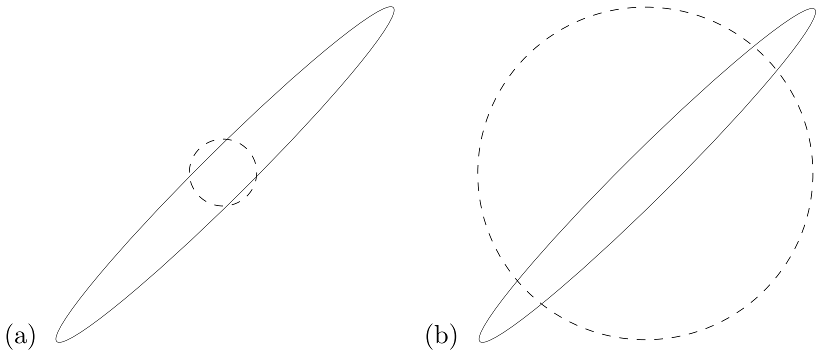

Figure 33.6. Two separable Gaussian approximations (dotted lines) to a bivariate Gaussian distribution (solid line). (a) The approximation that minimizes the variational free energy. (b) The approximation that minimizes the objective function G. In each figure, the lines show the contours at which $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} = 1$, where $\mathbf{A}$ is the inverse covariance matrix of the Gaussian.

In contrast, if we use the objective function G then we find:

$$G(\sigma_Q^2) = \frac{1}{2}\left(\ln \sigma_Q^2 + \frac{\sigma_1^2}{\sigma_Q^2} + \ln \sigma_Q^2 + \frac{\sigma_2^2}{\sigma_Q^2}\right) + \text{constant}, \tag{33.60}$$

where the constant depends on σ_1 and σ_2 only. Differentiating,

$$\frac{\mathrm{d}}{\mathrm{d}\ln \sigma_Q^2} G = \frac{1}{2}\left[2 - \left(\frac{\sigma_1^2}{\sigma_Q^2} + \frac{\sigma_2^2}{\sigma_Q^2}\right)\right], \tag{33.61}$$

which is zero when

$$\sigma_Q^2 = \frac{1}{2}\left(\sigma_1^2 + \sigma_2^2\right). \tag{33.62}$$

Thus we set the approximating distribution's variance to the mean variance of the target distribution P.

In the case $\sigma_1 = 10$ and $\sigma_2 = 1$, we obtain $\sigma_Q \simeq 10/\sqrt{2}$, which is just a factor of $\sqrt{2}$ smaller than σ_1, independent of the value of σ_2.

The two approximations are shown to scale in figure 33.6.

Solution to exercise 33.6 (p.436). The best possible variational approximation is of course the target distribution P. Assuming that this is not possible, a good variational approximation is *more compact* than the true distribution. In contrast, a good sampler is *more heavy tailed* than the true distribution. An over-compact distribution would be a lousy sampler with a large variance.

34

Independent Component Analysis and Latent Variable Modelling

34.1 Latent variable models

Many statistical models are generative models (that is, models that specify a full probability density over all variables in the situation) that make use of *latent variables* to describe a probability distribution over observables.

Examples of latent variable models include Chapter 22's mixture models, which model the observables as coming from a superposed mixture of simple probability distributions (the latent variables are the unknown class labels of the examples); hidden Markov models (Rabiner and Juang, 1986; Durbin *et al.*, 1998); and factor analysis.

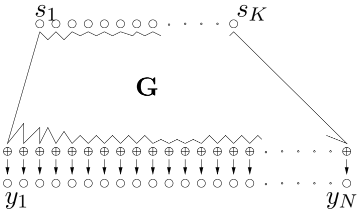

Figure 34.1. Error-correcting codes as latent variable models. The K latent variables are the independent source bits $s_1, \ldots, s_K$; these give rise to the observables via the generator matrix **G**.

The decoding problem for error-correcting codes can also be viewed in terms of a latent variable model – figure 34.1. In that case, the encoding matrix **G** is normally known in advance. In latent variable modelling, the parameters equivalent to **G** are usually not known, and must be inferred from the data along with the latent variables **s**.

Usually, the latent variables have a simple distribution, often a separable distribution. Thus when we fit a latent variable model, we are finding a description of the data in terms of 'independent components'. The 'independent component analysis' algorithm corresponds to perhaps the simplest possible latent variable model with continuous latent variables.

34.2 The generative model for independent component analysis

A set of N observations $D = \{\mathbf{x}^{(n)}\}_{n=1}^{N}$ are assumed to be generated as follows. Each J-dimensional vector $\mathbf{x}$ is a linear mixture of I underlying source signals, $\mathbf{s}$:

$$\mathbf{x} = \mathbf{G}\mathbf{s}, \tag{34.1}$$

where the matrix of mixing coefficients **G** is not known.

The simplest algorithm results if we assume that the number of sources is equal to the number of observations, i.e., $I = J$. Our aim is to recover the source variables $\mathbf{s}$ (within some multiplicative factors, and possibly permuted). To put it another way, we aim to create the inverse of **G** (within a post-multiplicative factor) given only a set of examples $\{\mathbf{x}\}$. We assume that the latent variables are independently distributed, with marginal distributions $P(s_i|\mathcal{H}) \equiv p_i(s_i)$. Here $\mathcal{H}$ denotes the assumed form of this model and the assumed probability distributions p_i of the latent variables.

The probability of the observables and the hidden variables, given **G** and

$\mathcal{H}$, is:

$$P(\{\mathbf{x}^{(n)}, \mathbf{s}^{(n)}\}_{n=1}^{N} \,|\, \mathbf{G}, \mathcal{H}) = \prod_{n=1}^{N} \left[P(\mathbf{x}^{(n)} \,|\, \mathbf{s}^{(n)}, \mathbf{G}, \mathcal{H}) P(\mathbf{s}^{(n)} \,|\, \mathcal{H}) \right] \tag{34.2}$$

$$= \prod_{n=1}^{N} \left[\left(\prod_j \delta\left(x_j^{(n)} - \textstyle\sum_i G_{ji} s_i^{(n)} \right) \right) \left(\prod_i p_i(s_i^{(n)}) \right) \right]. \tag{34.3}$$

We assume that the vector $\mathbf{x}$ is generated *without noise*. This assumption is not usually made in latent variable modelling, since noise-free data are rare; but it makes the inference problem far simpler to solve.

The likelihood function

For learning about $\mathbf{G}$ from the data D, the relevant quantity is the likelihood function

$$P(D \,|\, \mathbf{G}, \mathcal{H}) = \prod_n P(\mathbf{x}^{(n)} \,|\, \mathbf{G}, \mathcal{H}) \tag{34.4}$$

which is a product of factors each of which is obtained by marginalizing over the latent variables. When we marginalize over delta functions, remember that $\int \mathrm{d}s\, \delta(x - vs) f(s) = \frac{1}{v} f(x/v)$. We adopt summation convention at this point, such that, for example, $G_{ji} s_i^{(n)} \equiv \sum_i G_{ji} s_i^{(n)}$. A single factor in the likelihood is given by

$$\begin{aligned}
P(\mathbf{x}^{(n)} \,|\, \mathbf{G}, \mathcal{H}) &= \int \mathrm{d}^I \mathbf{s}^{(n)}\, P(\mathbf{x}^{(n)} \,|\, \mathbf{s}^{(n)}, \mathbf{G}, \mathcal{H}) P(\mathbf{s}^{(n)} \,|\, \mathcal{H}) &(34.5)\\
&= \int \mathrm{d}^I \mathbf{s}^{(n)} \prod_j \delta\left(x_j^{(n)} - G_{ji} s_i^{(n)} \right) \prod_i p_i(s_i^{(n)}) &(34.6)\\
&= \frac{1}{|\det \mathbf{G}|} \prod_i p_i(G_{ij}^{-1} x_j) &(34.7)\\
\Rightarrow \ln P(\mathbf{x}^{(n)} \,|\, \mathbf{G}, \mathcal{H}) &= -\ln |\det \mathbf{G}| + \sum_i \ln p_i(G_{ij}^{-1} x_j). &(34.8)
\end{aligned}$$

To obtain a maximum likelihood algorithm we find the gradient of the log likelihood. If we introduce $\mathbf{W} \equiv \mathbf{G}^{-1}$, the log likelihood contributed by a single example may be written:

$$\ln P(\mathbf{x}^{(n)} \,|\, \mathbf{G}, \mathcal{H}) = \ln |\det \mathbf{W}| + \sum_i \ln p_i(W_{ij} x_j). \tag{34.9}$$

We'll assume from now on that $\det \mathbf{W}$ is positive, so that we can omit the absolute value sign. We will need the following identities:

$$\begin{aligned}
\frac{\partial}{\partial G_{ji}} \ln \det \mathbf{G} &= G_{ij}^{-1} = W_{ij} &(34.10)\\
\frac{\partial}{\partial G_{ji}} G_{lm}^{-1} &= -G_{lj}^{-1} G_{im}^{-1} = -W_{lj} W_{im} &(34.11)\\
\frac{\partial}{\partial W_{ij}} f &= -G_{jm} \left(\frac{\partial}{\partial G_{lm}} f \right) G_{li}. &(34.12)
\end{aligned}$$

Let us define $a_i \equiv W_{ij} x_j$,

$$\phi_i(a_i) \equiv \mathrm{d} \ln p_i(a_i) / \mathrm{d}a_i, \tag{34.13}$$

Repeat for each datapoint $\mathbf{x}$:

1. Put $\mathbf{x}$ through a linear mapping:
$$\mathbf{a} = \mathbf{W}\mathbf{x}.$$
2. Put $\mathbf{a}$ through a nonlinear map:
$$z_i = \phi_i(a_i),$$
where a popular choice for ϕ is $\phi = -\tanh(a_i)$.
3. Adjust the weights in accordance with
$$\Delta\mathbf{W} \propto [\mathbf{W}^{\mathsf{T}}]^{-1} + \mathbf{z}\mathbf{x}^{\mathsf{T}}.$$

Algorithm 34.2. Independent component analysis – online steepest ascents version.
See also algorithm 34.4, which is to be preferred.

and $z_i = \phi_i(a_i)$, which indicates in which direction a_i needs to change to make the probability of the data greater. We may then obtain the gradient with respect to G_{ji} using equations (34.10) and (34.11):

$$\frac{\partial}{\partial G_{ji}} \ln P(\mathbf{x}^{(n)} \,|\, \mathbf{G}, \mathcal{H}) = -W_{ij} - a_i z_{i'} W_{i'j}. \tag{34.14}$$

Or alternatively, the derivative with respect to W_{ij}:

$$\frac{\partial}{\partial W_{ij}} \ln P(\mathbf{x}^{(n)} \,|\, \mathbf{G}, \mathcal{H}) = G_{ji} + x_j z_i. \tag{34.15}$$

If we choose to change $\mathbf{W}$ so as to ascend this gradient, we obtain the learning rule

$$\Delta\mathbf{W} \propto [\mathbf{W}^{\mathsf{T}}]^{-1} + \mathbf{z}\mathbf{x}^{\mathsf{T}}. \tag{34.16}$$

The algorithm so far is summarized in algorithm 34.2.

Choices of ϕ

The choice of the function ϕ defines the assumed prior distribution of the latent variable s.

Let's first consider the *linear* choice $\phi_i(a_i) = -\kappa a_i$, which implicitly (via equation 34.13) assumes a Gaussian distribution on the latent variables. The Gaussian distribution on the latent variables is invariant under rotation of the latent variables, so there can be no evidence favouring any particular alignment of the latent variable space. The linear algorithm is thus uninteresting in that it will never recover the matrix $\mathbf{G}$ or the original sources. Our only hope is thus that the sources are non-Gaussian. Thankfully, most real sources have non-Gaussian distributions; often they have heavier tails than Gaussians.

We thus move on to the popular tanh nonlinearity. If

$$\phi_i(a_i) = -\tanh(a_i) \tag{34.17}$$

then implicitly we are assuming

$$p_i(s_i) \propto 1/\cosh(s_i) \propto \frac{1}{e^{s_i} + e^{-s_i}}. \tag{34.18}$$

This is a heavier-tailed distribution for the latent variables than the Gaussian distribution.

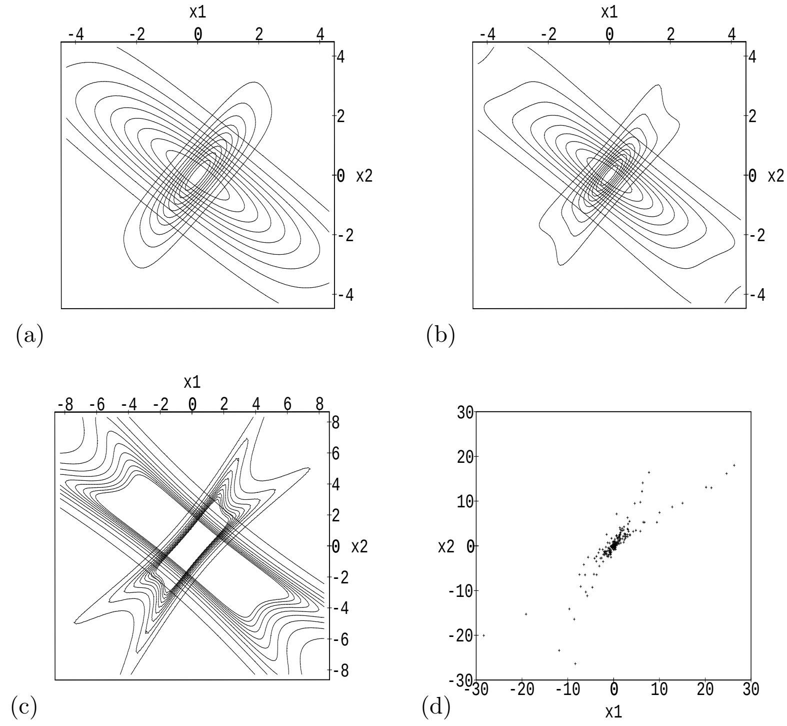

Figure 34.3. Illustration of the generative models implicit in the learning algorithm. (a) Distributions over two observables generated by $1/\cosh$ distributions on the latent variables, for $\mathbf{G} = \left[\begin{array}{cc} 3/4 & 1/2 \\ 1/2 & 1 \end{array}\right]$ (compact distribution) and $\mathbf{G} = \left[\begin{array}{cc} 2 & -1 \\ -1 & 3/2 \end{array}\right]$ (broader distribution). (b) Contours of the generative distributions when the latent variables have Cauchy distributions. The learning algorithm fits this amoeboid object to the empirical data in such a way as to maximize the likelihood. The contour plot in (b) does not adequately represent this heavy-tailed distribution. (c) Part of the tails of the Cauchy distribution, giving the contours $0.01 \ldots 0.1$ times the density at the origin. (d) Some data from one of the generative distributions illustrated in (b) and (c). Can you tell which? 200 samples were created, of which 196 fell in the plotted region.

We could also use a tanh nonlinearity with gain β, that is, $\phi_i(a_i) = -\tanh(\beta a_i)$, whose implicit probabilistic model is $p_i(s_i) \propto 1/[\cosh(\beta s_i)]^{1/\beta}$. In the limit of large β, the nonlinearity becomes a step function and the probability distribution $p_i(s_i)$ becomes a biexponential distribution, $p_i(s_i) \propto \exp(-|s|)$. In the limit $\beta \rightarrow 0$, $p_i(s_i)$ approaches a Gaussian with mean zero and variance $1/\beta$. Heavier-tailed distributions than these may also be used. The Student and Cauchy distributions spring to mind.

Example distributions

Figures 34.3(a–c) illustrate typical distributions generated by the independent components model when the components have $1/\cosh$ and Cauchy distributions. Figure 34.3d shows some samples from the Cauchy model. The Cauchy distribution, being the more heavy-tailed, gives the clearest picture of how the predictive distribution depends on the assumed generative parameters $\mathbf{G}$.

▶ 34.3 A covariant, simpler, and faster learning algorithm

We have thus derived a learning algorithm that performs steepest descents on the likelihood function. The algorithm does not work very quickly, even on toy data; the algorithm is ill-conditioned and illustrates nicely the general advice that, *while finding the gradient of an objective function is a splendid idea, ascending the gradient directly may not be.* The fact that the algorithm is ill-conditioned can be seen in the fact that it involves a matrix inverse, which can be arbitrarily large or even undefined.

Covariant optimization in general

The principle of covariance says that a consistent algorithm should give the same results independent of the units in which quantities are measured (Knuth, 1968). A prime example of a *non*-covariant algorithm is the popular steepest descents rule. A dimensionless objective function $L(\mathbf{w})$ is defined, its derivative with respect to some parameters $\mathbf{w}$ is computed, and then $\mathbf{w}$ is changed by the rule

$$\Delta w_i = \eta \frac{\partial L}{\partial w_i}. \qquad (34.19)$$

This popular equation is dimensionally inconsistent: the left-hand side of this equation has dimensions of $[w_i]$ and the right-hand side has dimensions $1/[w_i]$. The behaviour of the learning algorithm (34.19) is not covariant with respect to linear rescaling of the vector $\mathbf{w}$. Dimensional inconsistency is not the end of the world, as the success of numerous gradient descent algorithms has demonstrated, and indeed if η decreases with n (during on-line learning) as $1/n$ then the Munro–Robbins theorem (Bishop, 1992, p. 41) shows that the parameters will asymptotically converge to the maximum likelihood parameters. But the non-covariant algorithm may take a very large number of iterations to achieve this convergence; indeed many former users of steepest descents algorithms prefer to use algorithms such as conjugate gradients that adaptively figure out the curvature of the objective function. The defense of equation (34.19) that points out η could be a dimensional constant is untenable if not all the parameters w_i have the same dimensions.

The algorithm would be covariant if it had the form

$$\Delta w_i = \eta \sum_{i'} M_{ii'} \frac{\partial L}{\partial w_i}, \qquad (34.20)$$

where $\mathbf{M}$ is a positive-definite matrix whose i, i' element has dimensions $[w_i w_{i'}]$. From where can we obtain such a matrix? Two sources of such matrices are *metrics* and *curvatures.*

Metrics and curvatures

If there is a natural metric that defines *distances* in our parameter space $\mathbf{w}$, then a matrix $\mathbf{M}$ can be obtained from the metric. There is often a natural choice. In the special case where there is a known quadratic metric defining the length of a vector $\mathbf{w}$, then the matrix can be obtained from the quadratic form. For example if the length is $\mathbf{w}^2$ then the natural matrix is $\mathbf{M} = \mathbf{I}$, and steepest descents is appropriate.

Another way of finding a metric is to look at the curvature of the objective function, defining $\mathbf{A} \equiv -\nabla\nabla L$ (where $\nabla \equiv \partial/\partial\mathbf{w}$). Then the matrix $\mathbf{M} = \mathbf{A}^{-1}$ will give a covariant algorithm; what is more, this algorithm is the Newton algorithm, so we recognize that it will alleviate one of the principal difficulties with steepest descents, namely its slow convergence to a minimum when the objective function is at all ill-conditioned. The Newton algorithm converges to the minimum in a single step if L is quadratic.

In some problems it may be that the curvature $\mathbf{A}$ consists of both data-dependent terms and data-independent terms; in this case, one might choose to define the metric using the data-independent terms only (Gull, 1989). The resulting algorithm will still be covariant but it will not implement an exact Newton step. Obviously there are many covariant algorithms; there is no unique choice. But covariant algorithms are a small subset of the set of *all* algorithms!

Back to independent component analysis

For the present maximum likelihood problem we have evaluated the gradient with respect to $\mathbf{G}$ and the gradient with respect to $\mathbf{W} = \mathbf{G}^{-1}$. Steepest ascents in $\mathbf{W}$ is not covariant. Let us construct an alternative, covariant algorithm with the help of the curvature of the log likelihood. Taking the second derivative of the log likelihood with respect to $\mathbf{W}$ we obtain two terms, the first of which is data-independent:

$$\frac{\partial G_{ji}}{\partial W_{kl}} = -G_{jk}G_{li}, \tag{34.21}$$

and the second of which is data-dependent:

$$\frac{\partial (z_i x_j)}{\partial W_{kl}} = x_j x_l \delta_{ik} z_i', \text{(no sum over } i) \tag{34.22}$$

where z' is the derivative of z. It is tempting to drop the data-dependent term and define the matrix $\mathbf{M}$ by $[M^{-1}]_{(ij)(kl)} = [G_{jk}G_{li}]$. However, this matrix is not positive definite (it has at least one non-positive eigenvalue), so it is a poor approximation to the curvature of the log likelihood, which must be positive definite in the neighbourhood of a maximum likelihood solution. We must therefore consult the data-dependent term for inspiration. The aim is to find a convenient approximation to the curvature and to obtain a covariant algorithm, not necessarily to implement an exact Newton step. What is the average value of $x_j x_l \delta_{ik} z_i'$? If the true value of $\mathbf{G}$ is $\mathbf{G}^*$, then

$$\left\langle x_j x_l \delta_{ik} z_i' \right\rangle = \left\langle G^*_{jm} s_m s_n G^*_{ln} \delta_{ik} z_i' \right\rangle . \tag{34.23}$$

We now make several severe approximations: we replace $\mathbf{G}^*$ by the present value of $\mathbf{G}$, and replace the correlated average $\langle s_m s_n z_i' \rangle$ by $\langle s_m s_n \rangle \langle z_i' \rangle \equiv \Sigma_{mn} D_i$. Here $\mathbf{\Sigma}$ is the variance–covariance matrix of the latent variables (which is assumed to exist), and D_i is the typical value of the curvature $\mathrm{d}^2 \ln p_i(a)/\mathrm{d}a^2$. Given that the sources are assumed to be independent, $\mathbf{\Sigma}$ and $\mathbf{D}$ are both diagonal matrices. These approximations motivate the matrix $\mathbf{M}$ given by:

$$[M^{-1}]_{(ij)(kl)} = G_{jm}\Sigma_{mn}G_{ln}\delta_{ik}D_i, \tag{34.24}$$

that is,

$$M_{(ij)(kl)} = W_{mj}\Sigma^{-1}_{mn}W_{nl}\delta_{ik}D_i^{-1}. \tag{34.25}$$

For simplicity, we further assume that the sources are similar to each other so that $\mathbf{\Sigma}$ and $\mathbf{D}$ are both homogeneous and that $\mathbf{\Sigma D} = 1$. This will lead us to an algorithm that is covariant with respect to linear rescaling of the data $\mathbf{x}$, but not with respect to linear rescaling of the latent variables. We thus use:

$$M_{(ij)(kl)} = W_{mj}W_{ml}\delta_{ik}. \tag{34.26}$$

Multiplying this matrix by the gradient in equation (34.15) we obtain the following covariant learning algorithm:

$$\Delta W_{ij} = \eta \left(W_{ij} + W_{i'j} a_{i'} z_i \right). \tag{34.27}$$

Notice that this expression does not require any inversion of the matrix $\mathbf{W}$. The only additional computation once $\mathbf{z}$ has been computed is a single backward pass through the weights to compute the quantity

$$x_j' = W_{i'j} a_{i'} \tag{34.28}$$

Repeat for each datapoint $\mathbf{x}$:

1. Put $\mathbf{x}$ through a linear mapping:

$$\mathbf{a} = \mathbf{W}\mathbf{x}.$$

2. Put $\mathbf{a}$ through a nonlinear map:

$$z_i = \phi_i(a_i),$$

   where a popular choice for ϕ is $\phi = -\tanh(a_i)$.

3. Put $\mathbf{a}$ back through $\mathbf{W}$:

$$\mathbf{x}' = \mathbf{W}^{\mathsf{T}}\mathbf{a}.$$

4. Adjust the weights in accordance with

$$\Delta\mathbf{W} \propto \mathbf{W} + \mathbf{z}\mathbf{x}'^{\mathsf{T}}.$$

Algorithm 34.4. Independent component analysis – covariant version.

in terms of which the covariant algorithm reads:

$$\Delta W_{ij} = \eta \left(W_{ij} + x'_j z_i \right) . \tag{34.29}$$

The quantity $\left(W_{ij} + x'_j z_i \right)$ on the right-hand side is sometimes called the *natural gradient*. The covariant independent component analysis algorithm is summarized in algorithm 34.4.

Further reading

ICA was originally derived using an information maximization approach (Bell and Sejnowski, 1995). Another view of ICA, in terms of energy functions, which motivates more general models, is given by Hinton *et al.* (2001). Another generalization of ICA can be found in Pearlmutter and Parra (1996, 1997).There is now an enormous literature on applications of ICA. A variational free energy minimization approach to ICA-like models is given in (Miskin, 2001; Miskin and MacKay, 2000; Miskin and MacKay, 2001). Further reading on blind separation, including non-ICA algorithms, can be found in (Jutten and Herault, 1991; Comon *et al.*, 1991; Hendin *et al.*, 1994; Amari *et al.*, 1996; Hojen-Sorensen *et al.*, 2002).

Infinite models

While latent variable models with a finite number of latent variables are widely used, it is often the case that our beliefs about the situation would be most accurately captured by a very large number of latent variables.

Consider clustering, for example. If we attack speech recognition by modelling words using a cluster model, how many clusters should we use? The number of possible words is unbounded (section 18.2), so we would really like to use a model in which it's always possible for new clusters to arise.

Furthermore, if we do a careful job of modelling the cluster corresponding to just one English word, we will probably find that the cluster for one word should itself be modelled as composed of clusters – indeed, a hierarchy of clusters within clusters. The first levels of the hierarchy would divide male speakers from female, and would separate speakers from different regions – India, Britain, Europe, and so forth. Within each of those clusters would be subclusters for the different accents within each region. The subclusters could have subsubclusters right down to the level of villages, streets, or families.

Thus we would often like to have infinite numbers of clusters; in some cases the clusters would have a hierarchical structure, and in other cases the hierarchy would be flat. So, how should such infinite models be implemented in finite computers? And how should we set up our Bayesian models so as to avoid getting silly answers?

Infinite mixture models for categorical data are presented in Neal (1991), along with a Monte Carlo method for simulating inferences and predictions. Infinite Gaussian mixture models with a flat hierarchical structure are presented in Rasmussen (2000). Neal (2001) shows how to use Dirichlet diffusion trees to define models of hierarchical clusters. Most of these ideas build on the Dirichlet process (section 18.2). This remains an active research area (Rasmussen and Ghahramani, 2002; Beal *et al.*, 2002).

▶ 34.4 Exercises

Exercise 34.1.[3] Repeat the derivation of the algorithm, but assume a small amount of noise in $\mathbf{x}$: $\mathbf{x} = \mathbf{G}\mathbf{s} + \mathbf{n}$; so the term $\delta\left(x_j^{(n)} - \sum_i G_{ji} s_i^{(n)}\right)$ in the joint probability (34.3) is replaced by a probability distribution over $x_j^{(n)}$ with mean $\sum_i G_{ji} s_i^{(n)}$. Show that, if this noise distribution has sufficiently small standard deviation, the identical algorithm results.

Exercise 34.2.[3] Implement the covariant ICA algorithm and apply it to toy data.

Exercise 34.3.[4-5] Create algorithms appropriate for the situations: (a) $\mathbf{x}$ includes substantial Gaussian noise; (b) more measurements than latent variables ($J > I$); (c) fewer measurements than latent variables ($J < I$).

Factor analysis assumes that the observations $\mathbf{x}$ can be described in terms of independent latent variables $\{s_k\}$ and independent additive noise. Thus the observable $\mathbf{x}$ is given by

$$\mathbf{x} = \mathbf{G}\mathbf{s} + \mathbf{n}, \tag{34.30}$$

where $\mathbf{n}$ is a noise vector whose components have a separable probability distribution. In factor analysis it is often assumed that the probability distributions of $\{s_k\}$ and $\{n_i\}$ are zero-mean Gaussians; the noise terms may have different variances σ_i^2.

Exercise 34.4.[4] Make a maximum likelihood algorithm for inferring $\mathbf{G}$ from data, assuming the generative model $\mathbf{x} = \mathbf{G}\mathbf{s} + \mathbf{n}$ is correct and that $\mathbf{s}$ and $\mathbf{n}$ have independent Gaussian distributions. Include parameters σ_j^2 to describe the variance of each n_j, and maximize the likelihood with respect to them too. Let the variance of each s_i be 1.

Exercise 34.5.[4C] Implement the infinite Gaussian mixture model of Rasmussen (2000).