and thus we can apply the techniques described in section 4.3 also to the BSOM model. The only difference compared to the GTM, is in the computation of the partial derivatives with respect to the topographic variables, which we get, using (3.18) and (4.18), as

$$\begin{aligned} \frac{\partial \mathbf{y}}{\partial x^l} &= \sum_k \frac{\partial F}{\partial x^l}\mathbf{m}_k \\ &= \sum_k \frac{x^l - x_k^l}{\sigma^2}(F(\mathbf{x}, \mathbf{x}_k)^2 - F(\mathbf{x}, \mathbf{x}_k))\mathbf{m}_k. \end{aligned} \tag{4.19}$$

4.5 Examples

We now look at two examples of the techniques discussed in this chapter, which illustrate how they can be used, and we compare them to the U-matrix technique.

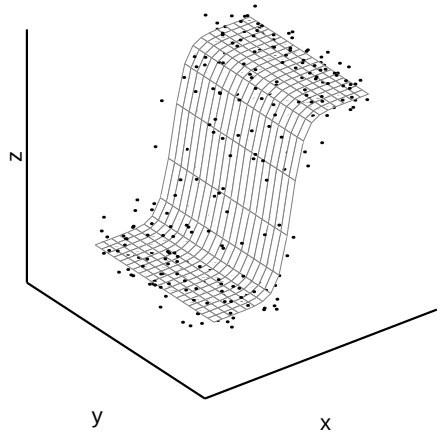


Figure 4.5: Toy data generated from a regular grid in 2-D which is then fed through a tanh-function along the x-direction. The plot shows the data generating manifold plotted in grey and the data points plotted as $\cdot$.

Example 4.1 (A ridge in 3-D) *The first example uses a toy data set, consisting of 400 data points distributed over a 2-D ridge shaped surface in 3-D, shown in figure 4.5. The data was generated from a regular, square grid in* $[-1, 1]^2$*, which gave the* x*- and* y*-coordinates, whereas* $z = \tanh(6x)$*. Finally, Gaussian spherical noise with standard deviation* 0.1 *was added. As can be seen in figure 4.5, this results in the data set being 'stretched' over the* tanh*-function.*

A GTM with a 10×10 *latent grid and* 6×6 *basis functions and a BSOM with* 10×10 *grid of nodes were fitted to this data and magnification factors and the U-matrix were evaluated. The results, which largely agree with what we would expect from this data, are shown in figures 4.6–4.8. Figure 4.6 shows grey-scale plots of the logarithm of the areal magnification factor for the GTM and BSOM, with darker areas corresponding to regions of high stretch (low magnification). Note that although the grid of latent points or nodes is* 10×10*, the magnification factor has been evaluated on a* 40×40 *grid in the latent space — in principle, this grid could have arbitrarily fine resolution. Overlayed on the GTM plot is the posterior mean plot of the data; correspondingly, in the BSOM plot, each node has been labelled with the number of data points won by that node, whenever that number exceeds zero. In figure 4.7, ellipses show the magnitude and direction of stretch, evaluated at the positions of the latent points/nodes for the GTM and BSOM; here the scale is linear and the plots have been individually scaled to avoid overlap between ellipses. Also this plot could be done at a finer resolution if desired. Figure 4.8, finally, shows the U-matrix for the BSOM model.*

Example 4.2 (Leptograpsus Crabs) *In the second example, we will look at a data set containing physical measurements from two species of Leptograpsus rock crabs*[3] *- blue and orange. This set was compiled in order to provide a statistical sample based on which preserved specimen (which have lost their colour) could be classified. There are 50 male and 50 female of each of the two species, so in all there are 200 samples.*

The data set is five dimensional; the measurements of each data vector correspond to the length of the frontal lip, rear width, length along mid-line, maximum width of carapace and body length. These

[3]This data set was obtained from Prof. Brian Ripley's homepage, `http://www.stats.ox.ac.uk/~ripley`.

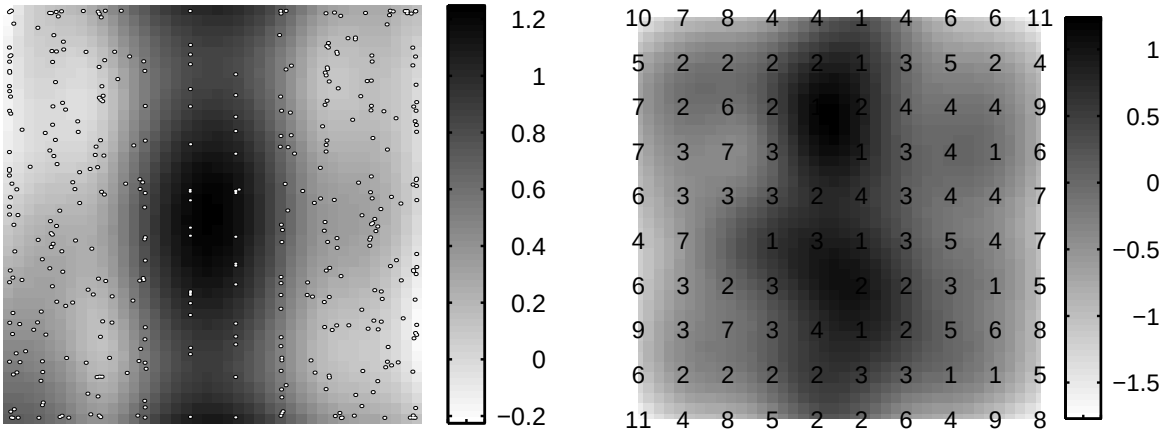


Figure 4.6: Plot of the magnification factor for a GTM (left) and a BSOM (right), trained on the toy data shown in figure 4.5. Overlayed on the GTM plot is the posterior mean plot of the data, while on the BSOM plot, each node has been labelled with the number of data points it 'won'.

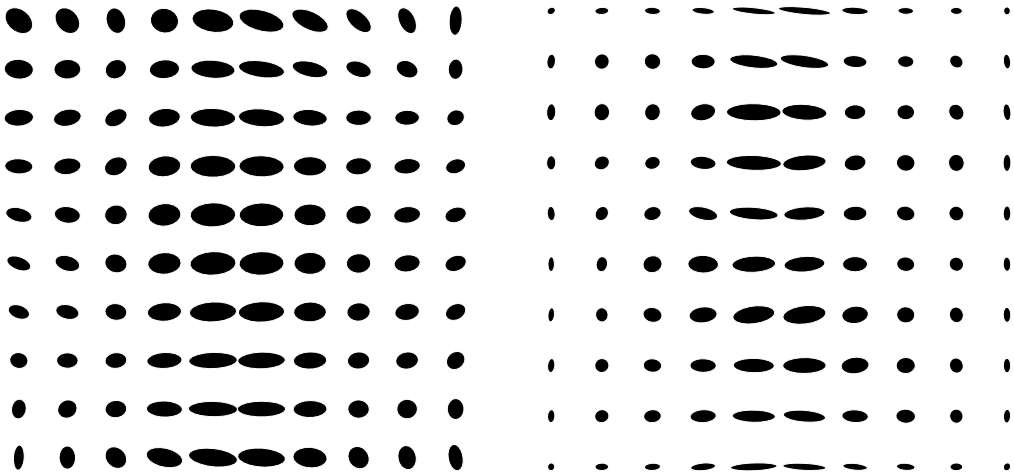


Figure 4.7: Plots showing the direction of stretch for the GTM (left) and the BSOM (right), corresponding respectively to the left and right plots in figure 4.6.

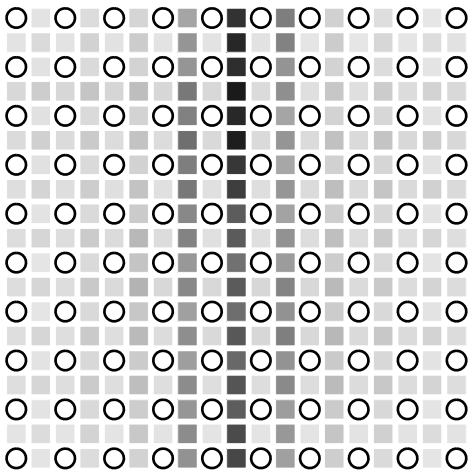


Figure 4.8: The U-matrix plot of for the BSOM model trained on the toy data shown in figure 4.5, for which the corresponding magnification factor is shown in the right panel of figure 4.6.

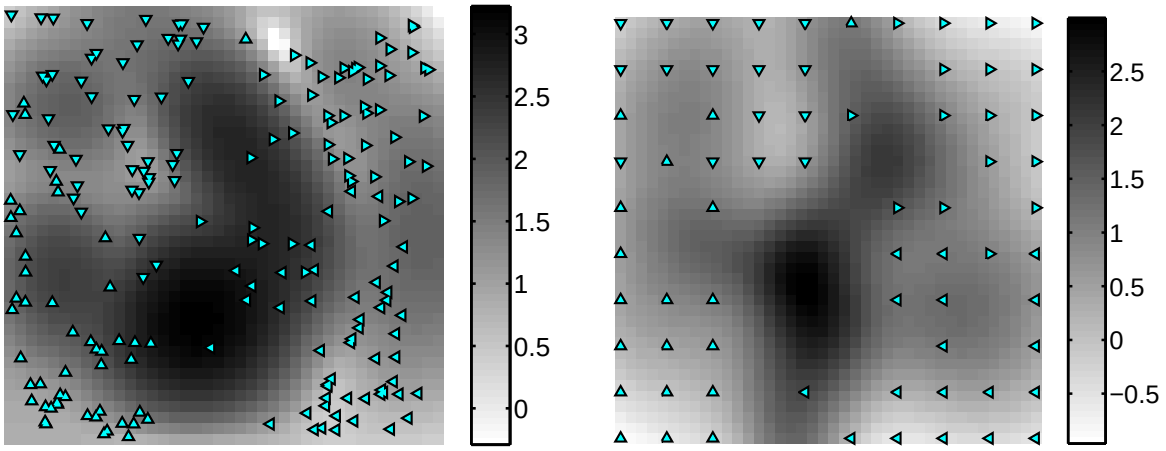


Figure 4.9: Plot of the magnification factor for the GTM (left) and the BSOM (right), trained on the crabs data. Overlayed on the GTM plot is the posterior mean plot of the data, while in the BSOM plot, each node has been labelled according to the dominating class among the data points assigned to them. $\triangledown$ denotes blue male, $\triangle$ denotes blue female, $\triangleleft$ denotes orange female and, finally, $\triangleright$ denotes orange male

measurements are all strongly correlated with the overall size of the crab, so the dominant underlying variable of this data set is size. To remove this effect, each data vector (sample) is normalized to unit mean. This seems reasonable if we assume that there are large and small specimens of males and females in both of the species. We must be aware, however, that there is a risk that this transformation may remove a feature which could be relevant in distinguishing (e.g.) males from females, if on average, there is a difference in size between males and females. After having normalized the individual data vectors, the variables of the data set are normalized to zero mean and unit variance.

As in the previous example, a GTM with 10×10 *latent points and* 4×4 *basis functions and a BSOM with* 10×10 *nodes were fitted to this data. The results are shown in figures 4.9–4.11, following the same 'line of presentation' as in previous example. Figure 4.9 shows a grey-scale plot of the logarithm of the areal magnification factor for the GTM and BSOM, again evaluated on a* 40×40 *grid in the latent space. The GTM plot again shows the posterior mean projection of the data, while in the BSOM plot, nodes has been labelled according to the dominating class among the data points assigned to them. Figure 4.10 shows ellipse-plots of the magnitude and direction of stretch, evaluated at the positions of the latent points/nodes for the GTM and BSOM. Figure 4.11, finally, shows the U-matrix for the BSOM model, with nodes labelled as in figure 4.9.*

4.6 Discussion

The examples given in the previous section suggests the magnification factor can indeed provide useful information, such as regions of stretch in the manifold which separates different regions in the data space. However, if the manifold takes a complex shape in the data space, the resulting magnification factor plot may be rather difficult to interpret.

Recall the data set from example 3.3, which consisted of two Gaussians in 2-D — the manifold of one of the two GTM models fitted to this data ended up having a rather complex shape, shown in the top-left panel of figure 3.12. Figure 4.12 shows the corresponding magnification factor, computed over a 30×30 grid in the latent space.

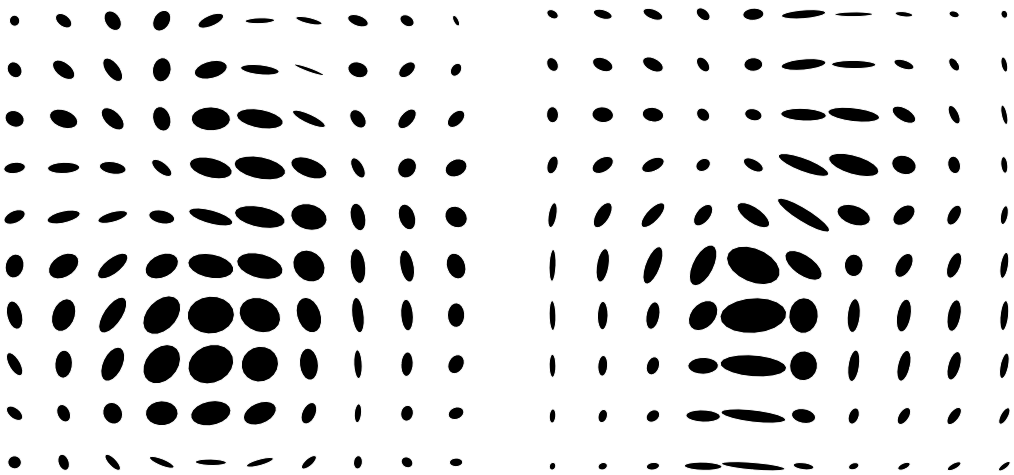


Figure 4.10: Plots showing the direction of stretch for the GTM (left) and the BSOM (right), corresponding respectively to the left and right plots in figure 4.9.

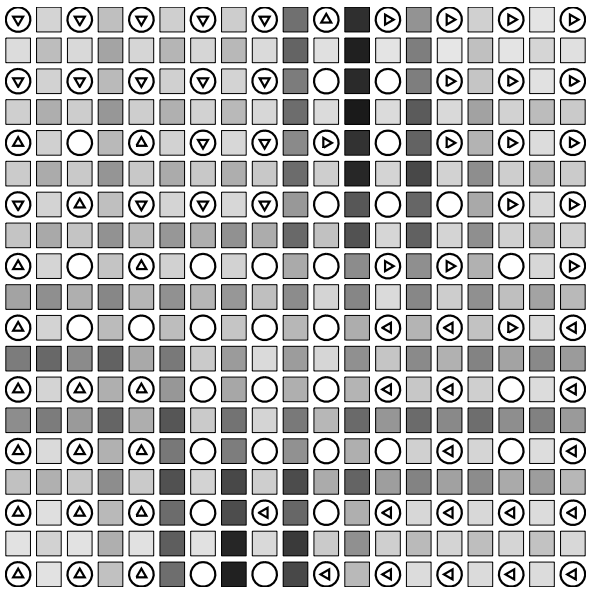


Figure 4.11: The U-matrix plot of for the BSOM model trained on the crabs data, for which the corresponding magnification factor is shown in the right panel of figure 4.9; the nodes has been labelled as in figure 4.9.

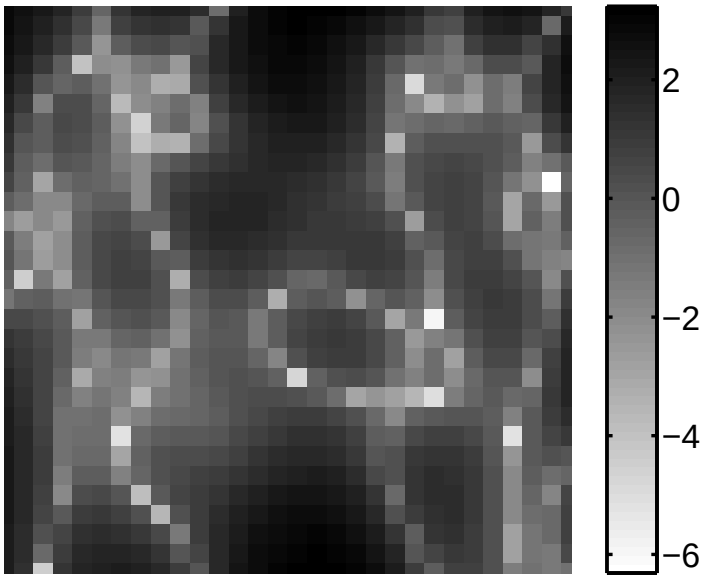


Figure 4.12: The plot shows the magnification factor, plotted on a log-scale for the more flexible GTM model discussed in example 3.3.

One may ask the question whether the magnification factor can be used to detect a severely warped manifold. Certainly, when the data lives in 2-D and has been normalized, a magnification factor ranging from -6 to 2 on a log-scale is an indication that 'something might be wrong', but this probably does not generalize to less extreme examples. Another potential indicator may be to compute the inner product of the vector $\hat{\mathbf{y}}$, representing the principal direction of stretch at some point $P_{\mathbf{y}}$ in the manifold, and the corresponding vectors at other points in the neighbourhood of $P_{\mathbf{y}}$ on the manifold. For a smooth manifold we would expect this inner product to be positive and significant close to $P_{\mathbf{y}}$. However this indicator may raise false alarms in regions of uniform stretch, which is why one would also have to consider the ratios of magnitudes of stretch in the different directions.

In section 3.5, the introduction of adaptable mixing coefficients was suggested as a potential solution to problems linked to the inherent square shape of the distribution of mixture components. It was then pointed out that this could affect GTM as a model for visualization, and this will also include the results can expect from the use of magnification factors. If density mass can be shifted between regions on the manifold, the training algorithm will not have to stretch manifold as much as would have been the case with fixed fixing coefficients.

Chapter 5

Parameter Selection

As is the case with all parametric models, constructing a GTM model will require us to choose values for a number of parameters, such as the number of latent points, the number and form of basis functions and the regularization coefficient, and this choice is likely to have a significant impact on the final model. Common sense will rule out certain combination of parameter values and intuition may provide additional 'rules of thumb', but nevertheless it would be desirable to have principled methods for making these choices. In this chapter we try to address this problem, at least partially, by looking at methods for finding suitable values for α, β and σ. These methods could also, in principle, be used to choose values for other parameters in the model, such as the number of basis functions. We will first look at the roles of the parameters we are about consider and try to understand how different choices affect the model.

5.1 Roles of different parameters

The parameters we concentrate on in this chapter are:

- β — the inverse noise variance,
- α — the inverse variance of the prior over the weights[1], and
- σ — the common width of the Gaussian basis functions ($\mathbf{\Phi}$).

In the approach discussed so far, β is estimated together with $\mathbf{W}$, using maximum-likelihood, while α and σ are set prior to training, essentially by rules of thumb, and then kept fixed.

Whereas $\mathbf{W}$ explicitly defines the shape of the manifold embedded in the data space, α, β and σ will have an implicit effect, by affecting the way the parameters in $\mathbf{W}$ are adapted during the training; we therefore sometimes refer to these as *hyper-parameters*. In this chapter we will normally use the shorter 'parameters' to refer to α, β and σ, while the elements in $\mathbf{W}$ will normally be referred to as 'weights'.

During training, β will affect the smoothness of the manifold at a local level, by defining how much noise or independent variability is associated with the observed variables. As β increases, the variance decreases and so we expect more of the variability in the data to be explained by the variability in the manifold. If β decreases on the other hand, corresponding to an increasing noise, more and

[1] In this chapter we only consider an isotropic prior over the weights, i.e. α is a scalar, but the methods described can also be extended to deal with more general cases.

more of the variance in the data will be regarded as noise, which will result in a smoother manifold. Eventually, if β becomes small enough, the manifold will simply collapse to a point at the sample mean of the data, with all the variance in the data being regarded as noise. The EM algorithm provides a maximum likelihood estimate of β, but these estimates can be overly optimistic, in the sense that they underestimate the noise level, and alternative estimates are discussed below.

σ controls the global smoothness of the manifold, since as the radially symmetric basis function gets broader, they also get more correlated in their responses to points in the latent space. Nearby points in latent space will therefore map to increasingly nearby points in the space of basis function activations, and consequently to increasingly nearby points in the data space, resulting in increasingly stiffer manifolds; an example of this was shown in figure 3.10. It will also be reflected in the fact that, as σ increases, at some point the matrix $\mathbf{\Phi}$ become rank deficient; taken to extreme, we end up with the same case as with small β, with the manifold collapsing to a point or, if we have also incorporated linear basis functions, a PCA-like solution. At the other extreme, as σ gets small, the basis functions eventually become (numerically) completely uncorrelated, and the smooth non-linearity in the manifold 'falls apart'; if σ keeps decreasing the non-linearities may vanish, unless basis function centres coincide with latent points.

α, finally, controls the magnitude of the weights and hence the scale of the manifold. One could argue that constraining the weights would seem unnecessary, since a model that did not get the overall scale right would not be a good model anyway. However, since we are working with finite data sets, degrees of freedom that are not spent on capturing the underlying distribution will be used to fit noise on the data.

5.2 Parameter selection and generalization

When we are trying to train a model on a data set, we are normally not interested in finding a model that perfectly fits the data, but rather one that fits the underlying distribution from which the data was generated. Assuming we are successful, we would expect this model to also fit well to other data sets drawn from the same distribution — we say that the model has good *generalization* capabilities [Bishop, 1995].

The issue of generalization is directly related to parameter selection, since our choice of parameters controls the flexibility of the model. A sufficiently flexible model will be able to fit any finite data set perfectly; a GTM with sufficiently many latent points ($K \geq N$) and flexible enough mapping will place a mixture component at each data point and set the common variance to zero yielding an infinite likelihood. For all other data sets, however, the likelihood under such a model will be zero. Since we assume that our data is generated from a systematic component and a random noise component, independently collected data sets are not expected to be identical. The perfect fit to training data is obviously an extreme example, but it highlights an important problem: a too flexible model will not capture the underlying distribution of a data set, but rather the structure of that particular data set, with its associated noise and artifacts. This phenomenon is known as *overfitting*. On the other hand, if the model is not flexible enough, it may not be able to successfully model the underlying distribution — a situation correspondingly known as *underfitting*. In either case, the resulting model is poor, so the challenge, within the framework we have worked in so far, would be to find a model which is flexible enough to capture the overall structure in the training data set, but not so flexible that it also catches on to features specific to that particular set of data. Since the flexibility of the model is controlled by the parameters, this corresponds to finding suitable values for these.

A different approach to learning is taken in Bayesian statistics. The Bayesian viewpoint is that, rather than trying to find a single set of parameter values, we should work with a *distribution* over possible values. Before we have seen any data, this distribution is specified entirely from whatever prior knowledge is available, and is therefore called a *prior* distribution, or just prior. Once data arrives, we combine the likelihood of the data with our prior, using Bayes' theorem, to yield a *posterior* distribution over parameter values. Typically, this posterior distribution will be narrower than the prior, since in the light of the data, certain parameter values will appear more likely than others. This treatment applies to all parameters — 'ordinary' parameters, such as the elements of $\mathbf{W}$ in the GTM, as well as hyper-parameters, such as α, β and σ — and the Bayesian framework naturally formalizes the relationship between different kinds of parameters through conditional probability distributions; thus we will see in the next section how the regularised log-likelihood function emerges from the conditional probability distribution over $\mathbf{W}$, given α, β, σ and the set of training data. We will also see how we can use the Bayesian machinery to infer suitable values for α, β and σ, but first we consider more traditional methods.

One method for parameter selection which we can use, provided we have sufficient amounts of data, is to partition the data available into one set that we use for fitting, or training, the model, called the training set, and one set that we use to evaluate the performance of the trained model, which we call a validation set. By training and evaluating models over a range of parameter values, we can find the parameter values that result in the best performance on the validation set. This is motivated by the belief that this model is flexible enough to model the common structure of the training and validation set, but not so flexible that it also fits noise and features specific to the training set. There are two obvious drawbacks with this approach:

- it relies on the availability of a sufficiently large set of data and
- it requires significant amounts of computation, which grows exponentially with the number of parameters.

What 'sufficient' amounts of data of data is of course varies with the problem; in general, the more complex the underlying structure is, the more flexible our model must be and, hence, the more data is required. One way to address a possible shortage of data is to use cross-validation.

5.2.1 Cross-validation

If we have only limited amounts of data at our disposal, setting aside parts of that data as a validation set might be considered too costly — we would like to be able to use all data available for training. *Cross-validation* [Stone, 1974, Bishop, 1995] allows us to do just this, at the expense of increased amounts of computation. The first step is to divide the data set into S disjoint, equally sized, subsets[2]. We set aside one of those subsets as a validation set and train the model on the union of the remaining $S-1$ subsets and once trained we evaluate its performance using the validation set. This procedure is repeated another $S-1$ times, every time using a different subset as validation set, as illustrated in figure 5.1. In the end, we have S validation error measurements and by averaging over these, we get the S-fold cross-validation error.

As we increase S, our confidence in the obtained error measure increases, since the trained models have been trained using larger amounts of training data. Obviously though, the amount of computation required also increases with S, so there must be a judged trade-off between the confidence we require

[2] Actually, the subsets need not be equally sized, but normally they are chosen to be of roughly equal size; when the sets differ in size, this should be corrected for when averaging the validation errors.

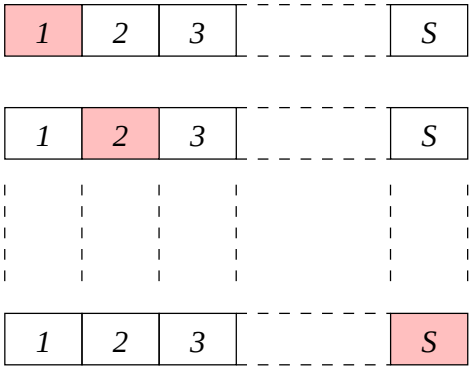


Figure 5.1: A pictorial illustration of cross-validation. Each row correspond to a different division of the data into validation set (shaded) and training set (others).

and the computational effort we can afford. Once we have found the parameter values that give optimum performance on independent (validation) data, we can re-train our model on all the data, using these values. For models where the dependency of the objective function on the adjustable parameters is non-quadratic, the use of cross-validation becomes somewhat questionable, since models trained on different fractions of the data may converge to very different local maxima[3]. It is then not clear that averaging these different likelihood scores will actually tell us anything about the expected performance of models with the corresponding parameter setting.

5.3 A Bayesian approach

We now return to the Bayesian methods which were briefly discussed in the previous section. Our primary objective will be to derive an alternative method for estimating suitable values for the hyper-parameters α, β and σ, but Bayesian methods can also be applied for discriminating between different models in a wider sense, and could therefore be used e.g. to select the number of basis functions. The Bayesian methodology was introduced in the field of neural computing by MacKay [1991, 1992]. The presentation in this section largely follows the review of Bayesian methods in Bishop [1995] (sections 10.1 and 10.4).

So far, we have regarded the training algorithm for the GTM as a maximization procedure, aimed at finding the 'best' single matrix of weights. Taking the Bayesian perspective, it instead becomes a part in a machinery for statistical inference, which produces a distribution over possible weight matrices. This distribution will depend on the data we use for training, and so we write it $p(\mathbf{W}|\mathbf{T})$, which, using Bayes' theorem can be expressed as

$$p(\mathbf{w}|\mathbf{T}) = \frac{p(\mathbf{T}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{T})}, \tag{5.1}$$

where $\mathbf{w}$ denotes a vector of all the elements in $\mathbf{W}$. $p(\mathbf{T}|\mathbf{w})$ is the likelihood for $\mathbf{w}$, in this context sometimes also called the *evidence*, and defines a probability distribution over the space of the data set $\mathbf{T}$, conditioned on $\mathbf{w}$. $p(\mathbf{w})$ is the prior distribution over the weights, before having seen any data, and $p(\mathbf{T})$ is a normalization constant that ensures that the posterior distribution over the weights

[3]Note that this is the case for *all* the parameters in the GTM; the dependency of ℓ on $\mathbf{W}$ is only quadratic given the temporarily fixed responsibilities (which depend on $\mathbf{W}$).

integrates to one,

$$p(\mathbf{T}) = \int p(\mathbf{T}|\mathbf{w})p(\mathbf{w})\, d\mathbf{w}. \tag{5.2}$$

From chapter 3, we know that the density function defined in the data space is a Gaussian mixture with isotropic components; furthermore, one of our fundamental assumptions is that the data sets we use for training consists of independently drawn points. Thus we can write

$$p(\mathbf{T}|\mathbf{w}) = \frac{1}{Z_{\mathbf{T}}} \prod_n^N \sum_k^K \exp\left\{-\frac{\beta}{2}\|\mathbf{t}_n - \mathbf{y}(\mathbf{x}_k, \mathbf{w})\|^2\right\} \tag{5.3}$$

$$= \frac{1}{Z_{\mathbf{T}}} \exp\{-S_{\mathbf{T}}(\mathbf{w}, \beta)\}, \tag{5.4}$$

where

$$S_{\mathbf{T}}(\mathbf{w}, \beta) = -\sum_n^N \ln \sum_k^K \exp\left\{-\frac{\beta}{2}\|\mathbf{t}_n - \mathbf{y}(\mathbf{x}_k, \mathbf{w})\|^2\right\}. \tag{5.5}$$

The reason for introducing the form in (5.4) will soon be clear. From (5.3), we can calculate the normalization constant, $Z_{\mathbf{T}}$, as

$$\begin{aligned} Z_{\mathbf{T}} &= \int \prod_n^N \sum_k^K \exp\left\{-\frac{\beta}{2}\|\mathbf{t}_n - \mathbf{y}(\mathbf{x}_k, \mathbf{w})\|^2\right\} d\mathbf{T} \\ &= \int \sum_{k_1}^K \cdots \sum_{k_N}^K \prod_n^N \exp\left\{-\frac{\beta}{2}\|\mathbf{t}_n - \mathbf{y}(\mathbf{x}_{k_n}, \mathbf{w})\|^2\right\} d\mathbf{T} \\ &= \sum_{k_1}^K \cdots \sum_{k_N}^K \int \exp\left\{-\sum_n^N \frac{\beta}{2}\|\mathbf{t}_n - \mathbf{y}(\mathbf{x}_{k_n}, \mathbf{w})\|^2\right\} d\mathbf{T} \\ &= K^N \left(\frac{2\pi}{\beta}\right)^{ND/2} \end{aligned} \tag{5.6}$$

where $d\mathbf{T}$ stands for $d\mathbf{t}_1 d\mathbf{t}_2 \ldots d\mathbf{t}_N$. After having written the product of sums as a sum of products we use the fact the exponential is strictly positive, in order to swap the order of integration and summation. This gives us a sum of K^N ND-dimensional, independent Gaussian distributions, yielding (5.6) [see e.g. Bishop, 1995, appendix B]. For the time being, we assume that β is fixed at a known value.

Our choice of prior distribution over the weights, $p(\mathbf{w})$, should reflect any prior knowledge we might have regarding the distribution of the data we are trying to model. Most of the time, we have little such prior knowledge, but we normally assume that the mapping from latent to data space should be smooth. As discussed above, the most important parameter for controlling the smoothness of the mapping is σ, but because of finite-size effects, the magnitude of the weights should be constrained, to maintain the smoothness imposed by σ. Here we follow MacKay [1992] and use a spherical Gaussian which, as well as constraining the weights as desired, has favourable analytical properties; thus

$$p(\mathbf{w}) = \frac{1}{Z_{\mathbf{W}}} \exp\{-S_{\mathbf{w}}(\mathbf{w}, \alpha)\}, \tag{5.7}$$

where

$$S_{\mathbf{w}}(\mathbf{w}, \alpha) = \frac{\alpha}{2} \sum_i^W w_i^2 \tag{5.8}$$

and hence

$$Z_{\mathbf{w}} = \int p(\mathbf{w})\, d\mathbf{w} = \left(\frac{2\pi}{\alpha}\right)^{W/2}. \tag{5.9}$$

Just as for β, we will assume for now that we know the value for α.

Since the denominator in (5.1) is independent of $\mathbf{w}$, we see from (5.4), (5.5), (5.7) and (5.8) that finding the mode of $p(\mathbf{w}|\mathbf{T})$ corresponds to the maximization of the regularized log-likelihood function, as described in the previous chapter. However, if we want to make use of $p(\mathbf{w}|\mathbf{T})$ for further statistical inference (such as inferring the distributions of α, β and σ), we must also compute the normalization constant $p(\mathbf{T})$. Unfortunately, the integration in (5.2) is not analytically tractable, so in order to make progress we must make some approximations. Again we will follow MacKay [1992] and approximate $p(\mathbf{w}|\mathbf{T})$ with a Gaussian distribution, which makes it easy to integrate. Some justification for this approximation with the GTM can be found in the fact that if for each data point, there was a single mixture component taking all the responsibility for that data point, this approximation would be exact. It is commonly the case for trained GTM models, that almost all the responsibility for a single data point rests with a single mixture component, although a counter-example was shown in figure 3.8.

To obtain the Gaussian approximation we first note that the maximization of the regularized log-likelihood is equivalent to minimizing the error function

$$S(\mathbf{w}, \beta, \alpha) = S_{\mathbf{T}}(\mathbf{w}, \beta) + S_{\mathbf{w}}(\mathbf{w}, \alpha). \tag{5.10}$$

From (5.8) we see that $S_{\mathbf{w}}$ is quadratic in $\mathbf{w}$, while $S_{\mathbf{T}}$, defined in (5.5), will also be approximately quadratic in $\mathbf{w}$, if we assume that, for each data point, $\mathbf{t}_n$, the sum over k is dominated by a single term; this would consequently result in the corresponding mixture component taking all the responsibility for that data point, as discussed above. We therefore approximate $S(\mathbf{w}, \beta, \alpha)$ by its second order Taylor expansion in $\mathbf{w}$, around its minimum, $\mathbf{w}_{\mathrm{MP}}$, yielding

$$S(\mathbf{w}_{\mathrm{MP}}, \beta, \alpha) + \frac{1}{2}\Delta\mathbf{w}^{\mathrm{T}}\mathbf{H}\Delta\mathbf{w},$$

where $\Delta\mathbf{w} = \mathbf{w}_{\mathrm{MP}} - \mathbf{w}$ and $\mathbf{H}$ is the matrix of second derivatives,

$$H_{ij} = \left.\frac{\partial^2 S(\mathbf{w}, \beta, \alpha)}{\partial w_i \partial w_j}\right|_{\mathbf{w}_{\mathrm{MP}}},$$

also known as the Hessian matrix. The linear term of the expansion vanishes since we are expanding around a minimum.

The Hessian is the the sum of two matrices, $\mathbf{H}_{\mathbf{T}}$ and $\mathbf{H}_{\mathbf{w}}$, resulting from $S_{\mathbf{T}}$ and $S_{\mathbf{w}}$, respectively. It is easily seen from (5.8) that

$$\mathbf{H}_{\mathbf{w}} = \alpha\mathbf{I}, \tag{5.11}$$

where $\mathbf{I}$ is a $W \times W$ identity matrix. In appendix A, we derive two formulae for $\mathbf{H}_{\mathbf{T}}$: one which is exact but computationally rather expensive, while the second is an approximation which is cheap to compute. We then get the Gaussian approximation as

$$p(\mathbf{w}|\mathbf{T}) = \frac{1}{Z}\exp\left\{-S(\mathbf{w}_{\mathrm{MP}}, \beta, \alpha) - \frac{1}{2}\Delta\mathbf{w}^{\mathrm{T}}\mathbf{H}\Delta\mathbf{w}\right\}, \tag{5.12}$$

where the normalization constant can be evaluated [Bishop, 1995, appendix B] as

$$Z = \exp\{-S(\mathbf{w}_{\mathrm{MP}}, \beta, \alpha)\}(2\pi)^{W/2}|\mathbf{H}|^{-1/2}. \tag{5.13}$$

5.3.1 Estimation of α, β and σ

In (5.1) we omitted the dependencies on α, β and σ, and in the discussion that followed α and β were assumed to be known wherever they appeared. We now make these dependencies explicit and re-write (5.1) as

$$p(\mathbf{w}|\mathbf{T},\alpha,\beta,\sigma) = \frac{p(\mathbf{T}|\mathbf{w},\beta,\sigma)p(\mathbf{w},\alpha)}{p(\mathbf{T}|\alpha,\beta,\sigma)}. \tag{5.14}$$

Here we have used the fact that the evidence factor is independent of α, while the prior is independent of β and σ. Normally, we will only have a very vague idea about what values that would be suitable for α, β and σ, and the correct way of treating such unknown parameters in a Bayesian framework is to integrate them out, so that

$$p(\mathbf{w}|\mathbf{T}) = \iiint p(\mathbf{w}|\mathbf{T},\alpha,\beta,\sigma)p(\alpha,\beta,\sigma|\mathbf{T})\,d\alpha\,d\beta\,d\sigma.$$

We therefore seek an expression for $p(\alpha,\beta,\sigma|\mathbf{T})$ and using Bayes' theorem we get

$$p(\alpha,\beta,\sigma|\mathbf{T}) = \frac{p(\mathbf{T}|\alpha,\beta,\sigma)p(\alpha,\beta,\sigma)}{p(\mathbf{T})}. \tag{5.15}$$

Here the normalization constant from (5.14) plays the role of the evidence factor and again we must specify a prior, this time for α, β and σ; computing the the normalization constant, $p(\mathbf{T})$, now involves integration over α, β and σ.

As with $p(\mathbf{w}|\mathbf{T})$ in equation (5.1), finding the mode of $p(\alpha,\beta,\sigma|\mathbf{T})$ only involves the prior and the evidence factors. Therefore, one approach would be to try to find the mode, corresponding to the most probable values for α, β and σ, and then use these values. This can be motivated by an assumption that $p(\alpha,\beta,\sigma|\mathbf{T})$ is sharply peaked around the mode, so that

$$\begin{aligned}
p(\mathbf{w}|\mathbf{T}) &= \iiint p(\mathbf{w}|\mathbf{T},\beta,\sigma)p(\mathbf{w},\alpha)p(\alpha,\beta,\sigma|\mathbf{T})\,d\alpha\,d\beta\,d\sigma \\
&\approx p(\mathbf{w}|\mathbf{T},\beta_{\mathrm{MP}},\sigma_{\mathrm{MP}})p(\mathbf{w},\alpha_{\mathrm{MP}}) \iiint p(\alpha,\beta,\sigma|\mathbf{T})\,d\alpha\,d\beta\,d\sigma \\
&= p(\mathbf{w}|\mathbf{T},\beta_{\mathrm{MP}},\sigma_{\mathrm{MP}})p(\mathbf{w},\alpha_{\mathrm{MP}})
\end{aligned}$$

If we take $p(\alpha,\beta,\sigma)$ to be uniform[4] on the positive region of $\Re^3$, finding the mode of $p(\alpha,\beta,\sigma|\mathbf{T})$ will correspond to maximising the evidence factor, $p(\mathbf{T}|\alpha,\beta,\sigma)$, which we can re-write in terms of quantities we have already evaluated,

$$\begin{aligned}
p(\mathbf{T}|\alpha,\beta,\sigma) &= \int p(\mathbf{T}|\mathbf{w},\beta,\sigma)p(\mathbf{w},\alpha)\,d\mathbf{w} \\
&= \frac{1}{Z_{\mathbf{T}}}\frac{1}{Z_{\mathbf{w}}}\int \exp\{-S(\mathbf{w},\beta,\alpha,\sigma)\}\,d\mathbf{w} \\
&= \frac{Z}{Z_{\mathbf{T}}Z_{\mathbf{w}}}
\end{aligned} \tag{5.16}$$

From (5.16), (5.13), (5.10), (5.9) and (5.6), we can write the logarithm of the evidence for α, β and σ as

$$\begin{aligned}
\ln p(\mathbf{T}|\alpha,\beta,\sigma) = &-S_{\mathbf{T}}(\mathbf{w}_{\mathrm{MP}},\beta,\sigma) - S_{\mathbf{w}}(\mathbf{w}_{\mathrm{MP}},\alpha) - \frac{1}{2}\ln(|\mathbf{H}|) \\
&+ \frac{W}{2}\ln(\alpha) + \frac{ND}{2}\ln(\beta) - \frac{ND}{2}\ln(2\pi) - N\ln(K).
\end{aligned} \tag{5.17}$$

[4]Such a prior is called an *improper* prior [Bishop, 1995], since it cannot be normalized. This would cause difficulties if we wanted to compute $p(\mathbf{T})$ in (5.15).

One obvious approach for finding the mode of $p(\alpha, \beta, \sigma|\mathbf{T})$, is simply to evaluate (5.17) over a grid of points in α-β-σ-space. Although such a simplistic approach will be computationally demanding, it may still be more efficient than cross-validation, and may also give clearer results. However, an almost certainly more efficient approach would be to incorporate the parameter estimation as part of the training algorithm, by maximising (5.17) with respect to α, β and σ during training.

Online estimation of α, β and σ

We first consider maximization of (5.17) with respect to α, and so we want to evaluate the corresponding derivative. We start by re-writing the term involving the Hessian as

$$\ln |\mathbf{H}| = \ln \prod_i^W (\lambda_i + \alpha) = \sum_i^W \ln(\lambda_i + \alpha), \tag{5.18}$$

where λ_i are the eigenvalues of $\mathbf{H_T}$. From (5.17) and (5.8), we then get

$$\frac{d \ln p(\mathbf{T}|\alpha, \beta, \sigma)}{d\alpha} = -\frac{1}{2}\sum_i^W w_i^2 + \frac{W}{2}\frac{1}{\alpha} - \frac{1}{2}\sum_i^W \frac{1}{\lambda_i + \alpha}, \tag{5.19}$$

which we can set to zero and then solve for α, yielding

$$\alpha = \frac{\gamma}{\sum_i^W w_i^2}, \tag{5.20}$$

where

$$\gamma = \sum_i^W \frac{\lambda_i}{\lambda_i + \alpha}. \tag{5.21}$$

If we assume that all λ_i are positive, the terms of this sum lies between 0 and 1, and the terms where $\alpha \ll \lambda_i$ will dominate. These terms correspond to directions in the weight space where the weights are relatively tightly constrained by the data, so γ can be interpreted as the number of *well-determined* weights [Gull, 1989, Bishop, 1995]. The result in (5.20) is actually only approximate, since it has been derived under the implicit assumption that the eigenvalues λ_i are independent of α, which generally is not true, since $\mathbf{H}$ is evaluated at $\mathbf{w}_{\mathrm{MP}}$, which depends on α.

Next, we turn our attention to β and now we must decide which of two forms of $\mathbf{H_T}$ we choose to work with. For the online estimation, we choose the approximate form, which is derived in appendix A as a block-diagonal matrix with identical building blocks

$$\beta \mathbf{\Phi}^{\mathrm{T}} \mathbf{G} \mathbf{\Phi}, \tag{A.11}$$

where $\mathbf{G}$ is defined in (3.14). This decision based on the following grounds:

- the exact form of $\mathbf{H_T}$, given in (A.13) is expensive to compute, which makes it unattractive to use for online parameter estimation; the approximate form we compute anyway, as a step in the normal training algorithm,
- as can be seen from (A.11), the approximate form of $\mathbf{H_T}$ depends linearly on β, which allows for an update formula for β in closed form. This would not be the case if we choose to work with the exact form (A.13), and hence we would be forced to either make other approximations (e.g. assume that the dependency of the exact $\mathbf{H_T}$ on β is approximately linear) or use costly numerical optimization to update β.

Using the approximate form of the Hessian we get

$$\frac{d\lambda_i}{d\beta} = \frac{\lambda_i}{\beta} \tag{5.22}$$

and hence, using (5.18),

$$\frac{d \ln |\mathbf{H}|}{d\beta} = \frac{1}{\beta} \sum_i^W \frac{\lambda_i}{\lambda_i + \alpha}, \tag{5.23}$$

and so, setting the derivatives of (5.17) with respect to β to zero, using (3.12) and (5.23) we get

$$\beta = \frac{ND - \gamma}{\sum_{n,k}^{N,K} r_{kn} \|\mathbf{t}_n - \mathbf{y}_k\|^2}. \tag{5.24}$$

When γ is zero this equals the EM-update for β, which would correspond to a case where no weights are well-determined by the data. This uncertainty in the weight could then be taken to account for some of the discrepancy between the data and the model. As γ increases, however, an increasing fraction of any remaining deviation must be attributed to inherent noise on the data, as reflected by a decreasing β.

For σ we are unfortunately unable to derive as elegant a solution, but since we are now down to a single variable, we can afford searching over a grid of σ-values, evaluating (5.17) at each point[5], after having trained the GTM model, re-estimating α and β online.

For the practical implementation, we follow the approach taken by MacKay [1992] in applying Bayesian techniques to feed-forward neural networks, periodically re-estimating α and β during the training of $\mathbf{W}$. More precisely, we take a three-level approach, as follows:

for $i = 1$ to I **do**
  initialize GTM using σ_i
  repeat
    repeat
      optimize $\mathbf{W}$ by EM, with α and β kept fixed
    until stop criterion $\mathbf{W}$ for is met
    re-estimate α and β, using (5.20) and (5.24), respectively
  until stop criterion for α and β is met
  record the log-evidence for σ_i
end for

The stop criteria are typically chosen to be a threshold for the change in log-likelihood, combined with a maximum number of iterations allowed at each level. After we have found the value of σ which gives the highest log-evidence, we simply use that to train our model, again estimating α and β online. By adopting this hierarchical scheme, first optimizing with respect to $\mathbf{W}$, we hope that the approximations we have used to derive the update formulae for α and β will be reasonable by the time we start to apply them. This assumes that $\mathbf{W}$ will then be close to its optimum — the mean of the posterior of $\mathbf{W}$ — given the current values for α, β and σ, and that most of the responsibility for any data point is assigned to a single mixture component.

5.4 An Experimental evaluation

In this section we investigate empirically the selection of parameters by cross-validation and offline and online Bayesian methods. The data used was generated from a 20×20 regular grid of points on

[5]Numerical maximization of (5.17) with respect to σ was also considered, but was empirically found to be unacceptably inefficient.

a curved 2-D surface in a 3-D space, generated by the function

$$z = (1.5x^3 - x) + 0.25\cos(2y).$$

x and y had a range of $[-1, 1]$ and $[-2, 2]$, respectively; giving one of the variables significantly larger range was a deliberate choice that ensured that the PCA initialization would provide a reasonably good starting point for the GTM models to be trained. 20 training data sets were generated by adding random spherical Gaussian noise with standard deviation 0.2 (corresponding to $\beta = 0.2^{-2} = 25$) to the grid points. A sample data set is shown, together with the data generating manifold, in the top-left panel of figure 5.5. A separate test data set was created from a 32×32 grid on the same surface, again adding random noise.

5.4.1 Offline estimation of α, β and σ

For each of the 20 training data sets, a GTM with a 15×15 grid of latent points and a 5×5 grid basis functions was initialized using PCA, trained with α and β kept fixed and then evaluated using

- log-likelihood of the training set, measured by 10-fold cross-validation,
- the log-evidence of the training set, (5.17), using either
  - the exact (given by (A.13) and (5.11)) or
  - the approximate (given by (A.11) and (5.11))

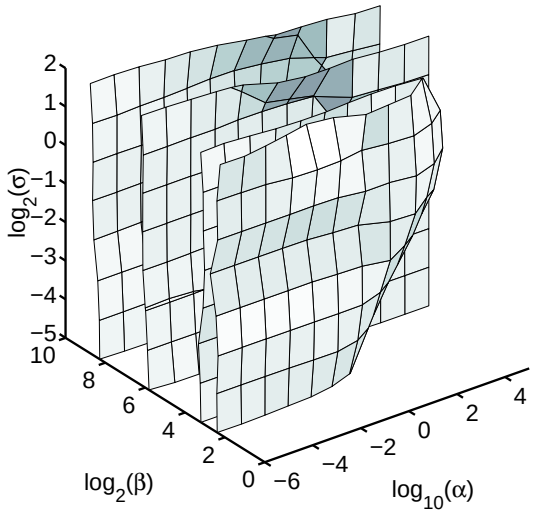
  form of the Hessian, and
- log-likelihood of the test set.

This procedure was repeated for all possible combinations of

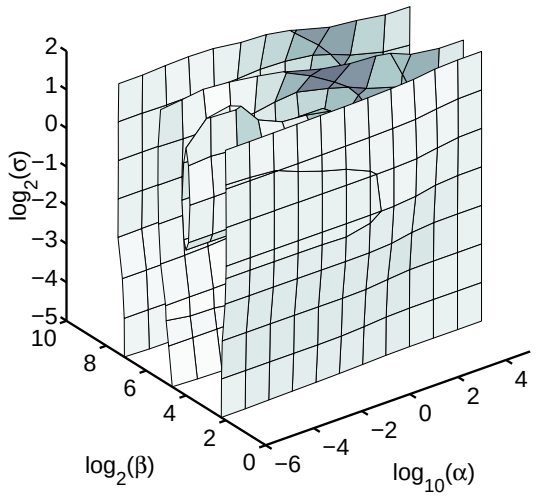
- $\alpha = 10^i$, $i = -6, -5, \ldots, 5$,
- $\beta = 2^j$, $j = 0, 1, \ldots, 10$, and
- $\sigma = 2^k$, $k = -5, -4, \ldots, 2$.

For α, this range was assumed to be sufficient and the empirical results supports this. For β, the lower limit was given by the variance in the data, whereas the upper limit was assumed to be high enough. The limits of σ were given by the fact that for smaller or larger values, the matrix of activations of the given basis functions, $\mathbf{\Phi}$, became rank deficient and hence deteriorated towards a PCA solution. As will be seen, the empirical evidence suggests that also these limits were sufficiently wide.

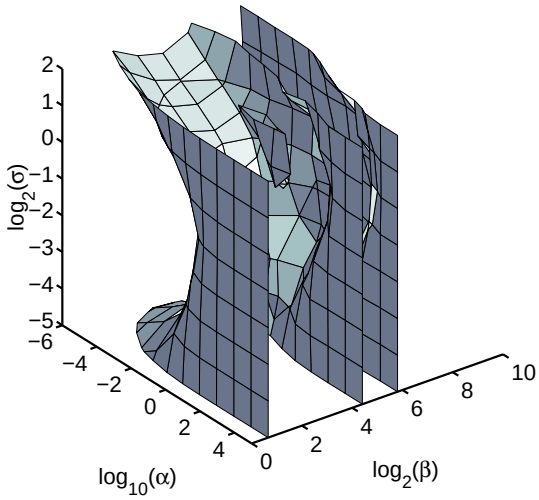
Figure 5.2 show surfaces of constant log-likelihood on validation and test data, and constant log-evidence computed using the exact and the approximate form of the Hessian. The log-likelihood score computed by cross-validation appears to be very flat around the maximum in the α-σ-plane, and selects rather narrow basis functions combined with a higher degree of weight regularization and greater noise variance. The log-likelihood score computed over the test set prefers the least regularized configuration, whereas the log-evidence computed using the approximate form of the Hessian choose the highest degree of weight regularization. The log-evidence scores appears to shun low-α regions, except at the extremes of σ, where the $\mathbf{\Phi}$ matrix tends towards rank deficiency — this tendency does not show for the log-likelihood scores. Figure 5.3 give an alternative view of the results, including histogram-indicators of the maxima found for the different data sets. The flatness of the log-likelihood score computed by cross-validation is reflected the large spread of the maxima found in the α-σ-plane.



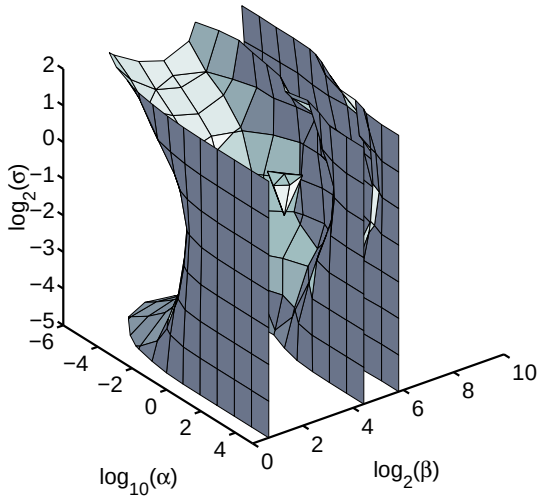
(a) Log-likelihood by cross-validation. The 'closest', folded surface represents a log-likelihood of -3.20; the maximum value observed was -3.15, found at $\langle \alpha, \beta, \sigma \rangle = \langle 10^2, 2^3, 2^{-2} \rangle$. The two surfaces behind represent log-likelihoods of -5 and -20.



(b) Log-likelihood of test data. The innermost surface, or shell, represents a log-likelihood of -2.70; the maximum value observed was -2.67, found at $\langle \alpha, \beta, \sigma \rangle = \langle 10^{-1}, 2^4, 2^0 \rangle$. The two surfaces on each side are part of the same surface, representing the log-likelihood value -3, and the farthest surface represents a log-likelihood of -6.



(c) 'Approximate' log-evidence. The small, top, clove-looking shell represent a log-evidence of -1195; the maximum value observed was -1184, found at $\langle \alpha, \beta, \sigma \rangle = \langle 10^1, 2^3, 2^2 \rangle$. The nearest surrounding surface represents a log-evidence of -1400 and the furthermost surface, a value of -2200.



(d) 'Exact' log-evidence. The innermost, tetrahedron-looking shell represent a log-evidence of -1170; the maximum value observed was -1159, found at $\langle \alpha, \beta, \sigma \rangle = \langle 10^1, 2^4, 2^0 \rangle$. The nearest surrounding surface represents a log-evidence of -1400 and the furthermost surface, a value of -2200.

Figure 5.2: The surfaces are computed from averaged observations of the 20 different training sets. The log-likelihood scores have been normalized by the number of data points. Note that the log-evidence plots are rotated 90° relative to the log-likelihood plots, as this was found to give the best view of these results.

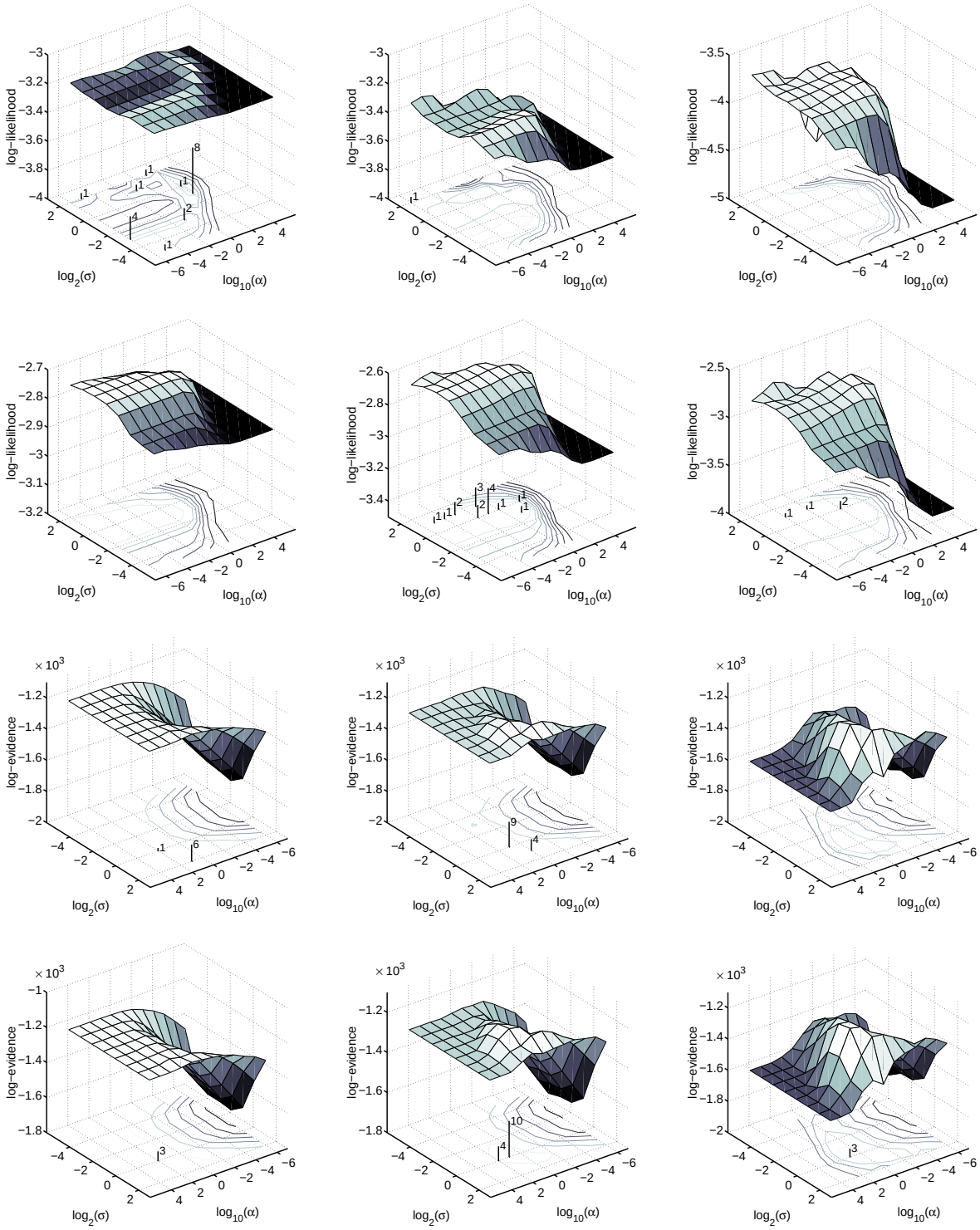


Figure 5.3: The surface and contour plots show the average log-likelihood or log-evidence, computed over the 20 data sets, plotted against $\log_{10}(\alpha)$ and $\log_2(\sigma)$. Rows 1 to 4 correspond to log-likelihood on validation data (1), ditto on test data (2), and log-evidence computed using the approximate (3) and exact (4) form of the Hessian. Columns 1, 2 and 3 correspond to β-values of 8, 16 and 32. The plots also contain histograms with numerical labels of the optimal α-β-σ-combinations found for the 20 data sets. Note that the plots of the log-evidence are rotated 180° relative to the log-likelihood plots.

5.4.2 Online estimation of α and β

Essentially the same set-up was used to evaluate the methods for online estimation, except that α and β were set to initial values and then were re-estimated during training. Figure 5.4 shows the resulting plots of log-evidence, log-likelihood, α and β, plotted against $\log_2 \sigma$. As can be seen, the results agree reasonably well with those obtained in the offline estimation experiments described above.

Figure 5.5 shows an example of a model selected using this 'semi-online' procedure ($\sigma = 1.0$), together with the generating manifold and the sample data set used for training, and examples of under- and over-fitting models.

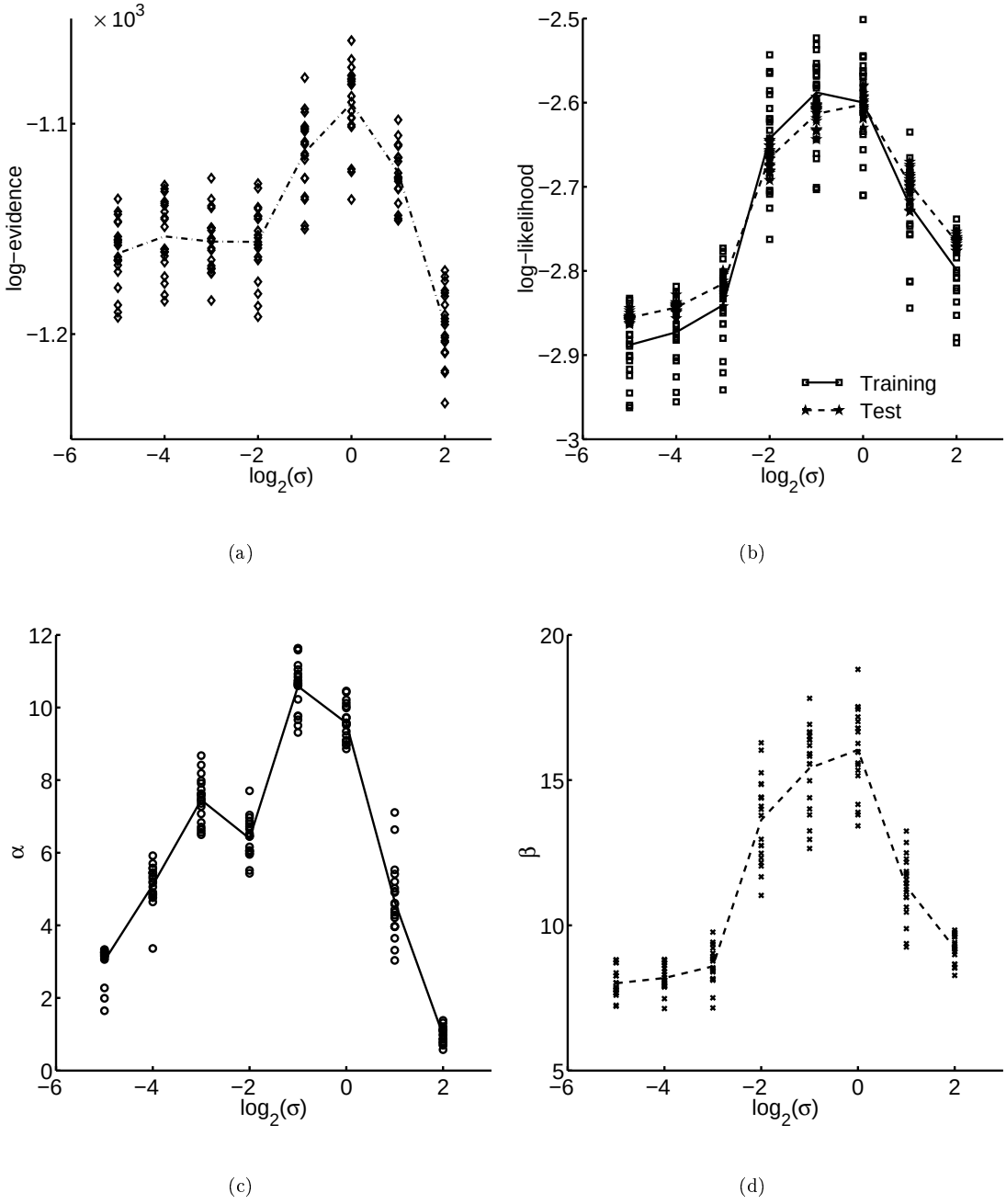
5.5 Discussion

The experimental results in the previous section suggest that all methods that have been considered can be used for parameter selection, although both log-likelihood computed by cross-validation and the log-evidence computed offline using the approximate form of the Hessian over-estimate the noise level rather significantly. From a practical point of view, online estimation of α and β combined with grid search in σ-space appear to be the most favourable alternative, requiring only a fraction of the computational effort for grid search in α-β-σ-space.

Taking a Bayesian perspective on the GTM, we no longer have one single manifold in the data space, but rather a distribution of manifolds, obtained by integrating over the posterior distribution over all parameters in the GTM ($\mathbf{W}$, α, β and σ). How should such a model be used for visualisation of data? There is no obvious answer to this question, but possible approaches are to use the manifolds corresponding to the mean or the mode of the joint distribution over parameters (these would be identical using the approximate Bayesian method described in this chapter). There are also more difficult questions that arise, which we so far have not addressed. One such problem is multi-modality — when we are using the Gaussian approximation of the posterior distribution in the weight space, we can only expect this to be true 'locally'. If we are using symmetrical grids for the latent points and the centres of the basis functions, we know that there are identical modes in the weight space, corresponding to different rotations and flips of the manifold. Moreover, we know that the EM-algorithm may find a local, rather than the global minima, and different parameter settings and different initializations may result in different local minima. However, if we assume that these different minima are sufficiently distant from each other in the weight space, we can still hope that using the Gaussian approximation should allow us to find values for α, β and σ, appropriate for the particular mode under consideration.

A possible solution to the problem of multiple modes of the posterior weight distribution would be to fit a Gaussian at each mode and then form a weighted combination of these models, which also can be carried out within the Bayesian framework. However, this has important implications for how we use the GTM; again, how do we use such a mixture of weights in visualization? As an example, consider the posterior mean projection of a data point for a GTM with a symmetrically aligned, square, 2-D latent space; if we combine the four modes corresponding to the four rotations of the manifold, which obviously will fit the data equally well and hence should carry equal weight, we end up with a point in the centre of the latent space, and this is going to be the result regardless of the location of the data point.

A different approach, that avoids using a Gaussian approximation for the posterior distribution of the weights, is to evaluate the necessary integrals numerically by using Monte-Carlo methods. However, also then ways of dealing with multi-modality and symmetries must be addressed, if the



(a) (b)

(c) (d)

Figure 5.4: The plots shows values after training for the log-evidence, 5.4(a), log-likelihood for training and test set, 5.4(b), and the estimates of α, 5.4(c), and β, 5.4(d), plotted against $\log_2(\sigma)$. Each plot shows the results for the 20 individual training sets, together with a line showing the mean.

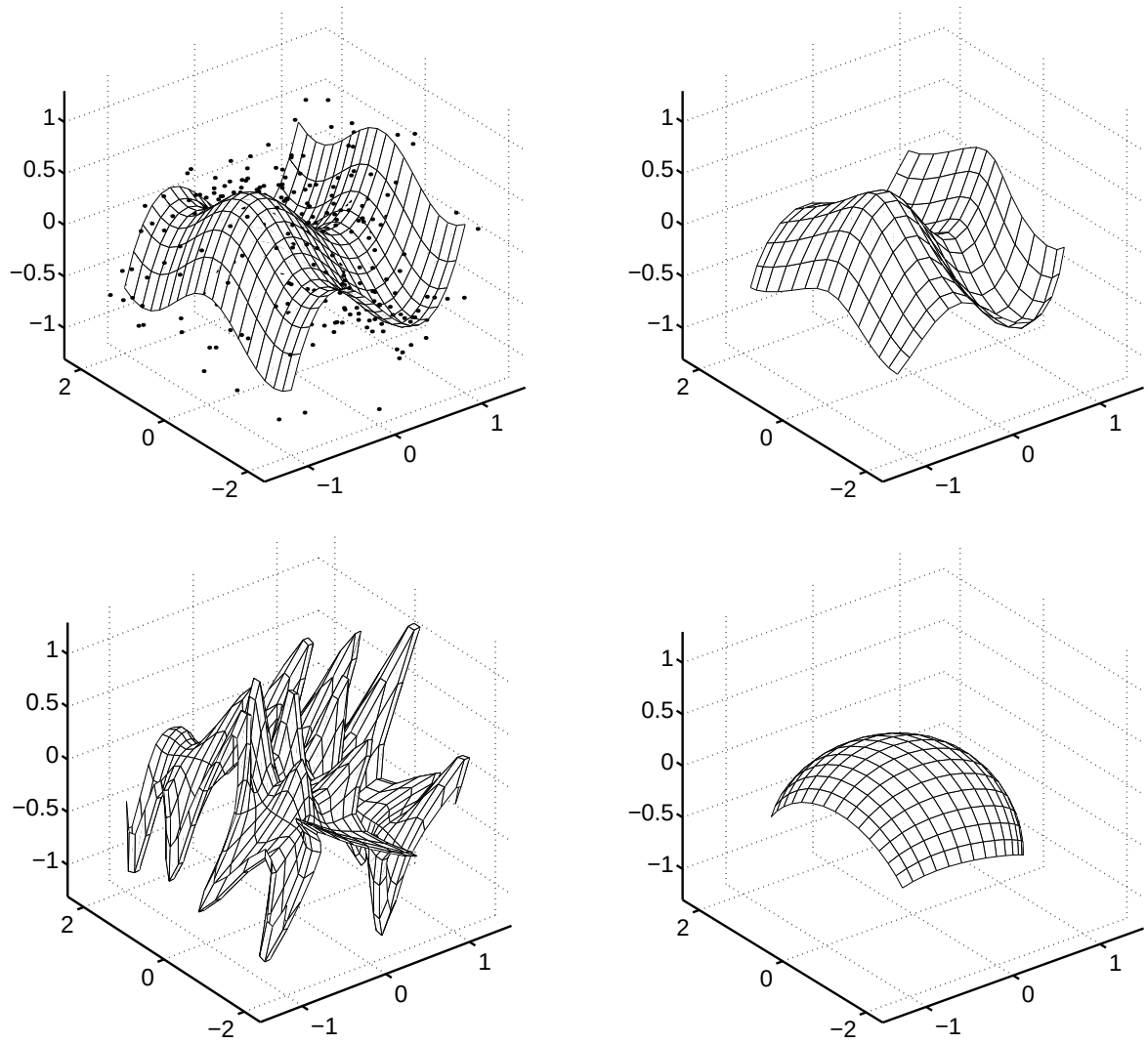


Figure 5.5: The top left plot shows the data generating manifold, together with a sample data set, plotted as $\cdot$. The top right plot shows the manifold of a GTM model trained on this data set, with σ fixed to 1 and α and β being re-estimated during training, final values being $\alpha = 9.2$ and $\beta = 18.3$. The bottom left plot shows a significantly more flexible model, $\sigma = 0.25$, trained using the standard GTM algorithm and no weight regularization; the final estimated value for β was 40.9. Note that this plot was produced using a finer grid in the latent space. The bottom right plot, shows a much stiffer model, $\sigma = 2$, trained using the standard GTM algorithm and constant weight regularization of 50; the final estimated value for β was 10.1.

resulting GTM model is to be used for visualization.

In principle, Bayesian methods could also be used for selecting other model parameters, such as the number of basis functions and the number of latent points, but as the number of parameters increase, grid search methods quickly becomes computationally infeasible. An alternative approach to implementing the mapping from latent to data space that eliminates the basis functions is discussed in section 6.9.

The number of latent points

A parameter that we have only briefly touched upon in the preceding discussion is the number of points on the grid in the latent space. If this grid is intended to approximate a continuous, uniform distribution, we obviously would want it to be as dense as possible, and in principle there is nothing preventing us from using a very dense grid. This will result in a very large mixture of Gaussians, measured by the number of components, but the mixture is constrained and the number of degrees of freedom in the model depends on the number of adjustable parameters, which is independent of the number of latent points. In practice, however, using large grids in the latent space is computationally prohibitive, both in terms of speed and memory usage, and so we must make a judged trade-off between the computational effort we can afford and the 'resolution' in the latent space.

Chapter 6

Extensions and Future Directions

This chapter discusses a number of possible extensions of the basic GTM model described in chapter 3, some of which have been discussed in Bishop et al. [1998a], and which may become subjects of future research. For some cases, preliminary work has already been done, whereas others are currently only proposals. They all highlight the advantages of having chosen a model that fits into the framework of probability theory.

6.1 Relaxations in the GTM model

There are several constraints governing the Gaussian mixture generated in the data space under the GTM model, when compared to a general Gaussian mixture. Apart from the constraints imposed on the centres by the latent variable construction, the basic GTM model uses an isotropic noise model with the noise level being equal for all components. Moreover, the mixing coefficients of the mixture are kept fixed and equal $(1/K)$. In principle, there is nothing preventing us from simply letting each component have a full covariance matrix of its own, possibly combined with variable mixture coefficients, $\pi_1, \ldots, \pi_K$, such that $\sum_k^K \pi_k = 1$. In practice, however, this would lead to an explosion in the number of parameters and a model with far too much freedom, with associated problems such as overfitting the training data, as discussed in section 5.2.

A more realistic approach, which is also more in the spirit of the GTM, is to allow the variances and the mixing coefficients to be functions of the latent variable $\mathbf{x}$. For β, one way to achieve this would be

$$\beta(\mathbf{x}_k) = \exp\left(\sum_m^M \phi_m(\mathbf{x}_k) w_{m\beta}\right), \tag{6.1}$$

where $\phi_m(\cdot)$ are basis functions that may or may not be identical with the basis functions used to compute the centres. The exponentiation ensures that β is always positive. Similarly, a possible way to compute the mixing coefficients is to use the soft-max function [Bridle, 1990], also called the normalized exponential,

$$\pi(\mathbf{x}_k) = \frac{\exp\left(\sum_m^M \phi_m(\mathbf{x}_k) w_{m\alpha}\right)}{\sum_{k'}^K \exp\left(\sum_m^M \phi_m(\mathbf{x}'_k) w_{m\alpha}\right)}, \tag{6.2}$$

which guarantees that $\pi(\mathbf{x}_k) \in [0,1]$ for all k and $\sum_k^K \pi(\mathbf{x}_k) = 1$. A complication with (6.1) and (6.2) is that we can no longer find update formulae in closed form, but must resort to numerical optimization.

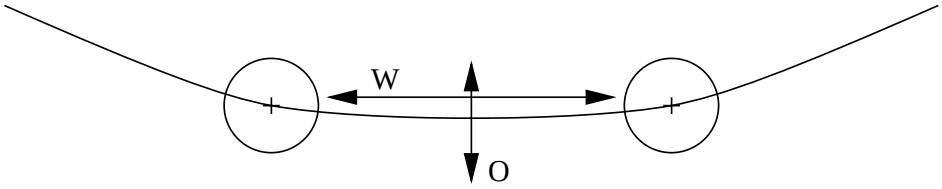


Figure 6.1: The within- and off-manifold variances, labelled W and O respectively, illustrated for a 1-D GTM with the standard isotropic noise model. The curved line represents the manifold, the '+'-signs centres of mixture components, with the surrounding circles representing the noise model.

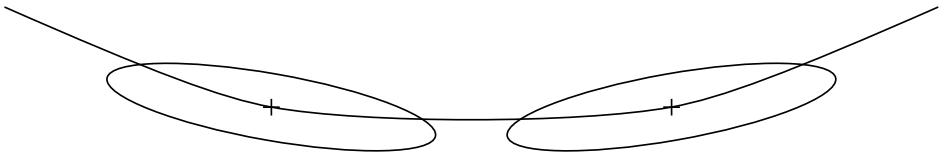


Figure 6.2: Illustration of a manifold-aligned noise model — as in figure 6.1, the curved line represents the manifold, the '+'-signs centres of mixture components, but the circles have been replaced by ellipses aligned with the manifold, illustrating the new noise model.

Assuming we model β and π as scalar functions of the latent variables[1] we can incorporate this information in visualization plots of data, showing (e.g.) how the noise on the data varies between different regions of the data space.

Another issue is whether we should use a spherical or a ellipsoidal, possibly axis-aligned, noise model. Alternative variants which cater for two cases of full covariance matrices are discussed in sections 6.2 and 6.4 below. The use of an isotropic or an independent noise model is what differentiates between probabilistic principal components analysis and factor analysis, respectively. A more general noise model will avoid skewing the structural model (the shape in the manifold, in the case of the GTM) in order to explain noise not catered for by a more restrictive noise model. However, if the noise indeed is approximately isotropic, the spare degrees of freedom provided by the more general model may cause problems associated with overfitting. When we use an isotropic noise model, we implicitly assume that any residual variance has the same scale on all observed variables. If assume that the *fraction* of noise is the same on all observed variables, — i.e. that observed variables with higher variance also are subject a higher level of noise — putting them on a common scale, by normalising them to all have unit variance over the training data, will meet the underlying assumption of the isotropic noise model.

6.2 A Manifold-aligned noise model

A potential problem with the basic GTM is that the noise model may have to take on double roles, of possibly conflicting natures. The noise model is intended to take account of *off-manifold* variance (see figure 6.1), i.e. the fact that data points, because of noise, normally do not lie exactly on the manifold. However, since our prior distribution in the latent space consists of a finite set of points, the noise model may also have to explain the *within-manifold* variance (see figure 6.1), arising from data points that lie close to the manifold, but fall between mixture components.

We would like to avoid this conflict by allowing for greater variance within the manifold, which

[1] This is not necessarily the case for β.

leads us to a Gaussian mixture with ellipsoidal components which are locally aligned with the manifold, as illustrated in figure 6.2. Other models has been proposed along these lines [Williams et al., 1993, Hinton et al., 1995a, Simard et al., 1992], where variance along certain directions in the data space is less penalized than variance in other directions. These models require the use of full covariance matrices for the mixture components, but this does not mean we have to increase the number of parameters in our model. Since we want the within-manifold noise to be locally aligned with the manifold, the corresponding covariance matrix for each component is determined by the derivatives of the mapping $\mathbf{y}(\mathbf{x}, \mathbf{W})$ with respect to the latent variable $\mathbf{x}$, computed at the location of the centre of the corresponding component. That is,

$$\mathbf{C}_k = \beta^{-1}\mathbf{I} + \frac{\upsilon}{L}\sum_{l}^{L} \frac{\partial \mathbf{y}}{\partial x_k^l}\frac{\partial \mathbf{y}}{\partial x_k^l}^{\mathrm{T}},$$

where $\mathbf{C}_k$ denotes the covariance matrix of the kh mixture component, $\partial \mathbf{y}/\partial x_k^l$ is defined in (4.5), υ is a scaling factor equal to some multiple of the distance between neighbouring points in latent space, and we have added the $\beta\mathbf{I}$ term so that the resulting probability distribution does not become singular. We could instead consider letting each component have its own off-manifold noise term, orthogonal to the within-manifold noise, rather than just adding isotropic noise.

This modification of the model requires that we, in the E-step, compute Mahalanobis distances, rather than square distances. Moreover, we generally lose the closed form update of the weights in the M-step, since now also the manifold-aligned part of the covariance matrix depends on the weights. However, the original M-step may still be used as an approximation, since if we assume that the manifold is smooth, a configuration where the centres have their 'right' location will automatically get the noise model approximately correctly aligned. Unfortunately, with this approximate 'M-step' the resulting algorithm is no longer an EM-algorithm and so there is no guarantee that this algorithm will converge to a maximum of the log-likelihood function.

We now return to the toy data set introduced in example 3.1. Figure 6.3 shows the result of trying a modified GTM model with 10 mixture components, which uses the manifold aligned noise just described, on this data, with the exception that we use the M-step of the original training algorithm, ignoring the influence of the covariance matrices. The model is compared to a GTM with the same number of mixture components, but restricted to use the standard, spherical noise model. The plots and the likelihood scores clearly show that the manifold-aligned noise model is superior here. However, when compared with a standard GTM that has 30 mixture components, that is no longer the case. One might think that the use of fewer mixture components should result in computational savings, but unfortunately, at least in this case, these savings are lost in the increased cost of the E-step.

6.3 Mixtures of GTMs

Since the GTM is itself a mixture model, an obvious and straightforward alternative to a single GTM model is to use a mixture of J GTM models. Computation of the mixing coefficients for the GTM mixture, $\hat{\pi}_j$, can easily be incorporated into an EM-algorithm for simultaneously training all GTM models of the mixture, as

$$\hat{\pi}_j = \sum_{n}^{N} \frac{\sum_{k_j}^{K_j} r_{k_j n}}{\sum_{j'}^{J}\sum_{k_{j'}}^{K_{j'}} r_{k_{j'} n}}, j = 1, \ldots, J. \tag{6.3}$$

This says that the posterior probability of the jth GTM model in the mixture equals its share of the total responsibility of the data. The E-step of the training algorithm will involve the whole mixture

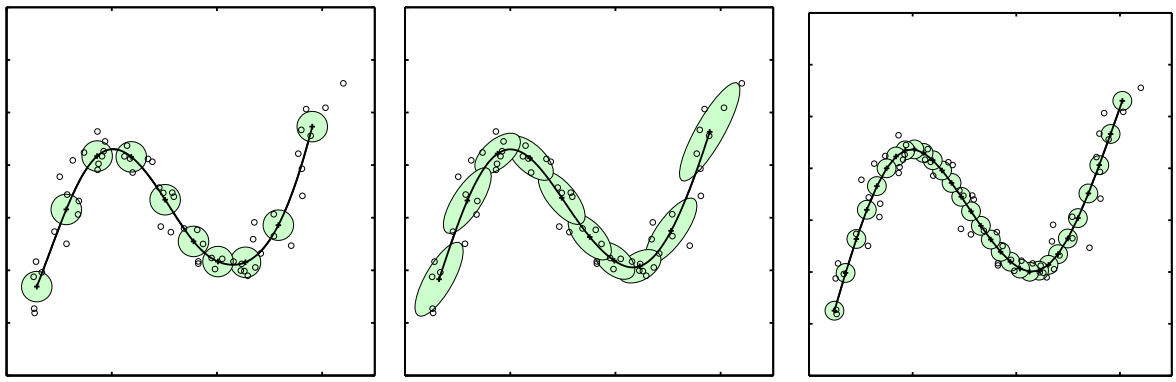


Figure 6.3: The left plot shows a GTM with 10 latent points and a spherical noise model fitted to the data from example 3.1; the log-likelihood for this model after training was -58.9. The middle plot shows a corresponding GTM, but with a manifold aligned noise model; the log-likelihood after training was -48.3. The right plot, finally, shows a GTM with 30 latent points and a spherical noise model fitted to the same data; the log-likelihood after training was -41.0.

of GTM models, but for the M-step, each GTM model can be updated separately, unless we make use of global parameters, e.g. having a single value for β, shared by all GTM models in the mixture.

An example where such a model might be appropriate is the 3-phase pipe flow data introduced in example 3.2. We know that data generated from different flow configurations live on a number of different 2-D manifolds, and the use of a mixture of *linear* models to visualize this data, by Bishop and Tipping [1996], has been shown to be successful.

6.4 A Semi-linear model

Normally, we select the latent space of the GTM to have a low dimensionality — typically, we would choose it to be 2-D. If we want to experiment with higher-dimensional latent spaces, although in principle straightforward, we would soon run into computational difficulties, since the number of latent points would grow exponentially with the number of dimensions. MacKay and Gibbs [1997] address this problem for a density network model by re-sampling the latent space using hybrid Monte-Carlo methods [Neal, 1996, 1992]. This is a potentially useful approach, but it suffers the problem of still being rather demanding in terms of computation.

Here, we instead consider the use of a *semi-linear* model, obtained by combining a GTM model with a probabilistic PCA model (see section 2.2.2). This gives a model where the observed variables depend non-linearly on, say, 2 of the latent variables, while depending linearly on the remaining $L-2$. The 'non-linear' latent variable are treated just like in the GTM model, essential by discretizing the latent space, while for the 'linear', or continuous, latent variables, the posterior distribution over the latent space can be calculated analytically. We have already seen that the basic GTM model can be seen as a constrained mixture of spherical Gaussians; similarly, this semi-linear model can be viewed as constrained mixture of probabilistic principal component analyzers, where the centres of the PPCAs lie in the manifold defined by the non-linear mapping from latent to data space. In contrast to the manifold aligned noised model discussed above, this model will allow greater variance along certain directions *off* the curved manifold. To be more precise, the distribution in the data space would now

be defined as

$$p(\mathbf{t}|\mathbf{W},\mathbf{V},\beta) = \frac{(2\pi)^{-D/2}}{K|\mathbf{C}|^{1/2}} \sum_{k}^{K} \exp\left\{-\frac{1}{2}(\mathbf{t}-\mathbf{y}(\mathbf{x}_k,\mathbf{W}))\mathbf{C}^{-1}(\mathbf{t}-\mathbf{y}(\mathbf{x}_k,\mathbf{W}))^{\mathrm{T}}\right\}, \tag{6.4}$$

where

$$\mathbf{C} = \beta^{-1}\mathbf{I} + \mathbf{V}\mathbf{V}^{\mathrm{T}}$$

and $\mathbf{V}$ is a $D \times q$ matrix that defines the linear mapping from latent to data space, q being the number of continuous latent variables.

Tipping and Bishop [1997a] presents an EM-algorithm for general mixtures of PPCAs, which is easily modified to deal with this constrained case. The E-step differs only in that we must now compute the full Mahalanobis-distance, as indicated in (6.4), while in the M-step, we first update $\mathbf{W}$ using the standard M-step (3.13). We then use the updated weights, $\widetilde{\mathbf{W}}$, to compute the *weighted* covariance matrix,

$$\mathbf{S} = \frac{1}{N}\sum_{n,k}^{N,K} r_{kn}(\mathbf{t}_n - \mathbf{y}(\mathbf{x}_k,\widetilde{\mathbf{W}}))^{\mathrm{T}}(\mathbf{t}_n - \mathbf{y}(\mathbf{x}_k,\widetilde{\mathbf{W}})).$$

Using the results of Tipping and Bishop [1997a], the maximum likelihood solution for $\mathbf{V}$ is given by

$$\widetilde{\mathbf{V}} = \mathbf{U}(\mathbf{\Lambda} - \beta^{-1}\mathbf{I})^{1/2},$$

where $\mathbf{U}$ is the $D \times q$ matrix whose columns is the q principal eigenvectors of $\mathbf{S}$ and $\mathbf{\Lambda}$ is a $q \times q$ diagonal matrix containing the corresponding eigenvalues, λ_d, $d = 1, \ldots, q$. This result corresponds to the traditional way of computing principal components, discussed in section 2.1.1. For β, we get

$$\frac{1}{\tilde{\beta}} = \frac{1}{D-q}\sum_{d=q+1}^{D} \lambda_d,$$

which has the intuitive interpretation as the average variance 'lost' when projecting the D-dimensional data on the q-dimensional subspace defined by the model.

These update formulae for $\mathbf{V}$ and β require computing the covariance matrix $\mathbf{S}$, which can be quite an effort if the dimensionality of the data space, D, is high. As noted by Tipping and Bishop [1997a], a better approach in such situations may be to take the latent variable perspective on PCA and use an EM-algorithm, similar to the one for factor analysis, discussed in section 2.2.1. Although this means using an iterative optimization scheme, the computational cost for each iteration only scales as $\mathcal{O}(ND)$, compared to $\mathcal{O}(ND^2)$ for the computation of $\mathbf{S}$. Thus, provided that the EM-algorithm converges quickly enough, this will be a computationally favourable alternative. The EM-algorithm for PPCA is discussed in detail in Tipping and Bishop [1997b].

To try this model, a toy data set of 400 data points was generated in a 3-D space. The first two variables, x and y were drawn from a regular, rectangular grid, with x having range $[-2, 2]$ and y range $[-1, 1]$. The third variable, z, was computed from x and y with the formula

$$z = 0.5\sin(0.5\pi x) + y,$$

so the z was linearly correlated with y. A semi-linear GTM with one non-linear latent variable, using 10 latent points and 5 basis functions with width 1.0, and one linear latent variable was trained on this data set, starting from a PCA initialization. The trained model, shown in figure 6.4, captures the structure of the data well. However, the data was generated so as to ensure that the initialization would map the continuous and discretized latent variables to the dimensions along which the data exhibited linear and non-linear behaviour, respectively. Initialized the other way around, the model fails to discover anything but linear structure in the data.

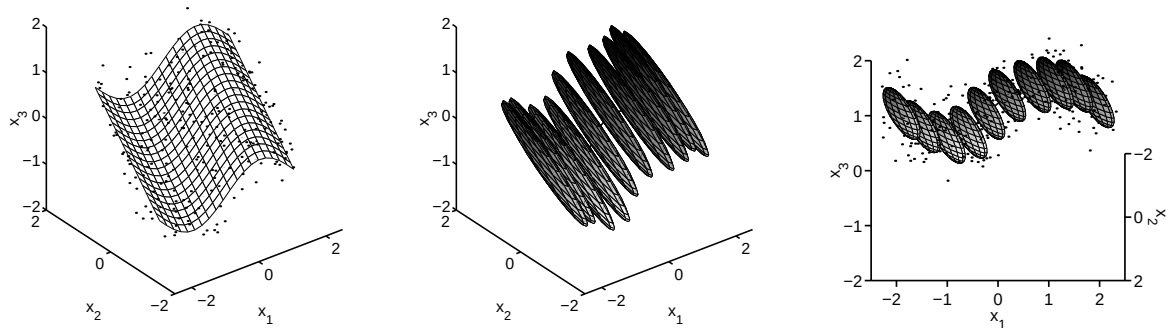


Figure 6.4: A demonstration of a semi-linear model trained on toy-data — the left plot shows the data generating manifold together with the data points plotted as · ; the middle plot shows the trained mixture of 10 PPCAs, plotted as ellipsoids, with their centres lying in the 1-D manifold defined by the GTM, which is plotted as a line; the right plot shows the same thing, but viewed along the 'linear' direction of the manifold (not plotted), highlighting that the model does capture the non-linearity in the data, which is included in this plot as · .

6.5 Missing data

A potential problem with real data sets, not discussed so far, is that of missing data [Little and Rubin, 1987]. Data values may be simply missing or may fall outside known possible ranges, and must therefore be considered as being missing. If we have large amounts of data, we can simply discard data vectors with missing values, but if this is not the case, we would like to be able to use information in the observed values of incomplete vectors. There may be many reasons for the missing data, but assuming that data is *missing at random* — that is, the 'missingness' itself does provide any information[2] — we can learn also from incomplete data.

We want to deal with the missing values in the data just like we dealt with the missing (unknown) latent variables, and integrate them out. For that purpose, we split our data set into two parts — observed[3], $\mathbf{T}^o$, and missing $\mathbf{T}^m$ — and equation (3.1), then becomes

$$p(\mathbf{t}^o|\mathbf{W},\beta) = \iint p(\mathbf{t}^o,\mathbf{t}^m|\mathbf{x},\mathbf{W},\beta)p(\mathbf{x})\,d\mathbf{x}\,d\mathbf{t}^m.$$

Using the fundamental assumption of latent variable models, namely that data variables in $\mathbf{t}$ are independent given the latent variables, we get

$$\begin{aligned} p(\mathbf{t}^o|\mathbf{W},\beta) &= \int p(\mathbf{t}^o|\mathbf{x},\mathbf{W},\beta)p(\mathbf{x}) \int p(\mathbf{t}^m|\mathbf{x},\mathbf{W},\beta)p(\mathbf{x})\,d\mathbf{x}\,d\mathbf{t}^m \\ &= \int p(\mathbf{t}^o|\mathbf{x},\mathbf{W},\beta)p(\mathbf{x})\,d\mathbf{x}. \end{aligned}$$

Thus we can deal with missing values by simply ignoring them, and carry out the calculations of the E- and M-step using only the observed values. Intuitively, for each data point, we are using the information it provides, while ignoring any 'non-information'. Interestingly, the same way of dealing with missing data has been suggested for the SOM [Samad and Harp, 1992].

[2] A counterexample of this is a sensor that fails to give readings when these exceed a certain value.

[3] Note that up till now, we have assumed there were no missing variables in $\mathbf{t}$ and we have referred to all the variables in $\mathbf{t}$ as observed, in contrast to the latent variables which are unobserved. In this section, all the variables in $\mathbf{t}$, some of which may be missing, are referred to as *data variables*.

6.6 Incremental learning

As has been pointed out earlier, the GTM training algorithm discussed in chapter 3 is a batch algorithm, i.e. the update of the parameters is based on all the data. This means that we have to perform the E-step for the whole data set, which is normally computationally rather demanding, before we can update the parameters, $\mathbf{W}$ and β. If instead we could do one iteration of EM for each data point, there is the possibility that the algorithm would converge more quickly, since the model will be updated for each data point, rather than having to wait for the full E-step over all data points. Such an incremental form of EM is presented by Neal and Hinton [1998], who give examples of its application to general Gaussian mixtures. Bishop et al. [1998a] show how it can be adapted for the GTM.

Consider some stage of the standard GTM training algorithm, after an M-step, where we have the 'old' responsibility matrix, $\mathbf{R}^{\text{old}}$, from the previous E-step and current parameters $\mathbf{W}$ and β. If, instead of doing a full E-step, we select a data point $\mathbf{t}_n$, which is the nth row of $\mathbf{T}$, and compute the column-vector, $\mathbf{r}_n^{\text{new}}$, with elements r_{kn} as defined in equation (3.7) (although we keep n fixed), we can revise the quantity $\mathbf{RT}$ in equations (3.13) or (3.15) to

$$(\mathbf{RT})^{\text{new}} = (\mathbf{RT})^{\text{old}} + (\mathbf{r}_n^{\text{new}} - \mathbf{r}_n^{\text{old}})\mathbf{t}_n,$$

where $\mathbf{r}_n^{\text{old}}$ is the nth column of $\mathbf{R}^{\text{old}}$. Similarly, we can revise our estimate of $\mathbf{G}$ from (3.14) by

$$g_{kk}^{\text{new}} = g_{kk}^{\text{old}} + (r_{kn}^{\text{new}} - r_{kn}^{\text{old}}),$$

yielding $\mathbf{G}^{\text{new}}$. We then substitute $(\mathbf{RT})^{\text{new}}$ and $\mathbf{G}^{\text{new}}$ for the corresponding factors in (3.13) or (3.15) and solve for $\mathbf{W}$. Similarly, for β, equation (3.12) becomes

$$\frac{1}{\tilde{\beta}} = \frac{1}{\beta} + \frac{1}{D}\sum_{k}^{K}(r_{kn}^{\text{new}} - r_{kn}^{\text{old}})\|\mathbf{y}(\mathbf{x}_k, \mathbf{W}) - \mathbf{t}_n\|^2.$$

The M-step for a general mixture of Gaussians is relatively simple and hence can be computed quickly. For the GTM, the M-step consists of solving a set of linear equations and is therefore more demanding in terms of computation. This may result in that savings made from faster convergence are lost, because of the increased amount of computation required by more frequent M-steps. This can easily be avoided by, rather than doing the partial E-step for just one data point at the time, doing it for batches of $\hat{N}$ data points, where $\hat{N}$ is chosen to be some suitable fraction of the total number of data points. Neal and Hinton [1998] report substantial net gains in speed when applying this semi-batch algorithm to general mixtures of Gaussians.

Neal and Hinton [1998] also discuss other variants of the EM-algorithm, including a *freeze*-EM algorithm where a proportion of the responsibilities are being frozen (kept fixed) for a number of iterations during which only responsibilities which are not frozen are recomputed. After a few iteration with 'frozen' E-steps, all the responsibilities are recomputed using the normal E-step. This variant has the potential to be particularly useful in the context of GTM, which often uses a rather large number of mixture components (> 100). The plots of responsibility distributions in figures 3.7 and 3.8 suggest that, after only few iterations of 'full' EM, up to 75% of the responsibilities can be frozen, to then be recomputed only every fifth iteration (say).

Note that this incremental form of EM is not an online algorithm, since we are only recycling a finite set of data points, for which we are keeping the old responsibilities. In a real online algorithms, data points arrive one at the time, the model is updated and then the data point is discarded. For the

GTM, we could derive such an online algorithm, either by constructing a gradient descent algorithm, where the Robbins-Monro theorem [Robbins and Monro, 1951] will guarantee convergence, or we could derive an online EM-algorithm [Titterington et al., 1985]. Not only would this allow us to use the GTM in a true online setting, but maybe more importantly, we could use it in tackling very large data sets.

6.7 A Memory-efficient OSL-algorithm

The standard training algorithm for the GTM is rather demanding in terms of memory usage, since it needs to store the $K \times N$ matrix, $\mathbf{R}$, containing the responsibilities[4]. This may pose a problem when applying the GTM to large data sets or when computer resources are scarce. One way to resolve this would be to derive an online training algorithm, as discussed in the end of section 6.6 above.

However, an alternative approach is to use a so called *one-step-late* (OSL) algorithm [Green, 1990]. From (3.15) and (3.12) we see that the only reason that we need to maintain the responsibility matrix, $\mathbf{R}$, is that we need both the responsibilities and the updated weights, $\widetilde{\mathbf{W}}$, in order to update β. To update $\mathbf{W}$, we only need the quantities $\mathbf{RT}$ and $\mathbf{G}$, which both are independent of the size of the data set, and can be computed incrementally. As part of this computation, we would compute the squared distances between mixture components and data points used to update β, but using the old $\mathbf{W}$. Thus we could obtain an EM-algorithm whose memory usage was independent of the size of the training set, where the update of β is one iteration behind the update of $\mathbf{W}$. Green [1990] suggests a similar algorithm for penalized maximum-likelihood estimation, where the penalization term at any given iteration is based on the parameters from the previous iteration. Using such an OSL estimate of β means that we lose the guarantee that the EM-algorithm will converge. However, it is easy to see that both algorithms have the same fixed points, so if the OSL algorithm converges, it will converge to a (local) maxima of the likelihood function. In practice, this algorithm appears to converge just as quickly and reliably as the original EM-algorithm.

6.8 Discrete and mixed data

Up till now, we have assumed that the observed variables have all been continuous. In this section we describe how the GTM can be extended also to model discrete data and, more generally, data with both discrete and continuous variables. The discrete variables may reflect the underlying continuous structure and can be of significant, sometimes even indispensable, help in discovering this structure. Before considering such instances of mixed data, we first consider how to model discrete data, starting with the binary case.

For a binary variable, t, which takes on values $\{0, 1\}$, we assume it follows a binomial distribution,

$$p(t|y) = y^t(1-y)^{(1-t)}$$

where y is the mean of the distribution, which is modelled by the GTM using a logistic sigmoid function, so that

$$y = \frac{1}{1 + \exp(\mathbf{\Phi w})},$$

[4] Typically, it would also maintain a matrix $\mathbf{D}$ of the same size, containing the squared distances between mixture components and data points, but this can be avoided in a more elaborate implementation.

where $\mathbf{w}$ is the $M \times 1$ weight vector mapping from the basis functions to the binary data space. Note that here there is no parameter β.

Variables that are assumed to indicate membership of one of D mutually classes can be modelled using D-dimensional binary vectors, where if a data point belongs to class d, the dth element of the corresponding binary vector is set to 1, while all the other elements are set to 0; this is commonly referred to as a 1-of-D coding scheme. We can model the distribution of such binary vector with a multinomial distribution,

$$p(\mathbf{t}|\mathbf{y}) = \prod_{d}^{D} y_d^{t_d},$$

which is analogous the binomial distribution if $D = 2$. The D-dimensional binary vector is modelled by the GTM using the soft-max function, which we used in section 6.1 to suggest a GTM model with variable mixing coefficients,

$$y_d = \frac{\exp(\mathbf{\Phi}\mathbf{w}_d)}{\sum_{d'}^{D} \exp(\mathbf{\Phi}\mathbf{w}_{d'})}.$$

Since the observed variables are assumed to be independent given the latent variables, we can deal with mixed data by simply multiplying the corresponding Gaussian, binomial and multinomial distributions in the E-step. For the continuous variables the M-step will stay the same, but for binary variables, we must use numerical maximization. This can be done efficiently using iterative least-square (IRLS) methods [McCullagh and Nelder, 1989], or alternatively a general non-linear optimization algorithm [Press et al., 1992]. In any case, it may turn out to be more efficient to do only a partial M-step, which increases but not necessarily maximizes the likelihood, resulting in a generalised EM-algorithm.

6.9 GTM using Gaussian processes

In the basic GTM model, the latent variables are mapped to the data space using a generalised linear regression model, consisting of a linear combination of a set of fixed linear and non-linear basis functions. As pointed out in section 3.2, this gives the computational advantage of an M-step in closed form. However, there are also disadvantages with this form of mapping — maybe most important, it require us to decide on a fixed number of basis functions. This will put a hard constraint on the flexibility of the mapping, which we then usually combine with a soft constraint, imposed by weight regularization. Alternatively we could constrain the mapping only using regularization, by specifying a Gaussian process prior over the distribution of possible functions [Williams and Rasmussen, 1996].

Consider a GTM model where we have removed the basis functions, and instead each latent point, $\mathbf{x}_k$, has a Gaussian mixture component with centre $\mathbf{w}_k$ directly associated with it (like nodes and the corresponding reference vectors in the SOM). Left like that, the model would simply be a K-component, general Gaussian mixture. However, now we specify a prior over the centres,

$$p(\mathbf{W}) = \prod_{d}^{D} (2\pi)^{-K/2} |\mathbf{C}_d|^{-1/2} \exp\left(-\frac{1}{2}\hat{\mathbf{w}}_d^{\mathrm{T}} \mathbf{C}_d^{-1} \hat{\mathbf{w}}_d\right)$$

where $\hat{\mathbf{w}}_d$ is the dth $(K \times 1)$ column of $\mathbf{W}$ and the $\mathbf{C}_d$ are positive definite matrices; typically, it will not be necessary to have separate matrices $\mathbf{C}_d$ for each dimension. This prior defines a distribution over all possible configurations of the centres, where some configurations, e.g. those where the centres are approximately ordered on a low-dimensional manifold, will be much more likely than others.

Combining the prior with the likelihood function of the training data results in a posterior distribution over the weights, which corresponds to a regularized log-likelihood function in the form

$$\ell = \sum_{n}^{N} \ln \left(\frac{1}{K} \sum_{k}^{K} p(\mathbf{t}_n | \mathbf{w}_k, \beta) \right) + \ln p(\mathbf{W}).$$

We can use the EM-algorithm from section 3.2, where the E-step will stay the same while in the M-step, we solve

$$(\mathbf{G} + \beta^{-1}\mathbf{C}^{-1})\mathbf{W} = \mathbf{RT},$$

with respect to $\mathbf{W}$, where all quantities except $\mathbf{C}$ are the same as in the original M-step (3.13). Here we have assumed that we use the same $\mathbf{C}$ for all dimensions. Again we have obtained an M-step in closed form, but this time we need to invert a $K \times K$ matrix, K being the number of latent points or mixture components; the original M-step required the inversion of a $M \times M$ matrix, M being the number of basis functions, which is typically significantly less than K.

We specify our prior, $p(\mathbf{W})$, by specifying $\mathbf{C}$, using a so-called covariance function. Apart from ensuring that $\mathbf{C}$ is positive definite, we would like the covariance function to give a prior such that centres $\mathbf{w}_i$ and $\mathbf{w}_j$ are encouraged to stay close to each other in the data space, when the corresponding nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ are close to each other in the latent space. The literature on Gaussian processes, or equivalently regularization networks [Girosi et al., 1995], provides a wide range of choice [Yaglom, 1987]. For example, we can choose

$$C_{ij} = C(\mathbf{x}_i, \mathbf{x}_j) = \upsilon \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\lambda^2} \right), \tag{6.5}$$

where υ gives overall scale of $\mathbf{C}$, and hence determining the overall degree of smoothing, while λ defines a length scale in the latent space, corresponding to the scale on which different mixture components will influence each other.

The use of a Gaussian process to specify the mapping from latent to data space in the GTM is similar to the discretized Laplacian smoothing used by Utsugi [1997] in the generalized elastic net model described in section (3.4.2). However, the resulting smoothing matrix, used by Utsugi, corresponds to a relatively simple covariance matrix[5], which is rather inflexible and cannot cater for new points in the latent space. From (6.5), we see that λ provides continuous adjustable parameter that controls the 'resolution' of the smoother in the latent space. Moreover, for any new point in the latent space, $\breve{\mathbf{x}}$, we can compute the corresponding point in the data space

$$\breve{\mathbf{y}} = \breve{\mathbf{w}}\mathbf{C}^{-1}\mathbf{W}$$

where, using (6.5), $\breve{\mathbf{w}} = [C(\breve{\mathbf{x}}, \mathbf{x}_1), \ldots, C(\breve{\mathbf{x}}, \mathbf{x}_K)]$.

In the revisited principal curve model, Tibshirani [1992] uses a cubic spline smoother in the M-step, which corresponds to the use of a Gaussian process with a particular choice of $\mathbf{C}$.

The principal advantage of using a Gaussian process rather than a generalised linear regression model is that the flexibility of the mapping can be controlled in a more elegant way, using λ in (6.5); in the generalised regression model the flexibility depends both on the width and the number of basis functions. Using Gaussian processes removes one model parameter (the number of basis functions) and may therefore facilitate the search for the right model complexity, as discussed in chapter 5, especially since both parameters that control the model complexity, υ and λ, are real valued rather

[5] In fact, this matrix is only positive semi-definite, due to the presence of a linear null-space.

than discrete. We could consider a similar scheme to the one used in chapter 5, which is similar to the work of Utsugi [1997], or a full Bayesian treatment using hybrid Monte Carlo methods. The principal disadvantage in using Gaussian processes is the increased amount of computation and memory storage required to do the matrix inversion; however, on a modern workstation, dealing with problems up to moderate size (say, $K \leq 1000$) should be straightforward, and for larger problems there exist efficient approximate methods [Gibbs and MacKay, 1996]

6.10 Altering the latent structure

In section 3.5 we discussed the inherent structural constraints built into the basic GTM model, and the kind of undesirable result that may follow when these structural constraints are at odds with the structure in the data. This is a potential problem whenever our prior knowledge about the underlying structure in the data is limited. However, there are also situations where we do have prior knowledge about the structure in the data which we can build into the GTM model, by chosing a latent space with a corresponding structure. As an example, consider the situation where we know that the data follows a 1-D, smooth cyclic structure; we can then use the 1-D latent space $[0, 2\pi]$ and trigonometric basis functions $\sin(x)$ and $\cos(x)$.

Building such prior knowledge into our model will aid the model fitting, since the space of possible models that we are searching will be smaller (often *much* smaller) than if we had chosen a more general model. This will also make it more likely that the fitted model really reflects the underlying structure in the data.

6.11 Discussion

This chapter has presented a number of possible extension or variations of the basic GTM model. Apart from being interesting on their own merits, the more important result is that, together, they highlight the advantages of the GTM being a probabilistic model. This allows us to make use of well-established ideas from probability theory and statistics, in order to develop the GTM to tackle new sorts of problems. Although one could imagine how to tackle the problems listed in this chapter instead using a SOM model, such attempts would invariably have to made on an *ad hoc* basis.

The variations of the GTM model discussed in this chapter have barely been tried out, and it remains to be seen whether they can become truly useful. Although they all carry some intuitive appeal, there might be other, simpler ways to achieve the same goals, as exemplified to some extent by the experiment with the manifold aligned noise model, shown in figure 6.3.

Chapter 7

Summary and Conclusions

In this final chapter we summarize the work described in this thesis and try to draw some conclusions. We consider potential applications for the GTM as well as problems which are still unresolved. We also briefly review independent work on the GTM, before coming to the final conclusions.

7.1 Related models

Chapter 2 gave a review of some of the models that have been proposed for discovering and exploiting low-dimensional structures in high-dimensional data sets. The GTM has drawn inspiration from many of these models, and can be seen as an extension of more than one of them. Given the latent variable interpretation of principal components analysis in section 2.2.2, it can be seen as a form of non-linear PCA, while generalizing the noise model to independent noise levels for the observed variables results in a non-linear form of factor analysis. However, in its current form, the GTM is limited to three, possibly four, non-linear principal components or factors. The new results on PCA also change our view on the kernel based PCA described in section 2.1.4, but this method is still very different from the GTM, since the corresponding latent space does not have an explicit representation.

Considering the various principal curve and surface models, the GTM provides an alternative, generative model, which is readily applicable for modelling two- and three-dimensional distributions, and could potentially also be used with higher dimensional latent space, provided a more sophisticated approach is adopted for modelling the posterior distribution in the latent space, e.g. using hybrid Monte Carlo methods.

The relationship to the elastic net (sec. 2.2.6) and, in particular, its generalized form (sec. 3.4.2) can be directly understood via Gaussian process variant of the GTM, described in section 6.9, where the difference between the two models boils down to the choice of covariance function. This also forms a connection to the generative variants of principal curves (sec. 2.2.4).

The relationship to the self-organizing map was discussed at length in section 3.4.1. In summary, the GTM can be seen as a principled alternative to the SOM, which circumvents many of its associated theoretical problems, without suffering any significant comparative drawbacks.

7.2 The GTM and its applications

This thesis has primarily been concerned with *establishing* the GTM as a model for non-linear latent variable density modelling and data visualisation. A more thorough investigation of its general ap-

plicability still remains to be done. However, its strong links to PCA and the SOM give reasons for optimism.

PCA is a classical method for feature extraction, in terms of low dimensional representations of data, and has found application in data compression, image analysis, visualization and data preprocessing. Also the SOM has been subject to a wide range of application [Kohonen, 1995], with examples such as categorizing messages in Internet newsgroups, recognizing topographic patterns in EEG spectra and production process control.

The GTM may also find a role in exploratory and confirmatory data analysis. As a related example, MacKay and Gibbs [1997] show how a density network model can be used for discovering correlations in protein sequences.

7.2.1 Visualisation

The GTM holds the potential of becoming a very powerful tool for visualisation of high dimensional data, capable of dealing with continuous as well as discrete and mixed data. Since it is a generative model, it is straightforward to incorporate the GTM into hierarchical, probabilistic visualisation models, such as that suggested by Bishop and Tipping [1996]. The possibility to compute magnification factors which can be visualised jointly with data further enhance this potential. The magnification factor 'adds a dimension' to the visual representation of data, and can thereby provide a better understanding of the data. In particular, it can be used to discover clusters in the data.

7.3 Open questions

There are still a number of questions about the GTM that have so far not been touched upon, and for which there are still no definite answers. These questions are not unique to the GTM — the corresponding questions exist unanswered also for many of the other models discussed in this thesis.

7.3.1 Dimensionality of the latent space

How do we choose the dimensionality of the latent space? Even for the linear PCA and FA models, it is usually not obvious how many principal components or factors should be used. In PCA, a common practice is to plot the eigenvalues of the covariance matrix of the data or, equivalently, the singular values of the singular value decomposition. If the variance in the data primarily is due to a linear combination of L latent variables, only the L largest eigenvalues will be significant, with the remaining $L-1$ being very small. Hence, having computed and plotted these eigenvalues, one may be able to judge, simply by eye, how many latent variables to use. The corresponding procedure for the GTM would be to fit GTM models with increasing number of latent variables and then plot the inverse noise variance, β, against the number of latent variables. Assuming that the distribution of the data is intrinsically L-dimensional, we would expect to see a sharp rise in β, for the first L latent variables, where after the increase of β with L should be much slower. However, in PCA all the D eigenvalues are available at a computationally moderate cost. This unfortunately not the case with the GTM. As has already been mentioned, the computational effort required to fit the model grows exponentially with the dimensionality of the latent space. Moreover, as discussed in chapter 5, the non-linearity in the GTM can result in overfitting-problems, in which case the break in the increasing trend of β may not be that obvious.

An important issue, when deciding on the number of latent variables to use, is the intended use of the GTM. If the purpose is visualisation of data, we may have to sacrifice modelling all underlying degrees of freedom in the data, in order to be able to visualise it using a single, global model. Unless we want to employ additional visualisation techniques, we will be restricted to three, possibly four, latent variables. We can still hope that the GTM model will provide a model which is as good as possible, given these restrictions, although it is not yet clear under which circumstances this will actually happen.

7.3.2 Structural constraints of the GTM

As pointed out in section 3.5, the basic GTM model is best suited to model moderately curved, L-dimensional distributions of roughly rectangular shape. What happens when we apply the GTM under different circumstances? The objective of the training algorithm of the GTM is to maximise the likelihood of the training data, subject to the constraints imposed by (e.g.) the number and the width of the basis functions and the degree of weight regularization, and this objective will always be the same. If the constraints imposed are too strong, e.g. if the manifold is too stiff, then the resulting GTM model is bound to be sub-optimal, at least as a density model. With more relaxed constraints, on the other hand, we may end up with a significantly better density model, but which gives a 'complex explanation' to an inherently simple structure, as in example 3.3. This is not to say that the GTM is overfitting, capturing structure in the data which is due to noise — the problem is that the structure in the data, although simple, is very different from the inherent structure of the density model provided by the GTM. Of course, there are ways in which we could handle this specific case (the two Gaussians in 2-D) — as pointed out in section 6.10, in situations where we do have prior knowledge about the structure in the data, we can build this knowledge into the GTM model, which usually leads to more efficient model fitting and better agreement between the fitted model and the data generating mechanism. However, in general we cannot expect the GTM to be able to provide 'interpretable' models of arbitrary low-dimensional distributions. Therefore, an important task will be to develop reliable diagnostics that can be used to detect folds and other undesirable sub-structures in the manifold.

7.3.3 The merits of being 'principled'

A recurring term used together with the GTM, especially when relating it to the SOM is 'principled'. This is motivated by the fact that the GTM is derived from probabilty theory and statistics, whereas the SOM is motivated only by heuristic and empirical arguments. However, are there any practical gains to be made from this? Are the results obtained with the GTM normally (if at all) 'better' than those obtained with the SOM? Is the choice of basis functions for the GTM any less arbitrary than the choice of neighbourhood functions for the SOM?

No doubt, results obtained in terms of visualisation from GTM and the SOM are typically very similar, as we would expect given the many similarities between the two models. However, except for simple toy examples it is typically very difficult or even impossible to judge what is a 'good' visualisation. With no other objective measure to discriminate between models, we ought to prefer models which have a sound theoretical foundation to those which have not.

It is also true that, as much as visualisation results vary for the SOM with varying choices of the neighbourhood function, as much will they vary for the GTM with varying choices of basis functions. However, if we are working with the SOM, we are left to little but rules of thumb for choosing

our neighbourhood function; attempts to empirically find suitable parameters for the neighbourhood function would be hampered by the fact that the SOM does not minimize an objective function, and hence we have no measure for comparison. For the GTM, the limitations are 'only' practical — given infinite amounts of data and computing time, we will be able to construct an optimal model for any distribution, given the constraints imposed by the particular GTM model we are using. Even though this would not be possible in practice, a framework where such an objective is at least theoretically achievable, is clearly more desirable than one where it does not even exist. To quote Judea Pearl " ... we find it more comfortable to compromise an ideal theory that is well understood than to search for a new surrogate theory, with only gut feeling for guidance" [Pearl, 1988, page 20].

7.4 Independent work on GTM

Although the GTM is relatively 'young', it has already inspired new work, also among independent researchers[1]. Bishop et al. [1997a] use the GTM to model the emission density of a hidden Markov model, thereby extending the GTM to deal with time series data, where the assumption that the data points are generated independently is no longer required. They show an example of how this model can be used to visualise time series data from a helicopter flight recorder, where different regions in the latent space corresponds to different modes of flying. Kiviluoto and Oja [1998] develop a probabilistic, hybrid GTM-SOM model which they call the *S-map*; they give empirical evidence that this model, under certain circumstances, has a stronger tendency to self-organize — that is, adapting so that the topological structure of the model reflects the topological structure of the data — when starting from a random initialization. Pajunen and Karhunen [1997] show how the GTM can be used to perform a non-linear form of *independent components analysis* (ICA) [Bell and Sejnowski, 1995, Amari et al., 1996], also knows as blind source separation. This GTM based model can be used to separate independent sources which have been non-linearly mixed, assuming that the probability distributions of the sources are known and that the non-linear mixing function is smooth.

A search on the Internet gave additional indications of the GTM being used as an unsupervised visualization technique for cloud type recognition, for risk prediction in pregnancy and for dimensionality reduction of articulatory data.

7.5 Conclusions

The generative topographic mapping provides a method for modelling continuous, low-dimensional, non-linear structures in high-dimensional spaces. It provides a way of doing non-linear PCA or FA, although in practice it is still limited to a small number of principal components or factors. It forms a principled alternative to the SOM, resolving many of its inherent problems.

As has been exemplified in this thesis, an important application for the GTM is visualisation of high-dimensional data. The possibility of computing the magnification factor as a continuous function over the latent space and incorporating this in visualisation plots, can make visualised data easier to interpret.

Since the GTM is a probabilistic model, it fits into the framework of probability theory and statistics. We can thus make use of established and well-founded theory to deal with issues such as selection of model complexity. Moreover, we benefit from it when extending the GTM to deal with e.g. missing data and data which take discrete or mixed discrete-continuous values.

[1] The GTM was proposed by Bishop, Svensén, and Williams.

Since there are examples where the GTM, when fitted to data with rather simple structure, ends up being rather complex, developing reliable diagnostics for detecting these situations will be an important future task. Moreover, before it can fully be assessed, the GTM will need to be thoroughly tested in a wide range of real applications.

Apart from this thesis and papers referenced herein, the work on GTM has also resulted in a MATLAB® implementation with associated documentation, which is freely available on the Internet, at `http://www.ncrg.aston.ac.uk/GTM/`.

Bibliography

S. Amari, A. Cichocki, and H. H. Yang. A new learning algorithm for blind signal separation. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1996.

P. Baldi and K. Hornik. Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

D. J. Bartholomew. *Latent Variable Models and Factor Analysis.* Charles Griffin and Co. Ltd, London, 1987.

A. J. Bell and T. J. Sejnowski. An information maximisation approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1004–1034, 1995.

C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

C. M. Bishop, G. E. Hinton, and I. G. D. Strachan. GTM Through Time. In *Proceedings of the IEE Fifth International Conference on Artificial Neural Networks*, pages 111–116, London, 1997a. IEE.

C. M. Bishop and G. D. James. Analysis of multiphase flows using dual-energy gamma densitometry and neural networks. *Nuclear Instruments and Methods in Physics Research*, A327:580–593, 1993.

C. M. Bishop, M. Svensén, and C. K. I. Williams. EM Optimization of Latent-Variable Density Models. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 465–471. MIT Press, 1996a.

C. M. Bishop, M. Svensén, and C. K. I. Williams. GTM: a Principled Alternative to the Self-Organizing Map. In C. von der Malsburg, W. von Selen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Proceedings 1996 International Conference on Artificial Neural Networks, ICANN 96*, volume 1112 of *Lecture Notes in Computer Science*, pages 165–170, Bochum, Germany, 1996b. Springer-Verlag.

C. M. Bishop, M. Svensén, and C. K. I. Williams. GTM: A Principled Alternative to the Self-Organizing Map. In M. C. Mozer, M. I. Jordan, and T. Petche T, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 354–360. MIT Press, 1997b.

C. M. Bishop, M. Svensén, and C. K. I. Williams. Magnification Factors for the GTM Algorithm. In *Proceedings of the IEE Fifth International Conference on Artificial Neural Networks*, pages 64–69, London, 1997c. IEE.

C. M. Bishop, M. Svensén, and C. K. I. Williams. Magnification Factors for the SOM and GTM Algorithms. In *Proceedings of the Workshop on Self-Organizing Maps*, pages 333–338. Helsinki University of Technology, 1997d.

C. M. Bishop, M. Svensén, and C. K. I. Williams. Developments of the GTM Algorithm. submitted to Neurocomputing — under review., 1998a.

C. M. Bishop, M. Svensén, and C. K. I. Williams. GTM: The Generative Topographic Mapping. *Neural Computation*, 10(1), 1998b.

C. M. Bishop and M. E. Tipping. A hierarchical latent variable model for data visualization. Technical report NCRG/96/028, Neural Computing Research Group, Aston University, 1996. A version to appear in *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294, 1988.

C. Bregler and S. M. Omohundro. Surface learning with applications to lipreading. In J. D. Cowan, G. T. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 43–50, San Mateo, CA, 1994. Morgan Kaufmann.

J. S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fogelman Soulié and J. Hérault, editors, *Neurocomputing: Algorithms, Architectures and Applications*, pages 227–236. Springer-Verlag, New York, 1990.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel. The Helmholtz machine. *Neural Computation*, 7(5):889–904, 1995.

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

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

R. Durbin, R. Szeliski, and A. Yuille. An analysis of the elastic net approach to the travelling salesman problem. *Neural Computation*, 1(3):348–358, 1989.

R. Durbin and D. Willshaw. An analogue approach to the travelling salesman problem. *Nature*, 326: 689–691, 1987.

E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetics*, 67:47–55, 1992.

J. Etezadi-Amoli and R. P. McDonald. A second generation nonlinear factor analysis. *Psychometrika*, 48(3):315–342, 1983.

B. S. Everitt. *An Introduction to Latent Variable Models*. Chapman and Hall, London, 1984.

J. H. Friedman. Multivariate adaptive regression splines (with discussion). *Annals of Statistics*, 19 (1):1–141, 1991.

K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, San Diego, second edition, 1990.

Z. Ghahramani and G. E. Hinton. Hierarchical Nonlinear Factor Analysis and Topographic Maps. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1998.

M. N. Gibbs and D. J. C. MacKay. Efficient implementation of Gaussian processes. in preparation, available at: `ftp://wol.ra.phy.cam.ac.uk/pub/www/mng10/GP/gpros.ps.gz`, 1996.

F. Girosi, M. Jones, and T. Poggio. Regularization Theory and Neural Networks Architectures. *Neural Computation*, 7(2):219–269, 1995.

P. J. Green. On the use of the EM Algorithm for Penalized Likelihood Estimation. *J. Roy. Stat. Soc. B*, 52(3):443–452, 1990.

S. F. Gull. Developments in maximum entropy data analysis. In J. Skilling, editor, *Maximum Entropy and Bayesian Methods, Cambridge, 1988*, pages 53–71. Kluwer, Dordrecht, 1989.

G. De Haan and Ö Egecioglu. Links between self-organizing feature maps and weighted vector quantization. In *Proc. IEEE Int. Joint Conf. Neural Networks*, pages 887–892, 1991. Singapore.

T. Hastie. Principal curves and surfaces. Technical report, Department of Statistics, Stanford University, 1984.

T. Hastie and W. Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84(406): 502–516, 1989.

S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994.

G. Hinton, M Revow, and P Dayan. Recognizing handwritten digits using mixtures of linear models. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, Cambridge MA, 1995a. MIT Press.

G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1158–1161, 1995b.

G. E. Hinton, P. Dayan, and M. Revow. Modelling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8(1):65–74, 1997.

G. E. Hinton and Z. Ghahramani. Generative Models for Discovering Sparse Distributed Representations. *Philosophical Transactions Royal Society B*, 352:1177–1190, 1997.

G. E. Hinton, C. K. I. Williams, and M. D. Revow. Adaptive elastic models for hand-printed character recognition. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 512–519. Morgan Kauffmann, 1992.

I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

K. G. Jöreskog. Some contributions to maximum likelihood factor analysis. *Psychometrika*, 32:443–482, 1967.

J. H. Kaas, R. J. Nelson, M. Sur, and M. M. Merzenich. Organization of somatosensory cortex in primates. In F. O. Schmitt, F. G. Worden, G. Adelman, and S. G. Dennis, editors, *The organization of the cerebral cortex*, pages 237–261. MIT Press, 1981.

N. Kambhatla and T. K. Leen. Fast non-linear dimension reduction. In Jack D. Cowan, Gerald Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, volume 6, pages 152–159. San Francisco, CA: Morgan Kaufmann, 1994.

K. Kiviluoto and E. Oja. S-Map: A network with a simple self-organization algorithm for generative topographic mappings. In *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1998. To appear.

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 1995.

T. Kohonen, J. Hynninen, J. Kangas, and J. Laakksonen. *SOM_Pak The Self-Organizing Map Program Package*. Helsinki University of Technology, Espoo, Finland, 1995. version 3.1.

M. A. Kraaijveld, J. Mao, and A. K. Jain. A nonlinear projection method based on Kohonen's topology preserving maps. *IEEE Transactions on Neural Networks*, 6(3):548–559, 1995.

M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChe Journal*, 37(2):233–243, 1991.

D. N. Lawley. The estimation of factor loading by the method of maximum likelihood. *Proc. R. Soc. Edinb.*, A 60:64–82, 1940.

D. N. Lawley and A. E. Maxwell. *Factor Analysis as a Statstical Method*. Butterworth & Co., London, 2nd edition, 1971.

P. F. Lazarsfeld and N. W. Henry. *Latent structure analysis*. Houghton Mifflin Company, Boston, 1968.

M. LeBlanc and R. Tibshirani. Adaptive principal surfaces. *Journal of the American Statistical Association*, 89(425):53–64, 1994.

Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, 1980.

B. G. Lindsay. The geometry of mixture likelihoods: A general theory. *Annals of Statistics*, 11(1): 86–94, 1983.

R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. John Wiley, New York, 1987.

D. Lovelock and H. Rund. *Tensors, differential forms and variational principles*. John Wiley & Sons, Inc., 1975.

D. Lowe and M. Tipping. Feed-forward neural networks and topographic mappings for exploratory data analysis. *Neural Computing and Applications*, 4:83–95, 1996.

S. J. Luttrell. Using self-organizing maps to classify radar range profiles. In *Proceedings IEE Fourth International Conference on Artificial Neural Networks*, pages 335–340, London, 1995. IEE.

S. P. Luttrell. A Bayesian analysis of self-organizing maps. *Neural Computation*, 6(5):767–794, 1994.

D. J. C. MacKay. *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology, 1991.

D. J. C. MacKay. A practical Bayesian framework for back-propagation networks. *Neural Computation*, 4(3):448–472, 1992.

D. J. C. MacKay. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research, Section A*, 354(1):73–80, 1995.

D. J. C. MacKay. Density networks and their application to protein modelling. In J. Skilling and S. Sibisi, editors, *Maximum Entropy and Bayesian Methods, Cambridge 1994*, pages 259–268, Dordrecht, 1996. Kluwer.

D. J. C. MacKay and M. N. Gibbs. Density networks. In Kay and Titterington, editors, *Proceedings of Meeting on Statistics and Neural Nets, Edinburgh*. O.U.P., 1997. in press.

K. Mardia, J. Kent, and M. Bibby. *Multivariate analysis*. Academic Press, 1979.

P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman and Hall, 1989. second editon.

R. P. McDonald. A general approach to nonlinear factor analysis. *Psychometrika*, 27:397–415, 1962.

R. P. McDonald. Numerical methods for polynomials models in nonlinear factor analysis, 1967.

R. P. McDonald. The simultaneous estimation of factor loadings and scores. *British Journal of Mathematical and Statistical Psychology*, 32:212–228, 1979.

F. Mulier and V. Cherkassky. Self-organization as an iterative kernel smoothing process. *Neural Computation*, 7(6):1165–1177, 1995.

R. M. Neal. Bayesian training of backpropagation networks by the hybrid Monte Carlo method. Technical Report CRG-TR-92-1, Department of Computer Science, University of Toronto, Canada, 1992.

R. M. Neal. *Bayesian Learning for Neural Networks*. Springer, 1996. Lecture Notes in Statistics 118.

R. M. Neal and P. Dayan. Factor analysis using delta-rule wake-sleep learning. *Neural Computation*, 9(8):1781–1803, 1997.

R. M. Neal and G. E. Hinton. A new view of the EM algorithm that justifies incremental and other variants. Technical Report CRG-TR-92-1, Department of Computer Science, University of Toronto, Canada, 1998. To appear in Proceedings NATO Advanced Study Institute, Erice, ed. M. I. Jordan, Kluwer.

F. O'Sullivan. Discretized laplacian smoothing by fourier methods. *Journal of the American Statistical Association*, 86(415):634–642, 1991.

P. Pajunen and J. Karhunen. A maximum likelihood approach to nonlinear blind source separation. In *Proceedings of the 1997 Int. Conf. on Artificial Neural Networks (ICANN'97)*, pages 541–546. Springer-Verlag, 1997.

C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization*. Prentice-Hall, 1982.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Probable Inference*. Morgan Kaufmann, revised second printing edition, 1988.

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

B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, 1996.

H. Ritter. Parametrized self-organizing maps. In *Proceedings ICANN'93 International Conference on Artificial Neural Networks*, pages 568–575, Amsterdam, 1993. Springer-Verlag.

R. Ritter and K. Schulten. On the stationary state of Kohonen's self-organizing sensory mapping. *Biological Cybernetics*, 54:99–106, 1986.

R. Ritter and K. Schulten. Convergence properties of Kohonen's topology conserving maps: fluctuations, stability and dimension selection. *Biological Cybernetics*, 60:69–71, 1988.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

D. B. Rubin and D. T. Thayer. EM algorithms for ML factor analysis. *Psychometrika*, 47(1):69–76, 1982.

T. Samad and S. A. Harp. Self-organization with partial data. *Network: Computation in Neural Systems*, 3(2):205–212, 1992.

J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409, 1969.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical Report 44, Max-Planck-Institut für biolgische Kybernetik, 1996. available via `http://www.mpik-tueb.mpg.de/bu.html`.

P. Simard, B. Victorri, Y. Le Cun, and J. Denker. Tangent prop – a formalism for specifying selected invariances in an adaptive network. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 895–903, San Mateo, CA, 1992. Morgan Kaufmann.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society, B*, 36(1):111–147, 1974.

G. Strang. *Linear algebra and its applications*. Harcourt Brace Jovanovich, third edition, 1988.

N. Suga. Cortical computational maps for auditory imaging. *Neural Networks*, 3:3–22, 1991.

R. Tibshirani. Principal curves revisited. *Statistics and Computing*, 2:183–190, 1992.

M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analysers. Technical report NCRG/97/003, Neural Computing Research Group, Aston University, 1997a. Submitted to *Neural Computation*.

M. E. Tipping and C. M. Bishop. Probabilistic principal component analysers. Technical report, Neural Computing Research Group, Aston University, 1997b. Submitted to *Journal of the Royal Statistical Society, B*.

M. Titterington, Smith, and Makov. *The statistical analysis of finite mixture distributions*. Wiley, 1985.

A. Ultsch. Self-organizing networks for visualization and classification. In O. Opitz, B. Lausen, and R. Klar, editors, *Information and Classification*, pages 307–313, Berlin, 1993. Springer.

A. Ultsch and H. P. Siemon. Kohonen's self-organizing feature maps for exploratory data analysis. In *Proc. Intern. Neural Networks*, pages 305–308, Paris, 1990. Kluwer Academic Press.

A. Utsugi. Topology selection for self-organizing maps. *Network: Computation in Neural Systems*, 7: 727–740, 1996.

A. Utsugi. Hyperparameter selection for self-organizing maps. *Neural Computation*, 9(3):623–635, 1997.

A. R. Webb. An approach to nonlinear principal components–analysis using radially symmetrical kernel functions. *Statistics and Computing*, 6(2):159–168, 1996.

C. K. I. Williams. *Combining deformable models and neural networks for handprinted digit recognition.* PhD thesis, Dept. of Computer Science, University of Toronto, 1994.

C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 514–520. MIT Press, 1996.

C. K. I. Williams, M. D. Revow, and G. E. Hinton. Hand-printed digit recogntion using deformable models. In L. Harris and M. Jenkin, editors, *Spatial Vision in Humans and Robots.* Cambridge University Press, 1993.

A. M. Yaglom. *Correlation Theory of Stationary and Related Random Functions Volume I:Basic Results.* Springer Verlag, 1987.

R. S. Zemel and G. E. Hinton. Developing population codes by minimizing description length. In J. D. Cowan, G. T. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 11–18, San Mateo, CA, 1994. Morgan Kaufmann.

Appendix A

Derivatives of the Log-likelihood Function

In this appendix we derive formulae for the first and second derivatives, with respect to the weight parameters, $\mathbf{W}$, of the error function in (5.5), corresponding to the negative, unregularized log-likelihood function; here we re-write it as

$$S_{\mathbf{T}} = -\sum_{n}^{N} \ln \frac{1}{K} \sum_{k}^{K} p(\mathbf{t}_n|\mathbf{x}_k, \mathbf{W}, \beta). \tag{A.1}$$

From (3.1) and (3.10), we get

$$p(\mathbf{t}_n|\mathbf{x}_k, \mathbf{W}, \beta) = \left(\frac{\beta}{2\pi}\right)^{D/2} \exp\left\{-\frac{\beta}{2}\sum_{d}^{D}(t_{nd} - \boldsymbol{\phi}_k \mathbf{w}_d)^2\right\} = p_{kn}, \tag{A.2}$$

where $\boldsymbol{\phi}_k$ is the kth row of $\boldsymbol{\Phi}$, $\mathbf{w}_d$ is the dth column of $\mathbf{W}$, and we have introduced p_{kn}, in order to simplify the notation. Thus, (3.7) now reads

$$r_{kn} = \frac{p_{kn}}{\sum_{k'}^{K} p_{k'n}} \tag{A.3}$$

A.1 First derivatives

Differentiating (A.1), using (A.2) and (A.3), along with standard rules for differentiation, we get

$$\begin{aligned}
\frac{\partial S_{\mathbf{T}}}{\partial w_{ij}} = &-\sum_{n}^{N} \frac{1}{\sum_{k'}^{K}\left(\frac{\beta}{2\pi}\right)^{D/2} \exp\left\{-\frac{\beta}{2}\sum_{d}^{D}(t_{nd} - \boldsymbol{\phi}_{k'}\mathbf{w}_d)^2\right\}} \\
&\sum_{k}^{K}\left(\frac{\beta}{2\pi}\right)^{D/2} \exp\left\{-\frac{\beta}{2}\sum_{d}^{D}(t_{nd} - \boldsymbol{\phi}_k \mathbf{w}_d)^2\right\} \beta(t_{nj} - \boldsymbol{\phi}_k \mathbf{w}_j)\phi_{ki} \\
= &-\sum_{n,k}^{N,K} \frac{p_{kn}}{\sum_{k'}^{K} p_{k'n}} \beta(t_{nj} - \boldsymbol{\phi}_k \mathbf{w}_j)\phi_{ki} \\
= &-\sum_{n,k}^{N,K} r_{kn}\beta(t_{nj} - \boldsymbol{\phi}_k \mathbf{w}_j)\phi_{ki}.
\end{aligned} \tag{A.4}$$

A.2 Second derivatives

First, we introduce

$$\eta_{kn}^{ij} = \beta(t_{nj} - \boldsymbol{\phi}_k \mathbf{w}_j)\phi_{ki}, \tag{A.5}$$

from which it follows directly that

$$\frac{\partial \eta_{kn}^{ij}}{\partial w_{pq}} = \begin{cases} -\beta\phi_{kp}\phi_{ki} & \text{if } j = q, \\ 0 & \text{otherwise,} \end{cases} \tag{A.6}$$

and we see from (A.2) and (A.5) that

$$\frac{\partial p_{kn}}{\partial w_{ij}} = p_{kn}\eta_{kn}^{ij}. \tag{A.7}$$

Second, we differentiate r_{kn} with respect to w_{pq}, using (A.3) and (A.7), to get

$$\begin{aligned} \frac{\partial r_{kn}}{\partial w_{pq}} &= \frac{\partial}{\partial w_{pq}} \frac{p_{kn}}{\sum_{k'}^{K} p_{k'n}} \\ &= \frac{p_{kn}}{\sum_{k''}^{K} p_{k''n}}\eta_{kn}^{pq} - \frac{p_{kn}}{\sum_{k''}^{K} p_{k''n}} \sum_{k'}^{K} \frac{p_{k'n}}{\sum_{k''}^{K} p_{k''n}}\eta_{k'n}^{pq} \\ &= r_{kn}(\eta_{kn}^{pq} - \sum_{k'}^{K} r_{k'n}\eta_{k'n}^{pq}) \end{aligned} \tag{A.8}$$

Finally, from (A.4), (A.5), (A.6) and (A.8), we get

$$\begin{aligned} \frac{\partial^2 S_{\mathbf{T}}}{\partial w_{ij}\partial w_{pq}} &= \frac{\partial}{\partial w_{pq}} - \sum_{n,k}^{N,K} r_{kn}\eta_{kn}^{ij} \\ &= \begin{cases} -\sum_{n,k}^{N,K} r_{kn}(\eta_{kn}^{ij}(\eta_{kn}^{pq} - \sum_{k'}^{K} r_{k'n}\eta_{k'n}^{pq}) - \beta\phi_{kp}\phi_{ki}) & \text{if } j = q, \\ -\sum_{n,k}^{N,K} r_{kn}\eta_{kn}^{ij}(\eta_{kn}^{pq} - \sum_{k'}^{K} r_{k'n}\eta_{k'n}^{pq}) & \text{otherwise.} \end{cases} \end{aligned} \tag{A.9}$$

If we study (A.9) we see that whenever r_{kn} is close to one, $(\eta_{kn}^{pq} - \sum_{k'}^{K} r_{k'n}\eta_{k'n}^{pq})$ will be close to zero. For a trained GTM model, it is often the case that almost all the responsibility for a data point rests with a single mixture component, in which case it would be reasonable to use the approximation

$$\frac{\partial^2 S_{\mathbf{T}}}{\partial w_{ij}\partial w_{pq}} = \begin{cases} \sum_{n,k}^{N,K} r_{kn}\beta\phi_{kp}\phi_{ki} & \text{if } j = q, \\ \mathbf{0} & \text{otherwise,} \end{cases} \tag{A.10}$$

which is a block-diagonal matrix with D identical $M \times M$ building blocks

$$\beta\mathbf{\Phi}^{\mathrm{T}}\mathbf{G}\mathbf{\Phi}, \tag{A.11}$$

where $\mathbf{G}$ is defined in (3.14). If we combine this with $\mathbf{H}_{\mathbf{w}}$, given in (5.11), we get the building blocks of the full approximate Hessian as

$$\beta\mathbf{\Phi}^{\mathrm{T}}\mathbf{G}\mathbf{\Phi} + \alpha\mathbf{I},$$

which we recognize (subject to a multiplicative factor β) from the left-hand side of the M-step equation (3.15); thus, we have already computed this approximate form of the Hessian, as part of the normal training algorithm.

A.2.1 Computing the exact Hessian

To turn the formulae from (A.9) into a computational algorithm we expand the expression $r_{kn}\eta_{kn}^{ij}(\dots)$ using (A.5), to get

$$\begin{aligned} r_{kn}\eta_{kn}^{ij}(\eta_{kn}^{pq} - \sum_{k'}^{K} r_{k'n}\eta_{k'n}^{pq}) &= r_{kn}\beta(t_{nj} - \boldsymbol{\phi}_k \mathbf{w}_j)\phi_{ki} \\ &\quad \left(\beta(t_{nq} - \boldsymbol{\phi}_k \mathbf{w}_q)\phi_{kp} - \sum_{k'}^{K} r_{k'n}\beta(t_{nq} - \boldsymbol{\phi}_{k'} \mathbf{w}_q)\phi_{k'p}\right) \\ &= \beta^2 r_{kn}(\phi_{ki}\phi_{kp}(t_{nj} - y_{kj})(t_{nq} - y_{kq}) - \phi_{ki}(t_{nj} - y_{kj})z_n^{pq}) \end{aligned} \tag{A.12}$$

where

$$z_n^{pq} = \sum_{k'}^{K} r_{k'n}(t_{nq} - y_{k'q})\phi_{k'p}.$$

Using (A.11) and (A.12), the formulae from (A.9) can now be put into matrix form,

$$\mathbf{H}_{\mathbf{T}}^{jq} = \begin{cases} \boldsymbol{\Phi}^{\mathrm{T}}(\beta\mathbf{G} - \beta^2(\mathbf{Q}^{jq} - (\mathbf{D}^j \odot \mathbf{R})(\mathbf{D}^q \odot \mathbf{R})^{\mathrm{T}}))\boldsymbol{\Phi} & \text{if } j = q, \\ -\beta^2\boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{Q}^{jq} - (\mathbf{D}^j \odot \mathbf{R})(\mathbf{D}^q \odot \mathbf{R})^{\mathrm{T}})\boldsymbol{\Phi} & \text{otherwise,} \end{cases} \tag{A.13}$$

where $\mathbf{H}_{\mathbf{T}}^{jq}$ is the $M \times M$ sub-matrix of the data Hessian, $\mathbf{H}_{\mathbf{T}}$ corresponding to the weights of y_j and y_q, $\mathbf{Q}^{jq}$ is a $K \times K$ diagonal matrix with entries

$$q_{kk}^{jq} = \sum_{n}^{N} r_{kn}(t_{nj} - y_{kj})(t_{nq} - y_{kq}),$$

$\mathbf{D}^j$ is a $K \times N$ matrix with entries

$$d_{nk}^{j} = (t_{nj} - y_{kj}),$$

and $\mathbf{R}$ is the responsibility matrix from (3.13), with $\odot$ denoting component-wise multiplication.

A.2.2 Exact vs. approximate form

Clearly, computing the exact Hessian as well as using it in further computation, will require much more computation, compared to the approximate form. It would therefore be useful to be able to assess the penalty we pay in terms of inaccuracy when using the approximate form, and judge that against computational savings. A simple approach for estimating α, β and σ discussed in section 5.3, would be to evaluate the logarithm of the evidence for α, β and σ, given in (5.17), over a grid in α-β-σ-space. This includes the logarithm of the determinant of the Hessian,

$$\ln|\mathbf{H}| = \ln\prod_{i}^{W}(\lambda_i + \alpha) = \sum_{i}^{W}\ln(\lambda_i + \alpha), \tag{A.14}$$

where λ_i is the ith eigenvalue of $\mathbf{H}_{\mathbf{T}}$. Figure A.1 shows $\ln|\mathbf{H}|$ plotted against $\log_{10}\alpha$ during different stages of training on the artificial data from section 5.4. A problem with the exact Hessian is that is not guaranteed to be positive definite; in fact, it is generally the case that the eigen-decomposition results in a small number of non-positive eigenvalues. For the plots in figure A.1, terms with $\lambda_i \leq 0$ have been excluded from the sum in (A.14). It can happen that also the approximate Hessian has zero eigenvalues, although this is uncommon, which is then treated the same way.

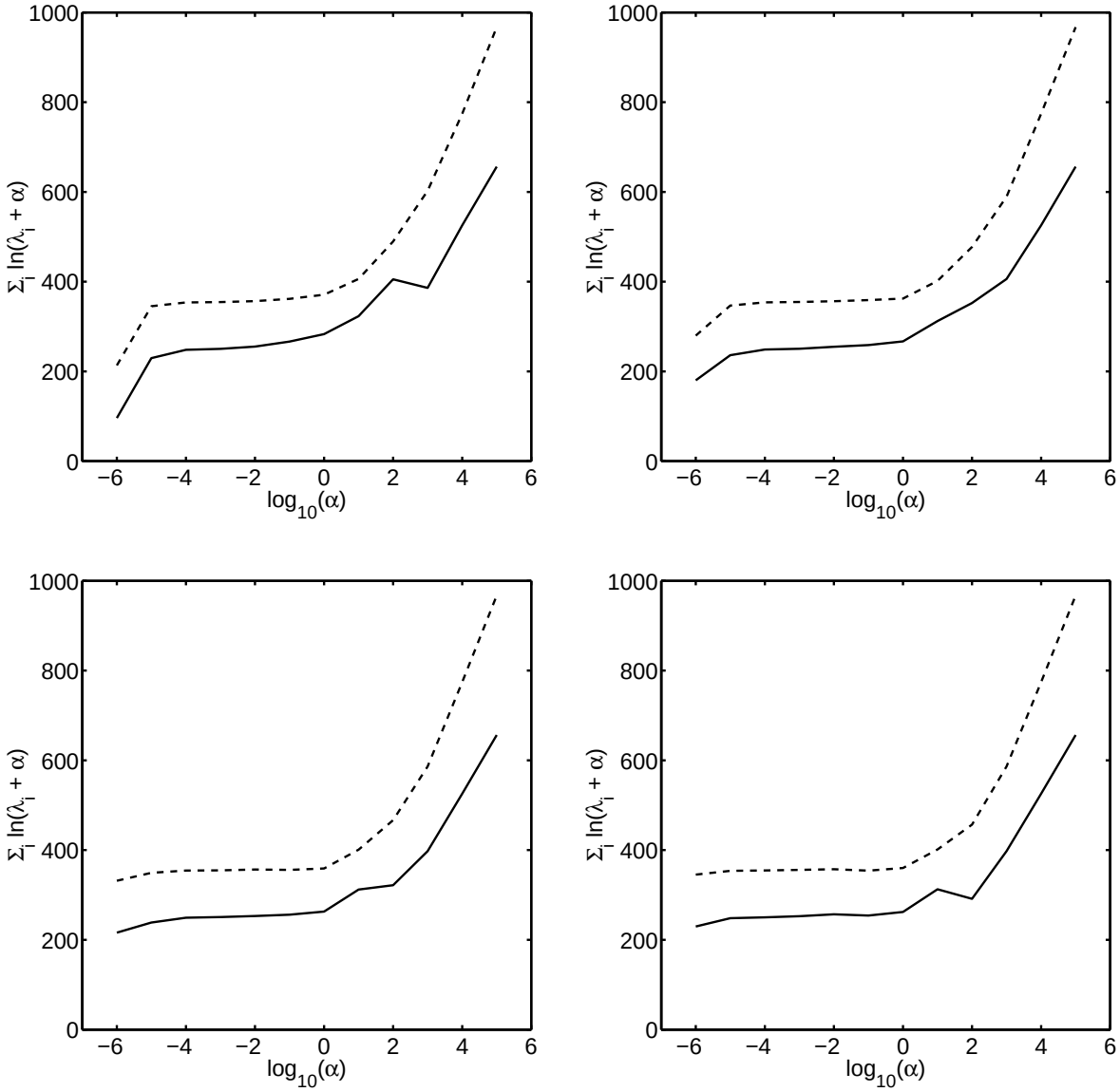


Figure A.1: The plots show $\sum_i^W \ln(\lambda_i + \alpha)$ against $\log_{10} \alpha$, where the dashed line shows results from using the approximate Hessian while the solid line shows results obtained using the exact form. The four plots correspond (top-down, left-to-right) to results evaluated after 0 (i.e. after initialization, but before training has started), 5, 15, and 50 iterations of training.

Data set	K	N	M	D	size(**H**)	**Method**	**Time**
Pipe flow	400	1000	84	12	1016064	Exact	99516
(example 3.2)						Apprx	11
Crabs	100	200	39	5	38025	Exact	72
(example 4.2)						Apprx	1
Synthetic	225	400	28	3	7056	Exact	270
(section 5.4)						Apprx	2

Table A.1: Comparison of the time (measured in ticks — 3 ticks per second) required for computing the exact and approximate forms of the Hessian, using some of the data sets described earlier in this thesis. The notation is the same as has been used before, i.e. K is the number of latent points, N is the number of data points, M is the number of basis functions and D is the dimensionality of the data.

In section 5.3, we also discuss methods for estimating the hyper-parameters α, β and σ during training. For this purpose, we make use of a quantity, γ, interpreted as the effective number of weight parameters and defined as

$$\gamma = \sum_{i}^{W} \frac{\lambda_i}{\lambda_i + \alpha}.$$

Figure A.2 show γ plotted against $\log_{10} \alpha$, during different iterations of training for the same data set which were used to produce the plots in A.1. Again, terms corresponding to non-positive eigenvalues, λ_i, have been excluded.

As can be seen, there seems to be rather significant discrepancies between the exact and approximate value for $\ln|\mathbf{H}|$ whereas the differences for γ are smaller. The differences for both $\ln|\mathbf{H}|$ and γ appear not to change very much with training, which is somewhat unexpected, since we would expect them to decrease as a consequence of improved fit to data.

Table A.1 contains a comparison of the computational effort required for the exact and approximate form of the Hessian, for the some of the data sets described in this thesis. Note that the computational effort required does not only depend on the size of the Hessian, but also the size of the data set and the number of latent points involved. Although this comparison is far from exhaustive, the figures in table A.1 clearly shows that computing the exact Hessian is *much* more expensive than computing the approximate form (which we have computed anyway, so the actual cost is 0). Moreover, in a lot of the subsequent calculation using the approximate Hessian, we really only need to compute with one of the identical blocks from the diagonal, which will result in additional savings.

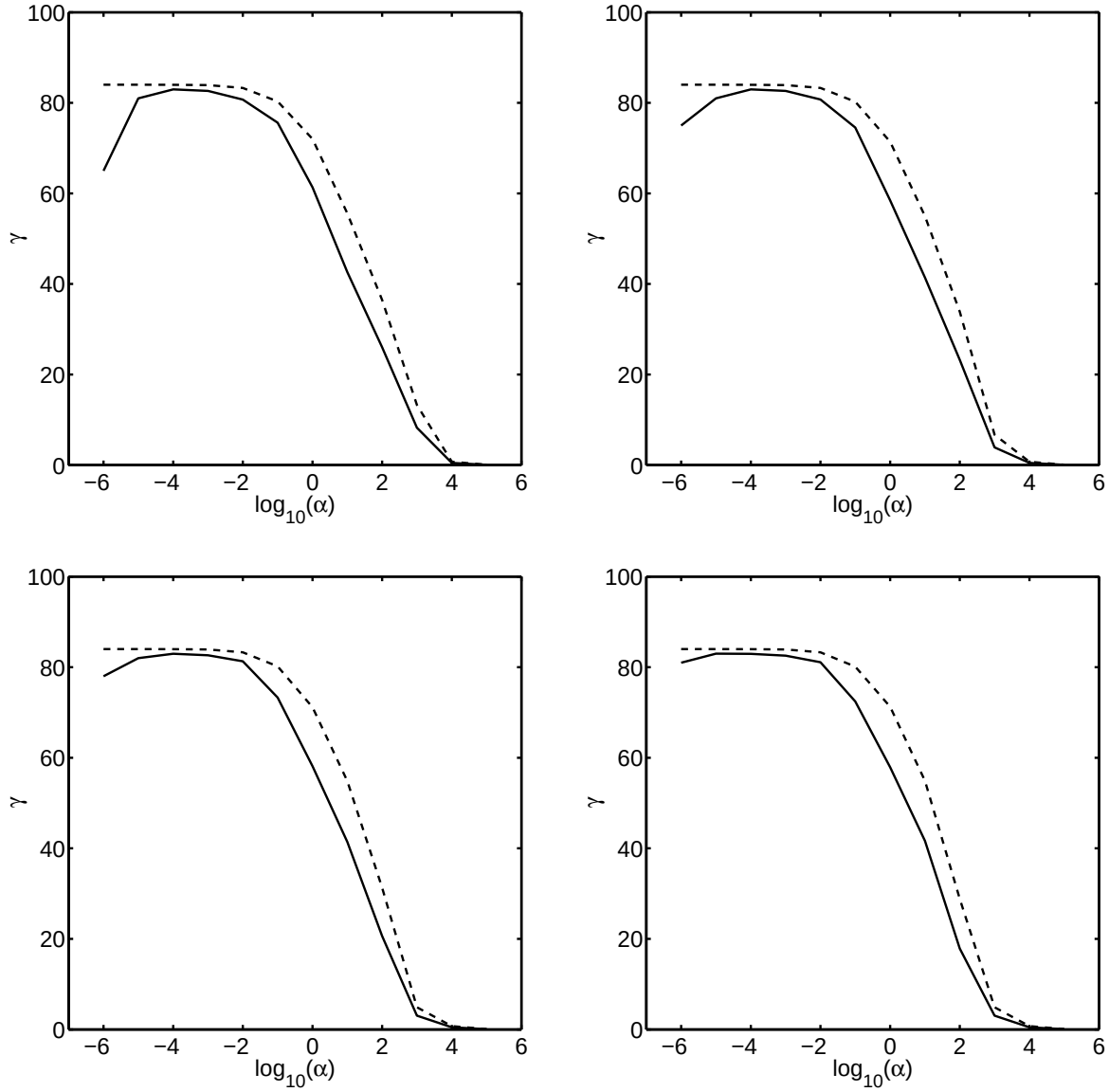


Figure A.2: Plots of γ against $\log_{10} \alpha$ at 0 (i.e. after initialization, but before training has started), 5, 15, and 50 iterations of training. The dashed line shows results from using the approximate Hessian while the solid line shows results obtained using the exact form.