

3 Linear Models for Regression

The focus so far in this book has been on unsupervised learning, including topics such as density estimation and data clustering. We turn now to a discussion of supervised learning, starting with regression. The goal of regression is to predict the value of one or more continuous *target* variables t given the value of a D -dimensional vector \mathbf{x} of *input* variables. We have already encountered an example of a regression problem when we considered polynomial curve fitting in Chapter 1. The polynomial is a specific example of a broad class of functions called linear regression models, which share the property of being linear functions of the adjustable parameters, and which will form the focus of this chapter. The simplest form of linear regression models are also linear functions of the input variables. However, we can obtain a much more useful class of functions by taking linear combinations of a fixed set of nonlinear functions of the input variables, known as *basis functions*. Such models are linear functions of the parameters, which gives them simple analytical properties, and yet can be nonlinear with respect to the input variables.

Given a training data set comprising N observations $\{\mathbf{x}_n\}$, where $n = 1, \dots, N$, together with corresponding target values $\{t_n\}$, the goal is to predict the value of t for a new value of \mathbf{x} . In the simplest approach, this can be done by directly constructing an appropriate function $y(\mathbf{x})$ whose values for new inputs \mathbf{x} constitute the predictions for the corresponding values of t . More generally, from a probabilistic perspective, we aim to model the predictive distribution $p(t|\mathbf{x})$ because this expresses our uncertainty about the value of t for each value of \mathbf{x} . From this conditional distribution we can make predictions of t , for any new value of \mathbf{x} , in such a way as to minimize the expected value of a suitably chosen loss function. As discussed in Section 1.5.5, a common choice of loss function for real-valued variables is the squared loss, for which the optimal solution is given by the conditional expectation of t .

Although linear models have significant limitations as practical techniques for pattern recognition, particularly for problems involving input spaces of high dimensionality, they have nice analytical properties and form the foundation for more sophisticated models to be discussed in later chapters.

3.1. Linear Basis Function Models

The simplest linear model for regression is one that involves a linear combination of the input variables

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \dots + w_D x_D \quad (3.1)$$

where $\mathbf{x} = (x_1, \dots, x_D)^T$. This is often simply known as *linear regression*. The key property of this model is that it is a linear function of the parameters w_0, \dots, w_D . It is also, however, a linear function of the input variables x_i , and this imposes significant limitations on the model. We therefore extend the class of models by considering linear combinations of fixed nonlinear functions of the input variables, of the form

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) \quad (3.2)$$

where $\phi_j(\mathbf{x})$ are known as *basis functions*. By denoting the maximum value of the index j by $M - 1$, the total number of parameters in this model will be M .

The parameter w_0 allows for any fixed offset in the data and is sometimes called a *bias* parameter (not to be confused with ‘bias’ in a statistical sense). It is often convenient to define an additional dummy ‘basis function’ $\phi_0(\mathbf{x}) = 1$ so that

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \quad (3.3)$$

where $\mathbf{w} = (w_0, \dots, w_{M-1})^T$ and $\boldsymbol{\phi} = (\phi_0, \dots, \phi_{M-1})^T$. In many practical applications of pattern recognition, we will apply some form of fixed pre-processing,

or feature extraction, to the original data variables. If the original variables comprise the vector \mathbf{x} , then the features can be expressed in terms of the basis functions $\{\phi_j(\mathbf{x})\}$.

By using nonlinear basis functions, we allow the function $y(\mathbf{x}, \mathbf{w})$ to be a nonlinear function of the input vector \mathbf{x} . Functions of the form (3.2) are called linear models, however, because this function is linear in \mathbf{w} . It is this linearity in the parameters that will greatly simplify the analysis of this class of models. However, it also leads to some significant limitations, as we discuss in Section 3.6.

The example of polynomial regression considered in Chapter 1 is a particular example of this model in which there is a single input variable x , and the basis functions take the form of powers of x so that $\phi_j(x) = x^j$. One limitation of polynomial basis functions is that they are global functions of the input variable, so that changes in one region of input space affect all other regions. This can be resolved by dividing the input space up into regions and fit a different polynomial in each region, leading to *spline functions* (Hastie *et al.*, 2001).

There are many other possible choices for the basis functions, for example

$$\phi_j(x) = \exp \left\{ -\frac{(x - \mu_j)^2}{2s^2} \right\} \quad (3.4)$$

where the μ_j govern the locations of the basis functions in input space, and the parameter s governs their spatial scale. These are usually referred to as ‘Gaussian’ basis functions, although it should be noted that they are not required to have a probabilistic interpretation, and in particular the normalization coefficient is unimportant because these basis functions will be multiplied by adaptive parameters w_j .

Another possibility is the sigmoidal basis function of the form

$$\phi_j(x) = \sigma \left(\frac{x - \mu_j}{s} \right) \quad (3.5)$$

where $\sigma(a)$ is the logistic sigmoid function defined by

$$\sigma(a) = \frac{1}{1 + \exp(-a)}. \quad (3.6)$$

Equivalently, we can use the ‘tanh’ function because this is related to the logistic sigmoid by $\tanh(a) = 2\sigma(a) - 1$, and so a general linear combination of logistic sigmoid functions is equivalent to a general linear combination of ‘tanh’ functions. These various choices of basis function are illustrated in Figure 3.1.

Yet another possible choice of basis function is the Fourier basis, which leads to an expansion in sinusoidal functions. Each basis function represents a specific frequency and has infinite spatial extent. By contrast, basis functions that are localized to finite regions of input space necessarily comprise a spectrum of different spatial frequencies. In many signal processing applications, it is of interest to consider basis functions that are localized in both space and frequency, leading to a class of functions known as *wavelets*. These are also defined to be mutually orthogonal, to simplify their application. Wavelets are most applicable when the input values live

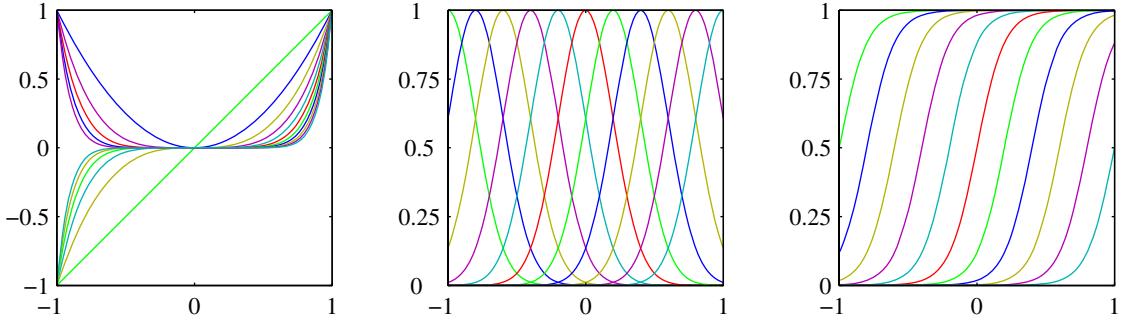


Figure 3.1 Examples of basis functions, showing polynomials on the left, Gaussians of the form (3.4) in the centre, and sigmoidal of the form (3.5) on the right.

on a regular lattice, such as the successive time points in a temporal sequence, or the pixels in an image. Useful texts on wavelets include Ogden (1997), Mallat (1999), and Vidakovic (1999).

Most of the discussion in this chapter, however, is independent of the particular choice of basis function set, and so for most of our discussion we shall not specify the particular form of the basis functions, except for the purposes of numerical illustration. Indeed, much of our discussion will be equally applicable to the situation in which the vector $\phi(\mathbf{x})$ of basis functions is simply the identity $\phi(\mathbf{x}) = \mathbf{x}$. Furthermore, in order to keep the notation simple, we shall focus on the case of a single target variable t . However, in Section 3.1.5, we consider briefly the modifications needed to deal with multiple target variables.

3.1.1 Maximum likelihood and least squares

In Chapter 1, we fitted polynomial functions to data sets by minimizing a sum-of-squares error function. We also showed that this error function could be motivated as the maximum likelihood solution under an assumed Gaussian noise model. Let us return to this discussion and consider the least squares approach, and its relation to maximum likelihood, in more detail.

As before, we assume that the target variable t is given by a deterministic function $y(\mathbf{x}, \mathbf{w})$ with additive Gaussian noise so that

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \quad (3.7)$$

where ϵ is a zero mean Gaussian random variable with precision (inverse variance) β . Thus we can write

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}). \quad (3.8)$$

Recall that, if we assume a squared loss function, then the optimal prediction, for a new value of \mathbf{x} , will be given by the conditional mean of the target variable. In the case of a Gaussian conditional distribution of the form (3.8), the conditional mean

will be simply

$$\mathbb{E}[t|\mathbf{x}] = \int tp(t|\mathbf{x}) dt = y(\mathbf{x}, \mathbf{w}). \quad (3.9)$$

Note that the Gaussian noise assumption implies that the conditional distribution of t given \mathbf{x} is unimodal, which may be inappropriate for some applications. An extension to mixtures of conditional Gaussian distributions, which permit multimodal conditional distributions, will be discussed in Section 14.5.1.

Now consider a data set of inputs $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ with corresponding target values t_1, \dots, t_N . We group the target variables $\{t_n\}$ into a column vector that we denote by \mathbf{t} where the typeface is chosen to distinguish it from a single observation of a multivariate target, which would be denoted \mathbf{t} . Making the assumption that these data points are drawn independently from the distribution (3.8), we obtain the following expression for the likelihood function, which is a function of the adjustable parameters \mathbf{w} and β , in the form

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n | \mathbf{w}^T \phi(\mathbf{x}_n), \beta^{-1}) \quad (3.10)$$

where we have used (3.3). Note that in supervised learning problems such as regression (and classification), we are not seeking to model the distribution of the input variables. Thus \mathbf{x} will always appear in the set of conditioning variables, and so from now on we will drop the explicit \mathbf{x} from expressions such as $p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)$ in order to keep the notation uncluttered. Taking the logarithm of the likelihood function, and making use of the standard form (1.46) for the univariate Gaussian, we have

$$\begin{aligned} \ln p(\mathbf{t}|\mathbf{w}, \beta) &= \sum_{n=1}^N \ln \mathcal{N}(t_n | \mathbf{w}^T \phi(\mathbf{x}_n), \beta^{-1}) \\ &= \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta E_D(\mathbf{w}) \end{aligned} \quad (3.11)$$

where the sum-of-squares error function is defined by

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2. \quad (3.12)$$

Having written down the likelihood function, we can use maximum likelihood to determine \mathbf{w} and β . Consider first the maximization with respect to \mathbf{w} . As observed already in Section 1.2.5, we see that maximization of the likelihood function under a conditional Gaussian noise distribution for a linear model is equivalent to minimizing a sum-of-squares error function given by $E_D(\mathbf{w})$. The gradient of the log likelihood function (3.11) takes the form

$$\nabla \ln p(\mathbf{t}|\mathbf{w}, \beta) = \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\} \phi(\mathbf{x}_n)^T. \quad (3.13)$$

Setting this gradient to zero gives

$$0 = \sum_{n=1}^N t_n \phi(\mathbf{x}_n)^T - \mathbf{w}^T \left(\sum_{n=1}^N \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^T \right). \quad (3.14)$$

Solving for \mathbf{w} we obtain

$$\mathbf{w}_{\text{ML}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t} \quad (3.15)$$

which are known as the *normal equations* for the least squares problem. Here Φ is an $N \times M$ matrix, called the *design matrix*, whose elements are given by $\Phi_{nj} = \phi_j(\mathbf{x}_n)$, so that

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}. \quad (3.16)$$

The quantity

$$\Phi^\dagger \equiv (\Phi^T \Phi)^{-1} \Phi^T \quad (3.17)$$

is known as the *Moore-Penrose pseudo-inverse* of the matrix Φ (Rao and Mitra, 1971; Golub and Van Loan, 1996). It can be regarded as a generalization of the notion of matrix inverse to nonsquare matrices. Indeed, if Φ is square and invertible, then using the property $(\mathbf{AB})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$ we see that $\Phi^\dagger \equiv \Phi^{-1}$.

At this point, we can gain some insight into the role of the bias parameter w_0 . If we make the bias parameter explicit, then the error function (3.12) becomes

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{t_n - w_0 - \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}_n)\}^2. \quad (3.18)$$

Setting the derivative with respect to w_0 equal to zero, and solving for w_0 , we obtain

$$w_0 = \bar{t} - \sum_{j=1}^{M-1} w_j \bar{\phi}_j \quad (3.19)$$

where we have defined

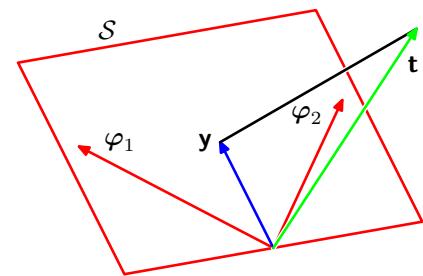
$$\bar{t} = \frac{1}{N} \sum_{n=1}^N t_n, \quad \bar{\phi}_j = \frac{1}{N} \sum_{n=1}^N \phi_j(\mathbf{x}_n). \quad (3.20)$$

Thus the bias w_0 compensates for the difference between the averages (over the training set) of the target values and the weighted sum of the averages of the basis function values.

We can also maximize the log likelihood function (3.11) with respect to the noise precision parameter β , giving

$$\frac{1}{\beta_{\text{ML}}} = \frac{1}{N} \sum_{n=1}^N \{t_n - \mathbf{w}_{\text{ML}}^T \phi(\mathbf{x}_n)\}^2 \quad (3.21)$$

Figure 3.2 Geometrical interpretation of the least-squares solution, in an N -dimensional space whose axes are the values of t_1, \dots, t_N . The least-squares regression function is obtained by finding the orthogonal projection of the data vector \mathbf{t} onto the subspace spanned by the basis functions $\phi_j(\mathbf{x})$ in which each basis function is viewed as a vector φ_j of length N with elements $\phi_j(\mathbf{x}_n)$.



and so we see that the inverse of the noise precision is given by the residual variance of the target values around the regression function.

3.1.2 Geometry of least squares

At this point, it is instructive to consider the geometrical interpretation of the least-squares solution. To do this we consider an N -dimensional space whose axes are given by the t_n , so that $\mathbf{t} = (t_1, \dots, t_N)^T$ is a vector in this space. Each basis function $\phi_j(\mathbf{x}_n)$, evaluated at the N data points, can also be represented as a vector in the same space, denoted by φ_j , as illustrated in Figure 3.2. Note that φ_j corresponds to the j^{th} column of Φ , whereas $\phi(\mathbf{x}_n)$ corresponds to the n^{th} row of Φ . If the number M of basis functions is smaller than the number N of data points, then the M vectors $\varphi_j(\mathbf{x}_n)$ will span a linear subspace S of dimensionality M . We define \mathbf{y} to be an N -dimensional vector whose n^{th} element is given by $y(\mathbf{x}_n, \mathbf{w})$, where $n = 1, \dots, N$. Because \mathbf{y} is an arbitrary linear combination of the vectors φ_j , it can live anywhere in the M -dimensional subspace. The sum-of-squares error (3.12) is then equal (up to a factor of $1/2$) to the squared Euclidean distance between \mathbf{y} and \mathbf{t} . Thus the least-squares solution for \mathbf{w} corresponds to that choice of \mathbf{y} that lies in subspace S and that is closest to \mathbf{t} . Intuitively, from Figure 3.2, we anticipate that this solution corresponds to the orthogonal projection of \mathbf{t} onto the subspace S . This is indeed the case, as can easily be verified by noting that the solution for \mathbf{y} is given by $\Phi \mathbf{w}_{\text{ML}}$, and then confirming that this takes the form of an orthogonal projection.

In practice, a direct solution of the normal equations can lead to numerical difficulties when $\Phi^T \Phi$ is close to singular. In particular, when two or more of the basis vectors φ_j are co-linear, or nearly so, the resulting parameter values can have large magnitudes. Such near degeneracies will not be uncommon when dealing with real data sets. The resulting numerical difficulties can be addressed using the technique of *singular value decomposition*, or *SVD* (Press *et al.*, 1992; Bishop and Nabney, 2008). Note that the addition of a regularization term ensures that the matrix is non-singular, even in the presence of degeneracies.

3.1.3 Sequential learning

Batch techniques, such as the maximum likelihood solution (3.15), which involve processing the entire training set in one go, can be computationally costly for large data sets. As we have discussed in Chapter 1, if the data set is sufficiently large, it may be worthwhile to use *sequential* algorithms, also known as *on-line* algorithms,

Exercise 3.2

in which the data points are considered one at a time, and the model parameters updated after each such presentation. Sequential learning is also appropriate for real-time applications in which the data observations are arriving in a continuous stream, and predictions must be made before all of the data points are seen.

We can obtain a sequential learning algorithm by applying the technique of *stochastic gradient descent*, also known as *sequential gradient descent*, as follows. If the error function comprises a sum over data points $E = \sum_n E_n$, then after presentation of pattern n , the stochastic gradient descent algorithm updates the parameter vector \mathbf{w} using

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n \quad (3.22)$$

where τ denotes the iteration number, and η is a learning rate parameter. We shall discuss the choice of value for η shortly. The value of \mathbf{w} is initialized to some starting vector $\mathbf{w}^{(0)}$. For the case of the sum-of-squares error function (3.12), this gives

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \eta(t_n - \mathbf{w}^{(\tau)\top} \phi_n) \phi_n \quad (3.23)$$

where $\phi_n = \phi(\mathbf{x}_n)$. This is known as *least-mean-squares* or the *LMS algorithm*. The value of η needs to be chosen with care to ensure that the algorithm converges (Bishop and Nabney, 2008).

3.1.4 Regularized least squares

In Section 1.1, we introduced the idea of adding a regularization term to an error function in order to control over-fitting, so that the total error function to be minimized takes the form

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w}) \quad (3.24)$$

where λ is the regularization coefficient that controls the relative importance of the data-dependent error $E_D(\mathbf{w})$ and the regularization term $E_W(\mathbf{w})$. One of the simplest forms of regularizer is given by the sum-of-squares of the weight vector elements

$$E_W(\mathbf{w}) = \frac{1}{2} \mathbf{w}^\top \mathbf{w}. \quad (3.25)$$

If we also consider the sum-of-squares error function given by

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^\top \phi(\mathbf{x}_n)\}^2 \quad (3.26)$$

then the total error function becomes

$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^\top \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \mathbf{w}^\top \mathbf{w}. \quad (3.27)$$

This particular choice of regularizer is known in the machine learning literature as *weight decay* because in sequential learning algorithms, it encourages weight values to decay towards zero, unless supported by the data. In statistics, it provides an example of a *parameter shrinkage* method because it shrinks parameter values towards

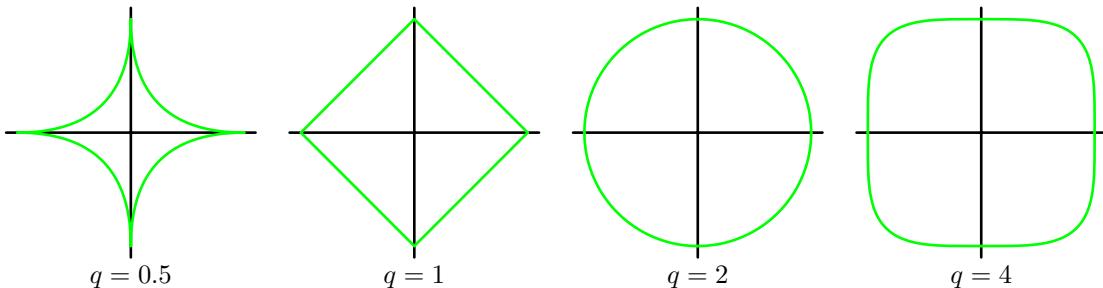


Figure 3.3 Contours of the regularization term in (3.29) for various values of the parameter q .

zero. It has the advantage that the error function remains a quadratic function of \mathbf{w} , and so its exact minimizer can be found in closed form. Specifically, setting the gradient of (3.27) with respect to \mathbf{w} to zero, and solving for \mathbf{w} as before, we obtain

$$\mathbf{w} = (\lambda \mathbf{I} + \Phi^T \Phi)^{-1} \Phi^T \mathbf{t}. \quad (3.28)$$

This represents a simple extension of the least-squares solution (3.15).

A more general regularizer is sometimes used, for which the regularized error takes the form

$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q \quad (3.29)$$

where $q = 2$ corresponds to the quadratic regularizer (3.27). Figure 3.3 shows contours of the regularization function for different values of q .

The case of $q = 1$ is known as the *lasso* in the statistics literature (Tibshirani, 1996). It has the property that if λ is sufficiently large, some of the coefficients w_j are driven to zero, leading to a *sparse* model in which the corresponding basis functions play no role. To see this, we first note that minimizing (3.29) is equivalent to minimizing the unregularized sum-of-squares error (3.12) subject to the constraint

$$\sum_{j=1}^M |w_j|^q \leq \eta \quad (3.30)$$

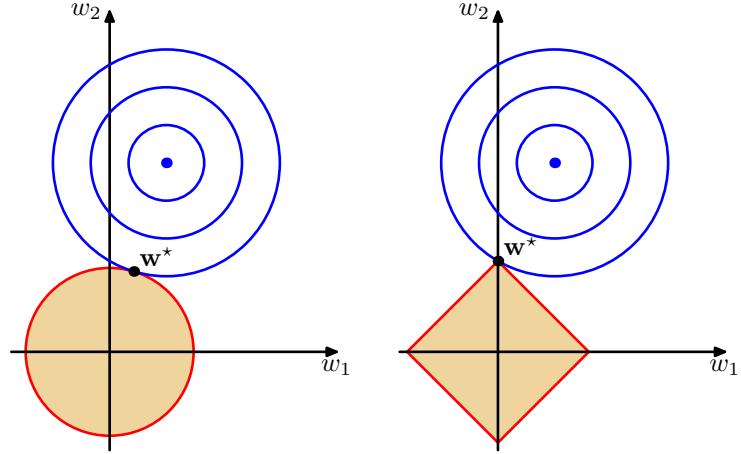
for an appropriate value of the parameter η , where the two approaches can be related using Lagrange multipliers. The origin of the sparsity can be seen from Figure 3.4, which shows that the minimum of the error function, subject to the constraint (3.30). As λ is increased, so an increasing number of parameters are driven to zero.

Regularization allows complex models to be trained on data sets of limited size without severe over-fitting, essentially by limiting the effective model complexity. However, the problem of determining the optimal model complexity is then shifted from one of finding the appropriate number of basis functions to one of determining a suitable value of the regularization coefficient λ . We shall return to the issue of model complexity later in this chapter.

Exercise 3.5

Appendix E

Figure 3.4 Plot of the contours of the unregularized error function (blue) along with the constraint region (3.30) for the quadratic regularizer $q = 2$ on the left and the lasso regularizer $q = 1$ on the right, in which the optimum value for the parameter vector w is denoted by w^* . The lasso gives a sparse solution in which $w_1^* = 0$.



For the remainder of this chapter we shall focus on the quadratic regularizer (3.27) both for its practical importance and its analytical tractability.

3.1.5 Multiple outputs

So far, we have considered the case of a single target variable t . In some applications, we may wish to predict $K > 1$ target variables, which we denote collectively by the target vector t . This could be done by introducing a different set of basis functions for each component of t , leading to multiple, independent regression problems. However, a more interesting, and more common, approach is to use the same set of basis functions to model all of the components of the target vector so that

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = \mathbf{W}^T \phi(\mathbf{x}) \quad (3.31)$$

where \mathbf{y} is a K -dimensional column vector, \mathbf{W} is an $M \times K$ matrix of parameters, and $\phi(\mathbf{x})$ is an M -dimensional column vector with elements $\phi_j(\mathbf{x})$, with $\phi_0(\mathbf{x}) = 1$ as before. Suppose we take the conditional distribution of the target vector to be an isotropic Gaussian of the form

$$p(t|\mathbf{x}, \mathbf{W}, \beta) = \mathcal{N}(t|\mathbf{W}^T \phi(\mathbf{x}), \beta^{-1} \mathbf{I}). \quad (3.32)$$

If we have a set of observations $\mathbf{t}_1, \dots, \mathbf{t}_N$, we can combine these into a matrix \mathbf{T} of size $N \times K$ such that the n^{th} row is given by \mathbf{t}_n^T . Similarly, we can combine the input vectors $\mathbf{x}_1, \dots, \mathbf{x}_N$ into a matrix \mathbf{X} . The log likelihood function is then given by

$$\begin{aligned} \ln p(\mathbf{T}|\mathbf{X}, \mathbf{W}, \beta) &= \sum_{n=1}^N \ln \mathcal{N}(\mathbf{t}_n|\mathbf{W}^T \phi(\mathbf{x}_n), \beta^{-1} \mathbf{I}) \\ &= \frac{NK}{2} \ln \left(\frac{\beta}{2\pi} \right) - \frac{\beta}{2} \sum_{n=1}^N \|\mathbf{t}_n - \mathbf{W}^T \phi(\mathbf{x}_n)\|^2. \end{aligned} \quad (3.33)$$

As before, we can maximize this function with respect to \mathbf{W} , giving

$$\mathbf{W}_{\text{ML}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{T}. \quad (3.34)$$

If we examine this result for each target variable t_k , we have

$$\mathbf{w}_k = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}_k = \Phi^\dagger \mathbf{t}_k \quad (3.35)$$

where \mathbf{t}_k is an N -dimensional column vector with components t_{nk} for $n = 1, \dots, N$. Thus the solution to the regression problem decouples between the different target variables, and we need only compute a single pseudo-inverse matrix Φ^\dagger , which is shared by all of the vectors \mathbf{w}_k .

The extension to general Gaussian noise distributions having arbitrary covariance matrices is straightforward. Again, this leads to a decoupling into K independent regression problems. This result is unsurprising because the parameters \mathbf{W} define only the mean of the Gaussian noise distribution, and we know from Section 2.3.4 that the maximum likelihood solution for the mean of a multivariate Gaussian is independent of the covariance. From now on, we shall therefore consider a single target variable t for simplicity.

3.2. The Bias-Variance Decomposition

So far in our discussion of linear models for regression, we have assumed that the form and number of basis functions are both fixed. As we have seen in Chapter 1, the use of maximum likelihood, or equivalently least squares, can lead to severe over-fitting if complex models are trained using data sets of limited size. However, limiting the number of basis functions in order to avoid over-fitting has the side effect of limiting the flexibility of the model to capture interesting and important trends in the data. Although the introduction of regularization terms can control over-fitting for models with many parameters, this raises the question of how to determine a suitable value for the regularization coefficient λ . Seeking the solution that minimizes the regularized error function with respect to both the weight vector \mathbf{w} and the regularization coefficient λ is clearly not the right approach since this leads to the unregularized solution with $\lambda = 0$.

As we have seen in earlier chapters, the phenomenon of over-fitting is really an unfortunate property of maximum likelihood and does not arise when we marginalize over parameters in a Bayesian setting. In this chapter, we shall consider the Bayesian view of model complexity in some depth. Before doing so, however, it is instructive to consider a frequentist viewpoint of the model complexity issue, known as the *bias-variance* trade-off. Although we shall introduce this concept in the context of linear basis function models, where it is easy to illustrate the ideas using simple examples, the discussion has more general applicability.

In Section 1.5.5, when we discussed decision theory for regression problems, we considered various loss functions each of which leads to a corresponding optimal prediction once we are given the conditional distribution $p(t|\mathbf{x})$. A popular choice is

Exercise 3.6

the squared loss function, for which the optimal prediction is given by the conditional expectation, which we denote by $h(\mathbf{x})$ and which is given by

$$h(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] = \int tp(t|\mathbf{x}) dt. \quad (3.36)$$

At this point, it is worth distinguishing between the squared loss function arising from decision theory and the sum-of-squares error function that arose in the maximum likelihood estimation of model parameters. We might use more sophisticated techniques than least squares, for example regularization or a fully Bayesian approach, to determine the conditional distribution $p(t|\mathbf{x})$. These can all be combined with the squared loss function for the purpose of making predictions.

We showed in Section 1.5.5 that the expected squared loss can be written in the form

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - h(\mathbf{x})\}^2 p(\mathbf{x}) d\mathbf{x} + \int \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt. \quad (3.37)$$

Recall that the second term, which is independent of $y(\mathbf{x})$, arises from the intrinsic noise on the data and represents the minimum achievable value of the expected loss. The first term depends on our choice for the function $y(\mathbf{x})$, and we will seek a solution for $y(\mathbf{x})$ which makes this term a minimum. Because it is nonnegative, the smallest that we can hope to make this term is zero. If we had an unlimited supply of data (and unlimited computational resources), we could in principle find the regression function $h(\mathbf{x})$ to any desired degree of accuracy, and this would represent the optimal choice for $y(\mathbf{x})$. However, in practice we have a data set \mathcal{D} containing only a finite number N of data points, and consequently we do not know the regression function $h(\mathbf{x})$ exactly.

If we model the $h(\mathbf{x})$ using a parametric function $y(\mathbf{x}, \mathbf{w})$ governed by a parameter vector \mathbf{w} , then from a Bayesian perspective the uncertainty in our model is expressed through a posterior distribution over \mathbf{w} . A frequentist treatment, however, involves making a point estimate of \mathbf{w} based on the data set \mathcal{D} , and tries instead to interpret the uncertainty of this estimate through the following thought experiment. Suppose we had a large number of data sets each of size N and each drawn independently from the distribution $p(t, \mathbf{x})$. For any given data set \mathcal{D} , we can run our learning algorithm and obtain a prediction function $y(\mathbf{x}; \mathcal{D})$. Different data sets from the ensemble will give different functions and consequently different values of the squared loss. The performance of a particular learning algorithm is then assessed by taking the average over this ensemble of data sets.

Consider the integrand of the first term in (3.37), which for a particular data set \mathcal{D} takes the form

$$\{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2. \quad (3.38)$$

Because this quantity will be dependent on the particular data set \mathcal{D} , we take its average over the ensemble of data sets. If we add and subtract the quantity $\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]$

inside the braces, and then expand, we obtain

$$\begin{aligned} & \{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2 \\ &= \{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2 + \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2 \\ &\quad + 2\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}. \end{aligned} \quad (3.39)$$

We now take the expectation of this expression with respect to \mathcal{D} and note that the final term will vanish, giving

$$\begin{aligned} & \mathbb{E}_{\mathcal{D}} [\{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2] \\ &= \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2}_{\text{(bias)}^2} + \underbrace{\mathbb{E}_{\mathcal{D}} [\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2]}_{\text{variance}}. \end{aligned} \quad (3.40)$$

We see that the expected squared difference between $y(\mathbf{x}; \mathcal{D})$ and the regression function $h(\mathbf{x})$ can be expressed as the sum of two terms. The first term, called the squared *bias*, represents the extent to which the average prediction over all data sets differs from the desired regression function. The second term, called the *variance*, measures the extent to which the solutions for individual data sets vary around their average, and hence this measures the extent to which the function $y(\mathbf{x}; \mathcal{D})$ is sensitive to the particular choice of data set. We shall provide some intuition to support these definitions shortly when we consider a simple example.

So far, we have considered a single input value \mathbf{x} . If we substitute this expansion back into (3.37), we obtain the following decomposition of the expected squared loss

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise} \quad (3.41)$$

where

$$(\text{bias})^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2 p(\mathbf{x}) d\mathbf{x} \quad (3.42)$$

$$\text{variance} = \int \mathbb{E}_{\mathcal{D}} [\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2] p(\mathbf{x}) d\mathbf{x} \quad (3.43)$$

$$\text{noise} = \int \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt \quad (3.44)$$

and the bias and variance terms now refer to integrated quantities.

Our goal is to minimize the expected loss, which we have decomposed into the sum of a (squared) bias, a variance, and a constant noise term. As we shall see, there is a trade-off between bias and variance, with very flexible models having low bias and high variance, and relatively rigid models having high bias and low variance. The model with the optimal predictive capability is the one that leads to the best balance between bias and variance. This is illustrated by considering the sinusoidal data set from Chapter 1. Here we generate 100 data sets, each containing $N = 25$ data points, independently from the sinusoidal curve $h(x) = \sin(2\pi x)$. The data sets are indexed by $l = 1, \dots, L$, where $L = 100$, and for each data set $\mathcal{D}^{(l)}$ we

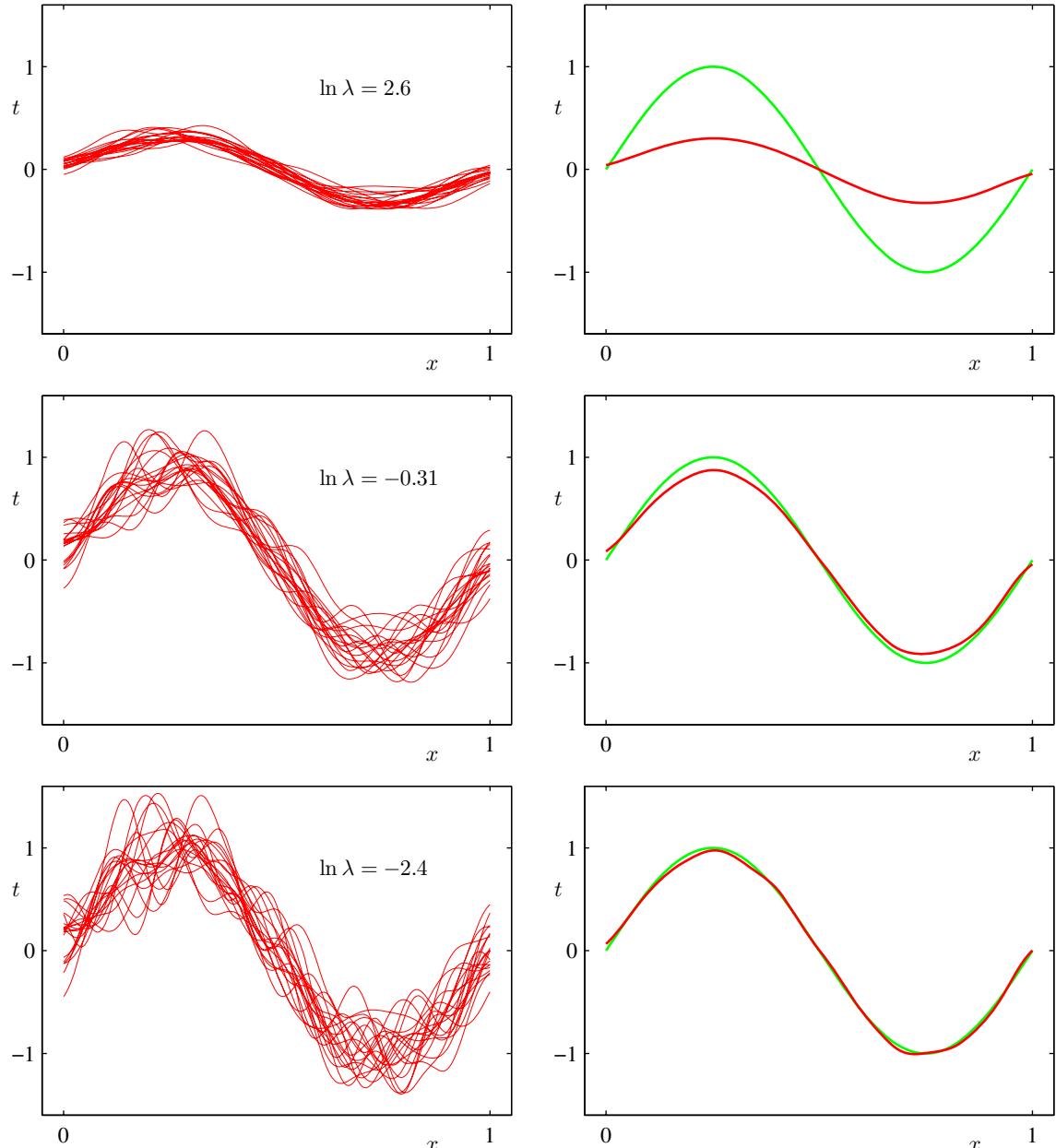
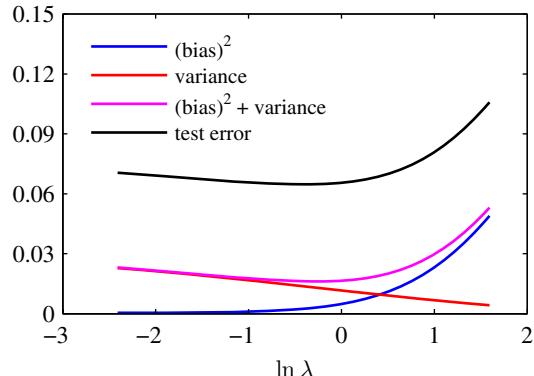


Figure 3.5 Illustration of the dependence of bias and variance on model complexity, governed by a regularization parameter λ , using the sinusoidal data set from Chapter 1. There are $L = 100$ data sets, each having $N = 25$ data points, and there are 24 Gaussian basis functions in the model so that the total number of parameters is $M = 25$ including the bias parameter. The left column shows the result of fitting the model to the data sets for various values of $\ln \lambda$ (for clarity, only 20 of the 100 fits are shown). The right column shows the corresponding average of the 100 fits (red) along with the sinusoidal function from which the data sets were generated (green).

Figure 3.6 Plot of squared bias and variance, together with their sum, corresponding to the results shown in Figure 3.5. Also shown is the average test set error for a test data set size of 1000 points. The minimum value of $(\text{bias})^2 + \text{variance}$ occurs around $\ln \lambda = -0.31$, which is close to the value that gives the minimum error on the test data.



fit a model with 24 Gaussian basis functions by minimizing the regularized error function (3.27) to give a prediction function $y^{(l)}(x)$ as shown in Figure 3.5. The top row corresponds to a large value of the regularization coefficient λ that gives low variance (because the red curves in the left plot look similar) but high bias (because the two curves in the right plot are very different). Conversely on the bottom row, for which λ is small, there is large variance (shown by the high variability between the red curves in the left plot) but low bias (shown by the good fit between the average model fit and the original sinusoidal function). Note that the result of averaging many solutions for the complex model with $M = 25$ is a very good fit to the regression function, which suggests that averaging may be a beneficial procedure. Indeed, a weighted averaging of multiple solutions lies at the heart of a Bayesian approach, although the averaging is with respect to the posterior distribution of parameters, not with respect to multiple data sets.

We can also examine the bias-variance trade-off quantitatively for this example. The average prediction is estimated from

$$\bar{y}(x) = \frac{1}{L} \sum_{l=1}^L y^{(l)}(x) \quad (3.45)$$

and the integrated squared bias and integrated variance are then given by

$$(\text{bias})^2 = \frac{1}{N} \sum_{n=1}^N \{\bar{y}(x_n) - h(x_n)\}^2 \quad (3.46)$$

$$\text{variance} = \frac{1}{N} \sum_{n=1}^N \frac{1}{L} \sum_{l=1}^L \{y^{(l)}(x_n) - \bar{y}(x_n)\}^2 \quad (3.47)$$

where the integral over x weighted by the distribution $p(x)$ is approximated by a finite sum over data points drawn from that distribution. These quantities, along with their sum, are plotted as a function of $\ln \lambda$ in Figure 3.6. We see that small values of λ allow the model to become finely tuned to the noise on each individual

data set leading to large variance. Conversely, a large value of λ pulls the weight parameters towards zero leading to large bias.

Although the bias-variance decomposition may provide some interesting insights into the model complexity issue from a frequentist perspective, it is of limited practical value, because the bias-variance decomposition is based on averages with respect to ensembles of data sets, whereas in practice we have only the single observed data set. If we had a large number of independent training sets of a given size, we would be better off combining them into a single large training set, which of course would reduce the level of over-fitting for a given model complexity.

Given these limitations, we turn in the next section to a Bayesian treatment of linear basis function models, which not only provides powerful insights into the issues of over-fitting but which also leads to practical techniques for addressing the question model complexity.

3.3. Bayesian Linear Regression

In our discussion of maximum likelihood for setting the parameters of a linear regression model, we have seen that the effective model complexity, governed by the number of basis functions, needs to be controlled according to the size of the data set. Adding a regularization term to the log likelihood function means the effective model complexity can then be controlled by the value of the regularization coefficient, although the choice of the number and form of the basis functions is of course still important in determining the overall behaviour of the model.

This leaves the issue of deciding the appropriate model complexity for the particular problem, which cannot be decided simply by maximizing the likelihood function, because this always leads to excessively complex models and over-fitting. Independent hold-out data can be used to determine model complexity, as discussed in Section 1.3, but this can be both computationally expensive and wasteful of valuable data. We therefore turn to a Bayesian treatment of linear regression, which will avoid the over-fitting problem of maximum likelihood, and which will also lead to automatic methods of determining model complexity using the training data alone. Again, for simplicity we will focus on the case of a single target variable t . Extension to multiple target variables is straightforward and follows the discussion of Section 3.1.5.

3.3.1 Parameter distribution

We begin our discussion of the Bayesian treatment of linear regression by introducing a prior probability distribution over the model parameters w . For the moment, we shall treat the noise precision parameter β as a known constant. First note that the likelihood function $p(t|w)$ defined by (3.10) is the exponential of a quadratic function of w . The corresponding conjugate prior is therefore given by a Gaussian distribution of the form

$$p(w) = \mathcal{N}(w|m_0, S_0) \quad (3.48)$$

having mean m_0 and covariance S_0 .

Next we compute the posterior distribution, which is proportional to the product of the likelihood function and the prior. Due to the choice of a conjugate Gaussian prior distribution, the posterior will also be Gaussian. We can evaluate this distribution by the usual procedure of completing the square in the exponential, and then finding the normalization coefficient using the standard result for a normalized Gaussian. However, we have already done the necessary work in deriving the general result (2.116), which allows us to write down the posterior distribution directly in the form

$$p(\mathbf{w}|\mathbf{t}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_N, \mathbf{S}_N) \quad (3.49)$$

where

$$\mathbf{m}_N = \mathbf{S}_N (\mathbf{S}_0^{-1} \mathbf{m}_0 + \beta \Phi^T \mathbf{t}) \quad (3.50)$$

$$\mathbf{S}_N^{-1} = \mathbf{S}_0^{-1} + \beta \Phi^T \Phi. \quad (3.51)$$

Note that because the posterior distribution is Gaussian, its mode coincides with its mean. Thus the maximum posterior weight vector is simply given by $\mathbf{w}_{\text{MAP}} = \mathbf{m}_N$. If we consider an infinitely broad prior $\mathbf{S}_0 = \alpha^{-1} \mathbf{I}$ with $\alpha \rightarrow 0$, the mean \mathbf{m}_N of the posterior distribution reduces to the maximum likelihood value \mathbf{w}_{ML} given by (3.15). Similarly, if $N = 0$, then the posterior distribution reverts to the prior. Furthermore, if data points arrive sequentially, then the posterior distribution at any stage acts as the prior distribution for the subsequent data point, such that the new posterior distribution is again given by (3.49).

For the remainder of this chapter, we shall consider a particular form of Gaussian prior in order to simplify the treatment. Specifically, we consider a zero-mean isotropic Gaussian governed by a single precision parameter α so that

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1} \mathbf{I}) \quad (3.52)$$

and the corresponding posterior distribution over \mathbf{w} is then given by (3.49) with

$$\mathbf{m}_N = \beta \mathbf{S}_N \Phi^T \mathbf{t} \quad (3.53)$$

$$\mathbf{S}_N^{-1} = \alpha \mathbf{I} + \beta \Phi^T \Phi. \quad (3.54)$$

The log of the posterior distribution is given by the sum of the log likelihood and the log of the prior and, as a function of \mathbf{w} , takes the form

$$\ln p(\mathbf{w}|\mathbf{t}) = -\frac{\beta}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 - \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} + \text{const.} \quad (3.55)$$

Maximization of this posterior distribution with respect to \mathbf{w} is therefore equivalent to the minimization of the sum-of-squares error function with the addition of a quadratic regularization term, corresponding to (3.27) with $\lambda = \alpha/\beta$.

We can illustrate Bayesian learning in a linear basis function model, as well as the sequential update of a posterior distribution, using a simple example involving straight-line fitting. Consider a single input variable x , a single target variable t and

Exercise 3.7

Exercise 3.8

a linear model of the form $y(x, \mathbf{w}) = w_0 + w_1x$. Because this has just two adaptive parameters, we can plot the prior and posterior distributions directly in parameter space. We generate synthetic data from the function $f(x, \mathbf{a}) = a_0 + a_1x$ with parameter values $a_0 = -0.3$ and $a_1 = 0.5$ by first choosing values of x_n from the uniform distribution $U(x| -1, 1)$, then evaluating $f(x_n, \mathbf{a})$, and finally adding Gaussian noise with standard deviation of 0.2 to obtain the target values t_n . Our goal is to recover the values of a_0 and a_1 from such data, and we will explore the dependence on the size of the data set. We assume here that the noise variance is known and hence we set the precision parameter to its true value $\beta = (1/0.2)^2 = 25$. Similarly, we fix the parameter α to 2.0. We shall shortly discuss strategies for determining α and β from the training data. Figure 3.7 shows the results of Bayesian learning in this model as the size of the data set is increased and demonstrates the sequential nature of Bayesian learning in which the current posterior distribution forms the prior when a new data point is observed. It is worth taking time to study this figure in detail as it illustrates several important aspects of Bayesian inference. The first row of this figure corresponds to the situation before any data points are observed and shows a plot of the prior distribution in \mathbf{w} space together with six samples of the function $y(x, \mathbf{w})$ in which the values of \mathbf{w} are drawn from the prior. In the second row, we see the situation after observing a single data point. The location (x, t) of the data point is shown by a blue circle in the right-hand column. In the left-hand column is a plot of the likelihood function $p(t|x, \mathbf{w})$ for this data point as a function of \mathbf{w} . Note that the likelihood function provides a soft constraint that the line must pass close to the data point, where close is determined by the noise precision β . For comparison, the true parameter values $a_0 = -0.3$ and $a_1 = 0.5$ used to generate the data set are shown by a white cross in the plots in the left column of Figure 3.7. When we multiply this likelihood function by the prior from the top row, and normalize, we obtain the posterior distribution shown in the middle plot on the second row. Samples of the regression function $y(x, \mathbf{w})$ obtained by drawing samples of \mathbf{w} from this posterior distribution are shown in the right-hand plot. Note that these sample lines all pass close to the data point. The third row of this figure shows the effect of observing a second data point, again shown by a blue circle in the plot in the right-hand column. The corresponding likelihood function for this second data point alone is shown in the left plot. When we multiply this likelihood function by the posterior distribution from the second row, we obtain the posterior distribution shown in the middle plot of the third row. Note that this is exactly the same posterior distribution as would be obtained by combining the original prior with the likelihood function for the two data points. This posterior has now been influenced by two data points, and because two points are sufficient to define a line this already gives a relatively compact posterior distribution. Samples from this posterior distribution give rise to the functions shown in red in the third column, and we see that these functions pass close to both of the data points. The fourth row shows the effect of observing a total of 20 data points. The left-hand plot shows the likelihood function for the 20th data point alone, and the middle plot shows the resulting posterior distribution that has now absorbed information from all 20 observations. Note how the posterior is much sharper than in the third row. In the limit of an infinite number of data points, the

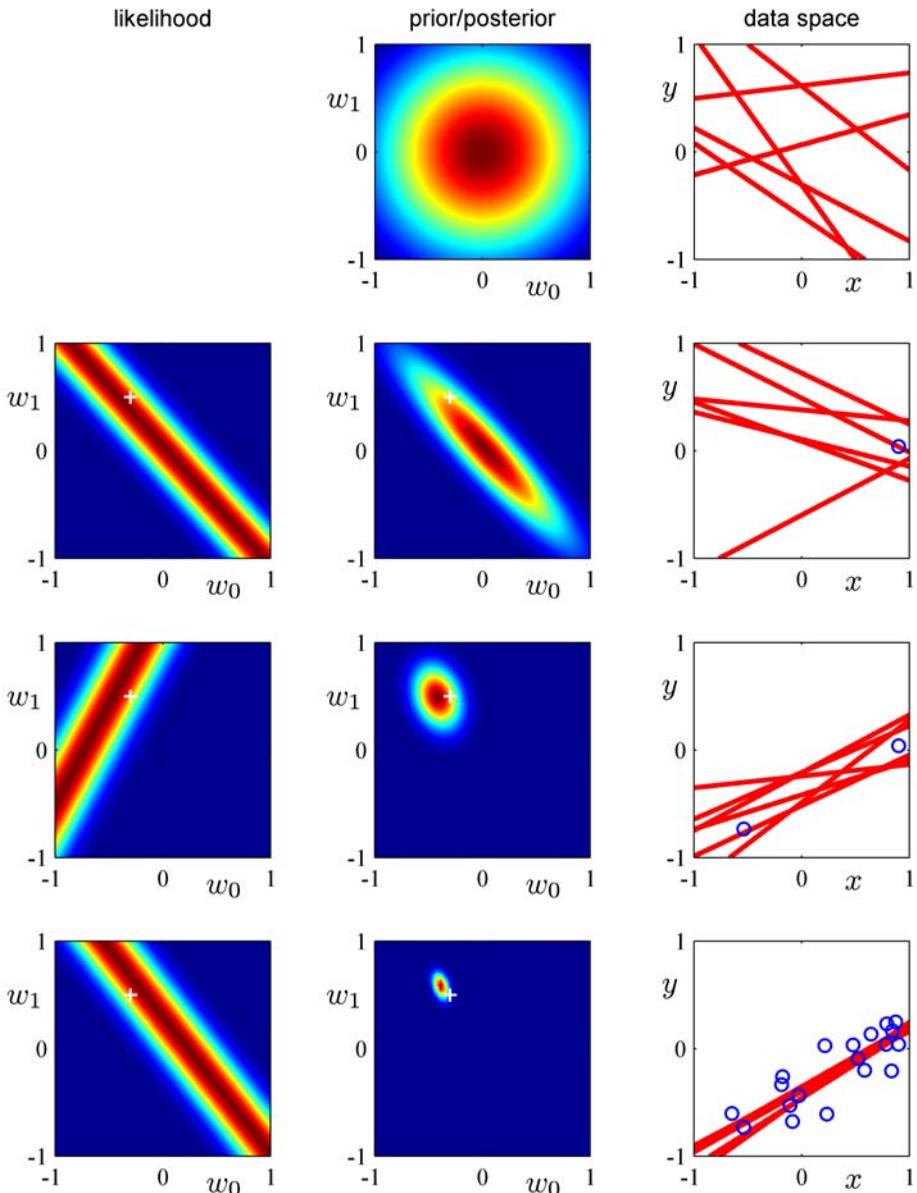


Figure 3.7 Illustration of sequential Bayesian learning for a simple linear model of the form $y(x, \mathbf{w}) = w_0 + w_1 x$. A detailed description of this figure is given in the text.

posterior distribution would become a delta function centred on the true parameter values, shown by the white cross.

Other forms of prior over the parameters can be considered. For instance, we can generalize the Gaussian prior to give

$$p(\mathbf{w}|\alpha) = \left[\frac{q}{2} \left(\frac{\alpha}{2} \right)^{1/q} \frac{1}{\Gamma(1/q)} \right]^M \exp \left(-\frac{\alpha}{2} \sum_{j=1}^M |w_j|^q \right) \quad (3.56)$$

in which $q = 2$ corresponds to the Gaussian distribution, and only in this case is the prior conjugate to the likelihood function (3.10). Finding the maximum of the posterior distribution over \mathbf{w} corresponds to minimization of the regularized error function (3.29). In the case of the Gaussian prior, the mode of the posterior distribution was equal to the mean, although this will no longer hold if $q \neq 2$.

3.3.2 Predictive distribution

In practice, we are not usually interested in the value of \mathbf{w} itself but rather in making predictions of t for new values of \mathbf{x} . This requires that we evaluate the *predictive distribution* defined by

$$p(t|\mathbf{t}, \alpha, \beta) = \int p(t|\mathbf{w}, \beta)p(\mathbf{w}|\mathbf{t}, \alpha, \beta) d\mathbf{w} \quad (3.57)$$

in which \mathbf{t} is the vector of target values from the training set, and we have omitted the corresponding input vectors from the right-hand side of the conditioning statements to simplify the notation. The conditional distribution $p(t|\mathbf{x}, \mathbf{w}, \beta)$ of the target variable is given by (3.8), and the posterior weight distribution is given by (3.49). We see that (3.57) involves the convolution of two Gaussian distributions, and so making use of the result (2.115) from Section 8.1.4, we see that the predictive distribution takes the form

$$p(t|\mathbf{x}, \mathbf{t}, \alpha, \beta) = \mathcal{N}(t|\mathbf{m}_N^T \phi(\mathbf{x}), \sigma_N^2(\mathbf{x})) \quad (3.58)$$

where the variance $\sigma_N^2(\mathbf{x})$ of the predictive distribution is given by

$$\sigma_N^2(\mathbf{x}) = \frac{1}{\beta} + \phi(\mathbf{x})^T \mathbf{S}_N \phi(\mathbf{x}). \quad (3.59)$$

The first term in (3.59) represents the noise on the data whereas the second term reflects the uncertainty associated with the parameters \mathbf{w} . Because the noise process and the distribution of \mathbf{w} are independent Gaussians, their variances are additive. Note that, as additional data points are observed, the posterior distribution becomes narrower. As a consequence it can be shown (Qazaz *et al.*, 1997) that $\sigma_{N+1}^2(\mathbf{x}) \leq \sigma_N^2(\mathbf{x})$. In the limit $N \rightarrow \infty$, the second term in (3.59) goes to zero, and the variance of the predictive distribution arises solely from the additive noise governed by the parameter β .

As an illustration of the predictive distribution for Bayesian linear regression models, let us return to the synthetic sinusoidal data set of Section 1.1. In Figure 3.8,

Exercise 3.10

Exercise 3.11

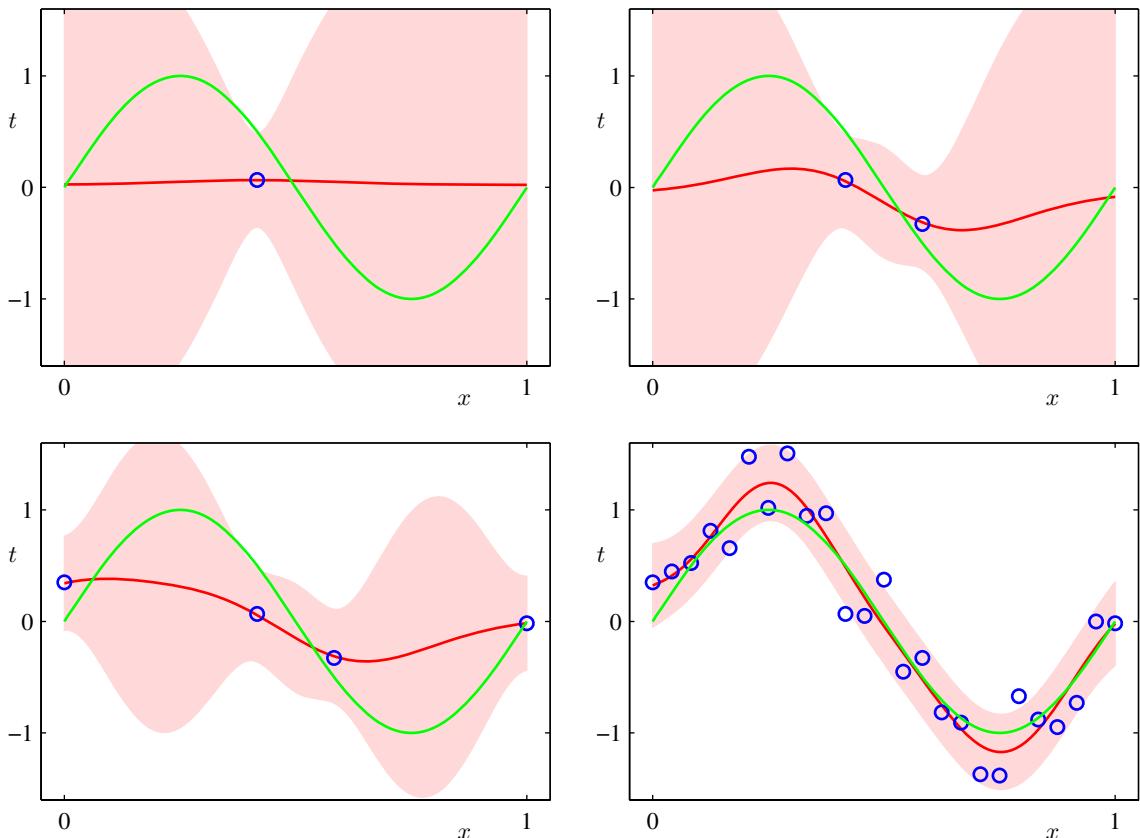


Figure 3.8 Examples of the predictive distribution (3.58) for a model consisting of 9 Gaussian basis functions of the form (3.4) using the synthetic sinusoidal data set of Section 1.1. See the text for a detailed discussion.

we fit a model comprising a linear combination of Gaussian basis functions to data sets of various sizes and then look at the corresponding posterior distributions. Here the green curves correspond to the function $\sin(2\pi x)$ from which the data points were generated (with the addition of Gaussian noise). Data sets of size $N = 1$, $N = 2$, $N = 4$, and $N = 25$ are shown in the four plots by the blue circles. For each plot, the red curve shows the mean of the corresponding Gaussian predictive distribution, and the red shaded region spans one standard deviation either side of the mean. Note that the predictive uncertainty depends on x and is smallest in the neighbourhood of the data points. Also note that the level of uncertainty decreases as more data points are observed.

The plots in Figure 3.8 only show the point-wise predictive variance as a function of x . In order to gain insight into the covariance between the predictions at different values of x , we can draw samples from the posterior distribution over w , and then plot the corresponding functions $y(x, w)$, as shown in Figure 3.9.

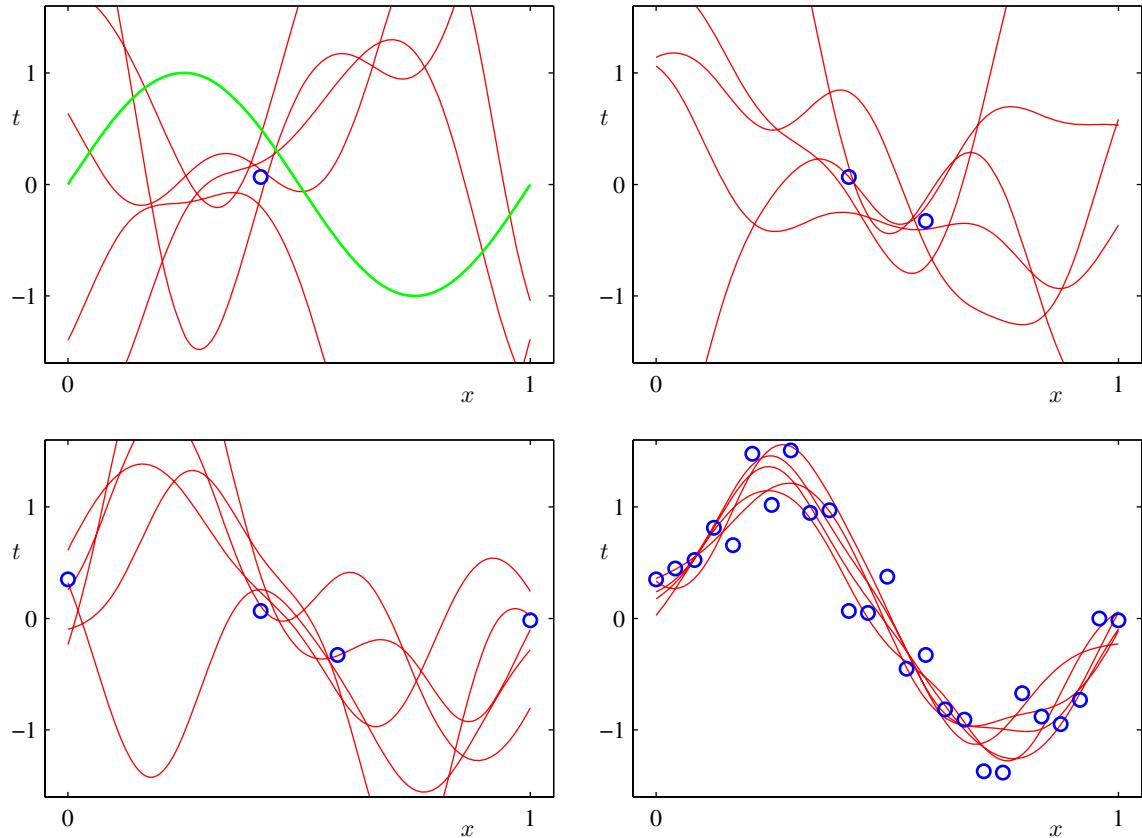


Figure 3.9 Plots of the function $y(x, w)$ using samples from the posterior distributions over w corresponding to the plots in Figure 3.8.

If we used localized basis functions such as Gaussians, then in regions away from the basis function centres, the contribution from the second term in the predictive variance (3.59) will go to zero, leaving only the noise contribution β^{-1} . Thus, the model becomes very confident in its predictions when extrapolating outside the region occupied by the basis functions, which is generally an undesirable behaviour. This problem can be avoided by adopting an alternative Bayesian approach to regression known as a Gaussian process.

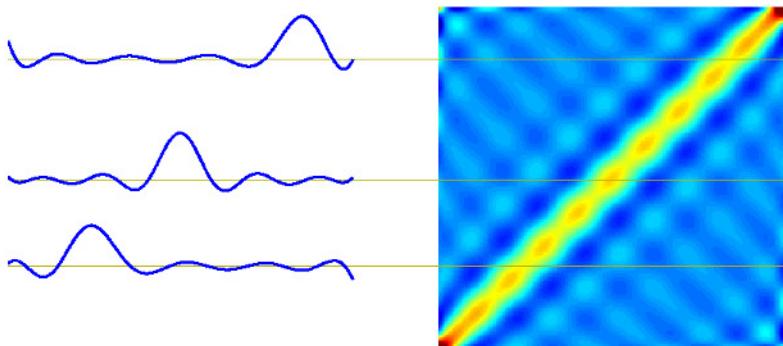
Note that, if both w and β are treated as unknown, then we can introduce a conjugate prior distribution $p(w, \beta)$ that, from the discussion in Section 2.3.6, will be given by a Gaussian-gamma distribution (Denison *et al.*, 2002). In this case, the predictive distribution is a Student's t-distribution.

Section 6.4

Exercise 3.12

Exercise 3.13

Figure 3.10 The equivalent kernel $k(x, x')$ for the Gaussian basis functions in Figure 3.1, shown as a plot of x versus x' , together with three slices through this matrix corresponding to three different values of x . The data set used to generate this kernel comprised 200 values of x equally spaced over the interval $(-1, 1)$.



3.3.3 Equivalent kernel

The posterior mean solution (3.53) for the linear basis function model has an interesting interpretation that will set the stage for kernel methods, including Gaussian processes. If we substitute (3.53) into the expression (3.3), we see that the predictive mean can be written in the form

$$y(\mathbf{x}, \mathbf{m}_N) = \mathbf{m}_N^T \boldsymbol{\phi}(\mathbf{x}) = \beta \boldsymbol{\phi}(\mathbf{x})^T \mathbf{S}_N \boldsymbol{\Phi}^T \mathbf{t} = \sum_{n=1}^N \beta \boldsymbol{\phi}(\mathbf{x})^T \mathbf{S}_N \boldsymbol{\phi}(\mathbf{x}_n) t_n \quad (3.60)$$

where \mathbf{S}_N is defined by (3.51). Thus the mean of the predictive distribution at a point \mathbf{x} is given by a linear combination of the training set target variables t_n , so that we can write

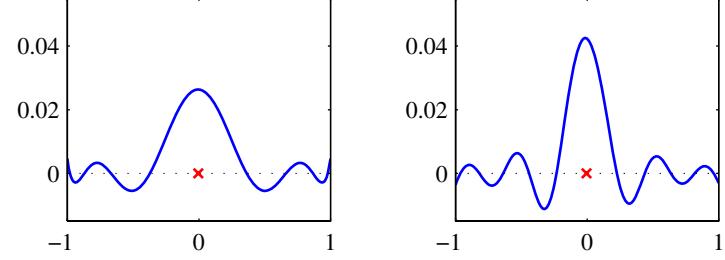
$$y(\mathbf{x}, \mathbf{m}_N) = \sum_{n=1}^N k(\mathbf{x}, \mathbf{x}_n) t_n \quad (3.61)$$

where the function

$$k(\mathbf{x}, \mathbf{x}') = \beta \boldsymbol{\phi}(\mathbf{x})^T \mathbf{S}_N \boldsymbol{\phi}(\mathbf{x}') \quad (3.62)$$

is known as the *smoother matrix* or the *equivalent kernel*. Regression functions, such as this, which make predictions by taking linear combinations of the training set target values are known as *linear smoothers*. Note that the equivalent kernel depends on the input values \mathbf{x}_n from the data set because these appear in the definition of \mathbf{S}_N . The equivalent kernel is illustrated for the case of Gaussian basis functions in Figure 3.10 in which the kernel functions $k(x, x')$ have been plotted as a function of x' for three different values of x . We see that they are localized around x , and so the mean of the predictive distribution at x , given by $y(x, \mathbf{m}_N)$, is obtained by forming a weighted combination of the target values in which data points close to x are given higher weight than points further removed from x . Intuitively, it seems reasonable that we should weight local evidence more strongly than distant evidence. Note that this localization property holds not only for the localized Gaussian basis functions but also for the nonlocal polynomial and sigmoidal basis functions, as illustrated in Figure 3.11.

Figure 3.11 Examples of equivalent kernels $k(x, x')$ for $x = 0$ plotted as a function of x' , corresponding (left) to the polynomial basis functions and (right) to the sigmoidal basis functions shown in Figure 3.1. Note that these are localized functions of x' even though the corresponding basis functions are nonlocal.



Further insight into the role of the equivalent kernel can be obtained by considering the covariance between $y(\mathbf{x})$ and $y(\mathbf{x}')$, which is given by

$$\begin{aligned}\text{cov}[y(\mathbf{x}), y(\mathbf{x}')] &= \text{cov}[\phi(\mathbf{x})^T \mathbf{w}, \mathbf{w}^T \phi(\mathbf{x}')] \\ &= \phi(\mathbf{x})^T \mathbf{S}_N \phi(\mathbf{x}') = \beta^{-1} k(\mathbf{x}, \mathbf{x}')\end{aligned}\quad (3.63)$$

where we have made use of (3.49) and (3.62). From the form of the equivalent kernel, we see that the predictive mean at nearby points will be highly correlated, whereas for more distant pairs of points the correlation will be smaller.

The predictive distribution shown in Figure 3.8 allows us to visualize the pointwise uncertainty in the predictions, governed by (3.59). However, by drawing samples from the posterior distribution over \mathbf{w} , and plotting the corresponding model functions $y(\mathbf{x}, \mathbf{w})$ as in Figure 3.9, we are visualizing the joint uncertainty in the posterior distribution between the y values at two (or more) x values, as governed by the equivalent kernel.

The formulation of linear regression in terms of a kernel function suggests an alternative approach to regression as follows. Instead of introducing a set of basis functions, which implicitly determines an equivalent kernel, we can instead define a localized kernel directly and use this to make predictions for new input vectors \mathbf{x} , given the observed training set. This leads to a practical framework for regression (and classification) called *Gaussian processes*, which will be discussed in detail in Section 6.4.

We have seen that the effective kernel defines the weights by which the training set target values are combined in order to make a prediction at a new value of \mathbf{x} , and it can be shown that these weights sum to one, in other words

$$\sum_{n=1}^N k(\mathbf{x}, \mathbf{x}_n) = 1 \quad (3.64)$$

Exercise 3.14

for all values of \mathbf{x} . This intuitively pleasing result can easily be proven informally by noting that the summation is equivalent to considering the predictive mean $\hat{y}(\mathbf{x})$ for a set of target data in which $t_n = 1$ for all n . Provided the basis functions are linearly independent, that there are more data points than basis functions, and that one of the basis functions is constant (corresponding to the bias parameter), then it is clear that we can fit the training data exactly and hence that the predictive mean will

be simply $\hat{y}(\mathbf{x}) = 1$, from which we obtain (3.64). Note that the kernel function can be negative as well as positive, so although it satisfies a summation constraint, the corresponding predictions are not necessarily convex combinations of the training set target variables.

Finally, we note that the equivalent kernel (3.62) satisfies an important property shared by kernel functions in general, namely that it can be expressed in the form an inner product with respect to a vector $\psi(\mathbf{x})$ of nonlinear functions, so that

$$k(\mathbf{x}, \mathbf{z}) = \psi(\mathbf{x})^T \psi(\mathbf{z}) \quad (3.65)$$

where $\psi(\mathbf{x}) = \beta^{1/2} \mathbf{S}_N^{1/2} \phi(\mathbf{x})$.

3.4. Bayesian Model Comparison

In Chapter 1, we highlighted the problem of over-fitting as well as the use of cross-validation as a technique for setting the values of regularization parameters or for choosing between alternative models. Here we consider the problem of model selection from a Bayesian perspective. In this section, our discussion will be very general, and then in Section 3.5 we shall see how these ideas can be applied to the determination of regularization parameters in linear regression.

As we shall see, the over-fitting associated with maximum likelihood can be avoided by marginalizing (summing or integrating) over the model parameters instead of making point estimates of their values. Models can then be compared directly on the training data, without the need for a validation set. This allows all available data to be used for training and avoids the multiple training runs for each model associated with cross-validation. It also allows multiple complexity parameters to be determined simultaneously as part of the training process. For example, in Chapter 7 we shall introduce the *relevance vector machine*, which is a Bayesian model having one complexity parameter for every training data point.

The Bayesian view of model comparison simply involves the use of probabilities to represent uncertainty in the choice of model, along with a consistent application of the sum and product rules of probability. Suppose we wish to compare a set of L models $\{\mathcal{M}_i\}$ where $i = 1, \dots, L$. Here a model refers to a probability distribution over the observed data \mathcal{D} . In the case of the polynomial curve-fitting problem, the distribution is defined over the set of target values \mathbf{t} , while the set of input values \mathbf{X} is assumed to be known. Other types of model define a joint distributions over \mathbf{X} and \mathbf{t} . We shall suppose that the data is generated from one of these models but we are uncertain which one. Our uncertainty is expressed through a prior probability distribution $p(\mathcal{M}_i)$. Given a training set \mathcal{D} , we then wish to evaluate the posterior distribution

$$p(\mathcal{M}_i | \mathcal{D}) \propto p(\mathcal{M}_i) p(\mathcal{D} | \mathcal{M}_i). \quad (3.66)$$

The prior allows us to express a preference for different models. Let us simply assume that all models are given equal prior probability. The interesting term is the *model evidence* $p(\mathcal{D} | \mathcal{M}_i)$ which expresses the preference shown by the data for

different models, and we shall examine this term in more detail shortly. The model evidence is sometimes also called the *marginal likelihood* because it can be viewed as a likelihood function over the space of models, in which the parameters have been marginalized out. The ratio of model evidences $p(\mathcal{D}|\mathcal{M}_i)/p(\mathcal{D}|\mathcal{M}_j)$ for two models is known as a *Bayes factor* (Kass and Raftery, 1995).

Once we know the posterior distribution over models, the predictive distribution is given, from the sum and product rules, by

$$p(t|\mathbf{x}, \mathcal{D}) = \sum_{i=1}^L p(t|\mathbf{x}, \mathcal{M}_i, \mathcal{D})p(\mathcal{M}_i|\mathcal{D}). \quad (3.67)$$

This is an example of a *mixture distribution* in which the overall predictive distribution is obtained by averaging the predictive distributions $p(t|\mathbf{x}, \mathcal{M}_i, \mathcal{D})$ of individual models, weighted by the posterior probabilities $p(\mathcal{M}_i|\mathcal{D})$ of those models. For instance, if we have two models that are a-posteriori equally likely and one predicts a narrow distribution around $t = a$ while the other predicts a narrow distribution around $t = b$, the overall predictive distribution will be a bimodal distribution with modes at $t = a$ and $t = b$, not a single model at $t = (a + b)/2$.

A simple approximation to model averaging is to use the single most probable model alone to make predictions. This is known as *model selection*.

For a model governed by a set of parameters \mathbf{w} , the model evidence is given, from the sum and product rules of probability, by

$$p(\mathcal{D}|\mathcal{M}_i) = \int p(\mathcal{D}|\mathbf{w}, \mathcal{M}_i)p(\mathbf{w}|\mathcal{M}_i) d\mathbf{w}. \quad (3.68)$$

Chapter 11

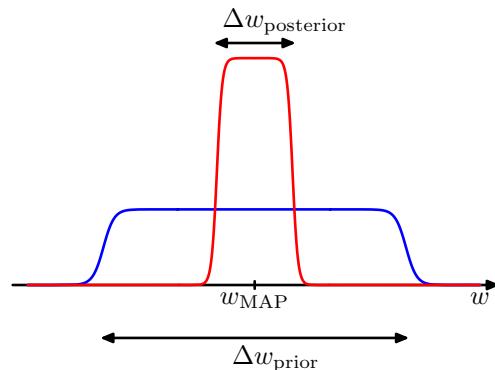
From a sampling perspective, the marginal likelihood can be viewed as the probability of generating the data set \mathcal{D} from a model whose parameters are sampled at random from the prior. It is also interesting to note that the evidence is precisely the normalizing term that appears in the denominator in Bayes' theorem when evaluating the posterior distribution over parameters because

$$p(\mathbf{w}|\mathcal{D}, \mathcal{M}_i) = \frac{p(\mathcal{D}|\mathbf{w}, \mathcal{M}_i)p(\mathbf{w}|\mathcal{M}_i)}{p(\mathcal{D}|\mathcal{M}_i)}. \quad (3.69)$$

We can obtain some insight into the model evidence by making a simple approximation to the integral over parameters. Consider first the case of a model having a single parameter w . The posterior distribution over parameters is proportional to $p(\mathcal{D}|w)p(w)$, where we omit the dependence on the model \mathcal{M}_i to keep the notation uncluttered. If we assume that the posterior distribution is sharply peaked around the most probable value w_{MAP} , with width $\Delta w_{posterior}$, then we can approximate the integral by the value of the integrand at its maximum times the width of the peak. If we further assume that the prior is flat with width Δw_{prior} so that $p(w) = 1/\Delta w_{prior}$, then we have

$$p(\mathcal{D}) = \int p(\mathcal{D}|w)p(w) dw \simeq p(\mathcal{D}|w_{MAP}) \frac{\Delta w_{posterior}}{\Delta w_{prior}} \quad (3.70)$$

Figure 3.12 We can obtain a rough approximation to the model evidence if we assume that the posterior distribution over parameters is sharply peaked around its mode w_{MAP} .



and so taking logs we obtain

$$\ln p(\mathcal{D}) \simeq \ln p(\mathcal{D}|w_{\text{MAP}}) + \ln \left(\frac{\Delta w_{\text{posterior}}}{\Delta w_{\text{prior}}} \right). \quad (3.71)$$

This approximation is illustrated in Figure 3.12. The first term represents the fit to the data given by the most probable parameter values, and for a flat prior this would correspond to the log likelihood. The second term penalizes the model according to its complexity. Because $\Delta w_{\text{posterior}} < \Delta w_{\text{prior}}$ this term is negative, and it increases in magnitude as the ratio $\Delta w_{\text{posterior}}/\Delta w_{\text{prior}}$ gets smaller. Thus, if parameters are finely tuned to the data in the posterior distribution, then the penalty term is large.

For a model having a set of M parameters, we can make a similar approximation for each parameter in turn. Assuming that all parameters have the same ratio of $\Delta w_{\text{posterior}}/\Delta w_{\text{prior}}$, we obtain

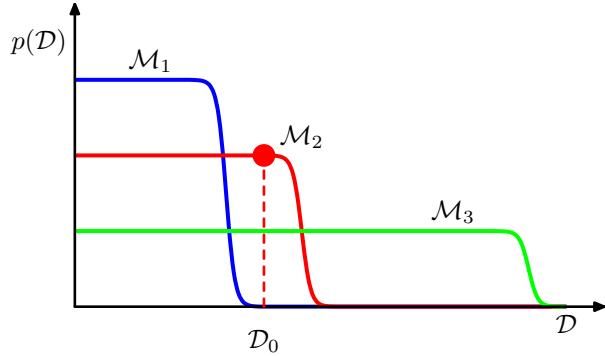
$$\ln p(\mathcal{D}) \simeq \ln p(\mathcal{D}|w_{\text{MAP}}) + M \ln \left(\frac{\Delta w_{\text{posterior}}}{\Delta w_{\text{prior}}} \right). \quad (3.72)$$

Thus, in this very simple approximation, the size of the complexity penalty increases linearly with the number M of adaptive parameters in the model. As we increase the complexity of the model, the first term will typically decrease, because a more complex model is better able to fit the data, whereas the second term will increase due to the dependence on M . The optimal model complexity, as determined by the maximum evidence, will be given by a trade-off between these two competing terms. We shall later develop a more refined version of this approximation, based on a Gaussian approximation to the posterior distribution.

We can gain further insight into Bayesian model comparison and understand how the marginal likelihood can favour models of intermediate complexity by considering Figure 3.13. Here the horizontal axis is a one-dimensional representation of the space of possible data sets, so that each point on this axis corresponds to a specific data set. We now consider three models \mathcal{M}_1 , \mathcal{M}_2 and \mathcal{M}_3 of successively increasing complexity. Imagine running these models generatively to produce example data sets, and then looking at the distribution of data sets that result. Any given

Section 4.4.1

Figure 3.13 Schematic illustration of the distribution of data sets for three models of different complexity, in which \mathcal{M}_1 is the simplest and \mathcal{M}_3 is the most complex. Note that the distributions are normalized. In this example, for the particular observed data set \mathcal{D}_0 , the model \mathcal{M}_2 with intermediate complexity has the largest evidence.



model can generate a variety of different data sets since the parameters are governed by a prior probability distribution, and for any choice of the parameters there may be random noise on the target variables. To generate a particular data set from a specific model, we first choose the values of the parameters from their prior distribution $p(\mathbf{w})$, and then for these parameter values we sample the data from $p(\mathcal{D}|\mathbf{w})$. A simple model (for example, based on a first order polynomial) has little variability and so will generate data sets that are fairly similar to each other. Its distribution $p(\mathcal{D})$ is therefore confined to a relatively small region of the horizontal axis. By contrast, a complex model (such as a ninth order polynomial) can generate a great variety of different data sets, and so its distribution $p(\mathcal{D})$ is spread over a large region of the space of data sets. Because the distributions $p(\mathcal{D}|\mathcal{M}_i)$ are normalized, we see that the particular data set \mathcal{D}_0 can have the highest value of the evidence for the model of intermediate complexity. Essentially, the simpler model cannot fit the data well, whereas the more complex model spreads its predictive probability over too broad a range of data sets and so assigns relatively small probability to any one of them.

Implicit in the Bayesian model comparison framework is the assumption that the true distribution from which the data are generated is contained within the set of models under consideration. Provided this is so, we can show that Bayesian model comparison will on average favour the correct model. To see this, consider two models \mathcal{M}_1 and \mathcal{M}_2 in which the truth corresponds to \mathcal{M}_1 . For a given finite data set, it is possible for the Bayes factor to be larger for the incorrect model. However, if we average the Bayes factor over the distribution of data sets, we obtain the expected Bayes factor in the form

$$\int p(\mathcal{D}|\mathcal{M}_1) \ln \frac{p(\mathcal{D}|\mathcal{M}_1)}{p(\mathcal{D}|\mathcal{M}_2)} d\mathcal{D} \quad (3.73)$$

where the average has been taken with respect to the true distribution of the data. This quantity is an example of the *Kullback-Leibler* divergence and satisfies the property of always being positive unless the two distributions are equal in which case it is zero. Thus on average the Bayes factor will always favour the correct model.

We have seen that the Bayesian framework avoids the problem of over-fitting and allows models to be compared on the basis of the training data alone. However,

a Bayesian approach, like any approach to pattern recognition, needs to make assumptions about the form of the model, and if these are invalid then the results can be misleading. In particular, we see from Figure 3.12 that the model evidence can be sensitive to many aspects of the prior, such as the behaviour in the tails. Indeed, the evidence is not defined if the prior is improper, as can be seen by noting that an improper prior has an arbitrary scaling factor (in other words, the normalization coefficient is not defined because the distribution cannot be normalized). If we consider a proper prior and then take a suitable limit in order to obtain an improper prior (for example, a Gaussian prior in which we take the limit of infinite variance) then the evidence will go to zero, as can be seen from (3.70) and Figure 3.12. It may, however, be possible to consider the evidence ratio between two models first and then take a limit to obtain a meaningful answer.

In a practical application, therefore, it will be wise to keep aside an independent test set of data on which to evaluate the overall performance of the final system.

3.5. The Evidence Approximation

In a fully Bayesian treatment of the linear basis function model, we would introduce prior distributions over the hyperparameters α and β and make predictions by marginalizing with respect to these hyperparameters as well as with respect to the parameters \mathbf{w} . However, although we can integrate analytically over either \mathbf{w} or over the hyperparameters, the complete marginalization over all of these variables is analytically intractable. Here we discuss an approximation in which we set the hyperparameters to specific values determined by maximizing the *marginal likelihood function* obtained by first integrating over the parameters \mathbf{w} . This framework is known in the statistics literature as *empirical Bayes* (Bernardo and Smith, 1994; Gelman *et al.*, 2004), or *type 2 maximum likelihood* (Berger, 1985), or *generalized maximum likelihood* (Wahba, 1975), and in the machine learning literature is also called the *evidence approximation* (Gull, 1989; MacKay, 1992a).

If we introduce hyperpriors over α and β , the predictive distribution is obtained by marginalizing over \mathbf{w} , α and β so that

$$p(t|\mathbf{t}) = \iiint p(t|\mathbf{w}, \beta) p(\mathbf{w}|\mathbf{t}, \alpha, \beta) p(\alpha, \beta|\mathbf{t}) d\mathbf{w} d\alpha d\beta \quad (3.74)$$

where $p(t|\mathbf{w}, \beta)$ is given by (3.8) and $p(\mathbf{w}|\mathbf{t}, \alpha, \beta)$ is given by (3.49) with \mathbf{m}_N and \mathbf{S}_N defined by (3.53) and (3.54) respectively. Here we have omitted the dependence on the input variable \mathbf{x} to keep the notation uncluttered. If the posterior distribution $p(\alpha, \beta|\mathbf{t})$ is sharply peaked around values $\hat{\alpha}$ and $\hat{\beta}$, then the predictive distribution is obtained simply by marginalizing over \mathbf{w} in which α and β are fixed to the values $\hat{\alpha}$ and $\hat{\beta}$, so that

$$p(t|\mathbf{t}) \simeq p(t|\mathbf{t}, \hat{\alpha}, \hat{\beta}) = \int p(t|\mathbf{w}, \hat{\beta}) p(\mathbf{w}|\mathbf{t}, \hat{\alpha}, \hat{\beta}) d\mathbf{w}. \quad (3.75)$$

From Bayes' theorem, the posterior distribution for α and β is given by

$$p(\alpha, \beta | \mathbf{t}) \propto p(\mathbf{t} | \alpha, \beta) p(\alpha, \beta). \quad (3.76)$$

If the prior is relatively flat, then in the evidence framework the values of $\hat{\alpha}$ and $\hat{\beta}$ are obtained by maximizing the marginal likelihood function $p(\mathbf{t} | \alpha, \beta)$. We shall proceed by evaluating the marginal likelihood for the linear basis function model and then finding its maxima. This will allow us to determine values for these hyperparameters from the training data alone, without recourse to cross-validation. Recall that the ratio α/β is analogous to a regularization parameter.

As an aside it is worth noting that, if we define conjugate (Gamma) prior distributions over α and β , then the marginalization over these hyperparameters in (3.74) can be performed analytically to give a Student's t-distribution over \mathbf{w} (see Section 2.3.7). Although the resulting integral over \mathbf{w} is no longer analytically tractable, it might be thought that approximating this integral, for example using the Laplace approximation discussed (Section 4.4) which is based on a local Gaussian approximation centred on the mode of the posterior distribution, might provide a practical alternative to the evidence framework (Buntine and Weigend, 1991). However, the integrand as a function of \mathbf{w} typically has a strongly skewed mode so that the Laplace approximation fails to capture the bulk of the probability mass, leading to poorer results than those obtained by maximizing the evidence (MacKay, 1999).

Returning to the evidence framework, we note that there are two approaches that we can take to the maximization of the log evidence. We can evaluate the evidence function analytically and then set its derivative equal to zero to obtain re-estimation equations for α and β , which we shall do in Section 3.5.2. Alternatively we use a technique called the expectation maximization (EM) algorithm, which will be discussed in Section 9.3.4 where we shall also show that these two approaches converge to the same solution.

3.5.1 Evaluation of the evidence function

The marginal likelihood function $p(\mathbf{t} | \alpha, \beta)$ is obtained by integrating over the weight parameters \mathbf{w} , so that

$$p(\mathbf{t} | \alpha, \beta) = \int p(\mathbf{t} | \mathbf{w}, \beta) p(\mathbf{w} | \alpha) d\mathbf{w}. \quad (3.77)$$

One way to evaluate this integral is to make use once again of the result (2.115) for the conditional distribution in a linear-Gaussian model. Here we shall evaluate the integral instead by completing the square in the exponent and making use of the standard form for the normalization coefficient of a Gaussian.

From (3.11), (3.12), and (3.52), we can write the evidence function in the form

$$p(\mathbf{t} | \alpha, \beta) = \left(\frac{\beta}{2\pi} \right)^{N/2} \left(\frac{\alpha}{2\pi} \right)^{M/2} \int \exp \{-E(\mathbf{w})\} d\mathbf{w} \quad (3.78)$$

Exercise 3.16

Exercise 3.17

where M is the dimensionality of \mathbf{w} , and we have defined

$$\begin{aligned} E(\mathbf{w}) &= \beta E_D(\mathbf{w}) + \alpha E_W(\mathbf{w}) \\ &= \frac{\beta}{2} \|\mathbf{t} - \Phi \mathbf{w}\|^2 + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w}. \end{aligned} \quad (3.79)$$

We recognize (3.79) as being equal, up to a constant of proportionality, to the regularized sum-of-squares error function (3.27). We now complete the square over \mathbf{w} giving

$$E(\mathbf{w}) = E(\mathbf{m}_N) + \frac{1}{2} (\mathbf{w} - \mathbf{m}_N)^T \mathbf{A} (\mathbf{w} - \mathbf{m}_N) \quad (3.80)$$

where we have introduced

$$\mathbf{A} = \alpha \mathbf{I} + \beta \Phi^T \Phi \quad (3.81)$$

together with

$$E(\mathbf{m}_N) = \frac{\beta}{2} \|\mathbf{t} - \Phi \mathbf{m}_N\|^2 + \frac{\alpha}{2} \mathbf{m}_N^T \mathbf{m}_N. \quad (3.82)$$

Note that \mathbf{A} corresponds to the matrix of second derivatives of the error function

$$\mathbf{A} = \nabla \nabla E(\mathbf{w}) \quad (3.83)$$

and is known as the *Hessian matrix*. Here we have also defined \mathbf{m}_N given by

$$\mathbf{m}_N = \beta \mathbf{A}^{-1} \Phi^T \mathbf{t}. \quad (3.84)$$

Using (3.54), we see that $\mathbf{A} = \mathbf{S}_N^{-1}$, and hence (3.84) is equivalent to the previous definition (3.53), and therefore represents the mean of the posterior distribution.

The integral over \mathbf{w} can now be evaluated simply by appealing to the standard result for the normalization coefficient of a multivariate Gaussian, giving

$$\begin{aligned} &\int \exp \{-E(\mathbf{w})\} d\mathbf{w} \\ &= \exp \{-E(\mathbf{m}_N)\} \int \exp \left\{ -\frac{1}{2} (\mathbf{w} - \mathbf{m}_N)^T \mathbf{A} (\mathbf{w} - \mathbf{m}_N) \right\} d\mathbf{w} \\ &= \exp \{-E(\mathbf{m}_N)\} (2\pi)^{M/2} |\mathbf{A}|^{-1/2}. \end{aligned} \quad (3.85)$$

Using (3.78) we can then write the log of the marginal likelihood in the form

$$\ln p(\mathbf{t} | \alpha, \beta) = \frac{M}{2} \ln \alpha + \frac{N}{2} \ln \beta - E(\mathbf{m}_N) - \frac{1}{2} \ln |\mathbf{A}| - \frac{N}{2} \ln(2\pi) \quad (3.86)$$

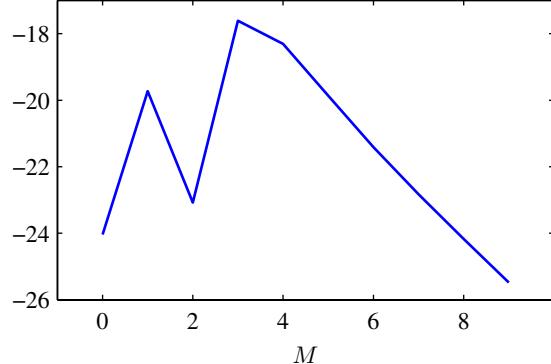
which is the required expression for the evidence function.

Returning to the polynomial regression problem, we can plot the model evidence against the order of the polynomial, as shown in Figure 3.14. Here we have assumed a prior of the form (1.65) with the parameter α fixed at $\alpha = 5 \times 10^{-3}$. The form of this plot is very instructive. Referring back to Figure 1.4, we see that the $M = 0$ polynomial has very poor fit to the data and consequently gives a relatively low value

Exercise 3.18

Exercise 3.19

Figure 3.14 Plot of the model evidence versus the order M , for the polynomial regression model, showing that the evidence favours the model with $M = 3$.



for the evidence. Going to the $M = 1$ polynomial greatly improves the data fit, and hence the evidence is significantly higher. However, in going to $M = 2$, the data fit is improved only very marginally, due to the fact that the underlying sinusoidal function from which the data is generated is an odd function and so has no even terms in a polynomial expansion. Indeed, Figure 1.5 shows that the residual data error is reduced only slightly in going from $M = 1$ to $M = 2$. Because this richer model suffers a greater complexity penalty, the evidence actually falls in going from $M = 1$ to $M = 2$. When we go to $M = 3$ we obtain a significant further improvement in data fit, as seen in Figure 1.4, and so the evidence is increased again, giving the highest overall evidence for any of the polynomials. Further increases in the value of M produce only small improvements in the fit to the data but suffer increasing complexity penalty, leading overall to a decrease in the evidence values. Looking again at Figure 1.5, we see that the generalization error is roughly constant between $M = 3$ and $M = 8$, and it would be difficult to choose between these models on the basis of this plot alone. The evidence values, however, show a clear preference for $M = 3$, since this is the simplest model which gives a good explanation for the observed data.

3.5.2 Maximizing the evidence function

Let us first consider the maximization of $p(\mathbf{t}|\alpha, \beta)$ with respect to α . This can be done by first defining the following eigenvector equation

$$(\beta \Phi^T \Phi) \mathbf{u}_i = \lambda_i \mathbf{u}_i. \quad (3.87)$$

From (3.81), it then follows that \mathbf{A} has eigenvalues $\alpha + \lambda_i$. Now consider the derivative of the term involving $\ln |\mathbf{A}|$ in (3.86) with respect to α . We have

$$\frac{d}{d\alpha} \ln |\mathbf{A}| = \frac{d}{d\alpha} \ln \prod_i (\lambda_i + \alpha) = \frac{d}{d\alpha} \sum_i \ln(\lambda_i + \alpha) = \sum_i \frac{1}{\lambda_i + \alpha}. \quad (3.88)$$

Thus the stationary points of (3.86) with respect to α satisfy

$$0 = \frac{M}{2\alpha} - \frac{1}{2} \mathbf{m}_N^T \mathbf{m}_N - \frac{1}{2} \sum_i \frac{1}{\lambda_i + \alpha}. \quad (3.89)$$

Multiplying through by 2α and rearranging, we obtain

$$\alpha \mathbf{m}_N^T \mathbf{m}_N = M - \alpha \sum_i \frac{1}{\lambda_i + \alpha} = \gamma. \quad (3.90)$$

Since there are M terms in the sum over i , the quantity γ can be written

$$\gamma = \sum_i \frac{\lambda_i}{\alpha + \lambda_i}. \quad (3.91)$$

The interpretation of the quantity γ will be discussed shortly. From (3.90) we see that the value of α that maximizes the marginal likelihood satisfies

$$\alpha = \frac{\gamma}{\mathbf{m}_N^T \mathbf{m}_N}. \quad (3.92)$$

Note that this is an implicit solution for α not only because γ depends on α , but also because the mode \mathbf{m}_N of the posterior distribution itself depends on the choice of α . We therefore adopt an iterative procedure in which we make an initial choice for α and use this to find \mathbf{m}_N , which is given by (3.53), and also to evaluate γ , which is given by (3.91). These values are then used to re-estimate α using (3.92), and the process repeated until convergence. Note that because the matrix $\Phi^T \Phi$ is fixed, we can compute its eigenvalues once at the start and then simply multiply these by β to obtain the λ_i .

It should be emphasized that the value of α has been determined purely by looking at the training data. In contrast to maximum likelihood methods, no independent data set is required in order to optimize the model complexity.

We can similarly maximize the log marginal likelihood (3.86) with respect to β . To do this, we note that the eigenvalues λ_i defined by (3.87) are proportional to β , and hence $d\lambda_i/d\beta = \lambda_i/\beta$ giving

$$\frac{d}{d\beta} \ln |\mathbf{A}| = \frac{d}{d\beta} \sum_i \ln(\lambda_i + \alpha) = \frac{1}{\beta} \sum_i \frac{\lambda_i}{\lambda_i + \alpha} = \frac{\gamma}{\beta}. \quad (3.93)$$

The stationary point of the marginal likelihood therefore satisfies

$$0 = \frac{N}{2\beta} - \frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{m}_N^T \phi(\mathbf{x}_n)\}^2 - \frac{\gamma}{2\beta} \quad (3.94)$$

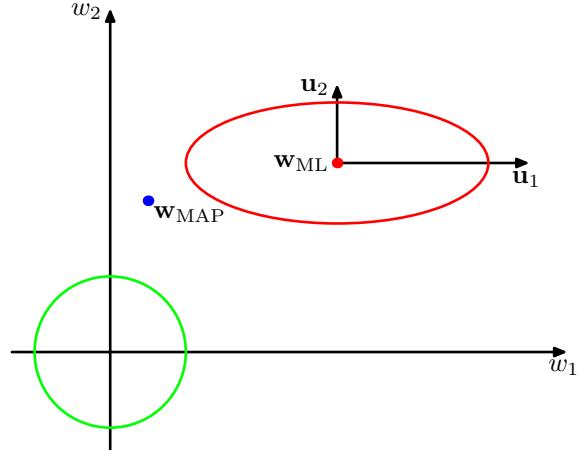
and rearranging we obtain

$$\frac{1}{\beta} = \frac{1}{N - \gamma} \sum_{n=1}^N \{t_n - \mathbf{m}_N^T \phi(\mathbf{x}_n)\}^2. \quad (3.95)$$

Again, this is an implicit solution for β and can be solved by choosing an initial value for β and then using this to calculate \mathbf{m}_N and γ and then re-estimate β using (3.95), repeating until convergence. If both α and β are to be determined from the data, then their values can be re-estimated together after each update of γ .

Exercise 3.20

Figure 3.15 Contours of the likelihood function (red) and the prior (green) in which the axes in parameter space have been rotated to align with the eigenvectors \mathbf{u}_i of the Hessian. For $\alpha = 0$, the mode of the posterior is given by the maximum likelihood solution \mathbf{w}_{ML} , whereas for nonzero α the mode is at $\mathbf{w}_{MAP} = \mathbf{m}_N$. In the direction w_1 the eigenvalue λ_1 , defined by (3.87), is small compared with α and so the quantity $\lambda_1/(\lambda_1 + \alpha)$ is close to zero, and the corresponding MAP value of w_1 is also close to zero. By contrast, in the direction w_2 the eigenvalue λ_2 is large compared with α and so the quantity $\lambda_2/(\lambda_2 + \alpha)$ is close to unity, and the MAP value of w_2 is close to its maximum likelihood value.



3.5.3 Effective number of parameters

The result (3.92) has an elegant interpretation (MacKay, 1992a), which provides insight into the Bayesian solution for α . To see this, consider the contours of the likelihood function and the prior as illustrated in Figure 3.15. Here we have implicitly transformed to a rotated set of axes in parameter space aligned with the eigenvectors \mathbf{u}_i defined in (3.87). Contours of the likelihood function are then axis-aligned ellipses. The eigenvalues λ_i measure the curvature of the likelihood function, and so in Figure 3.15 the eigenvalue λ_1 is small compared with λ_2 (because a smaller curvature corresponds to a greater elongation of the contours of the likelihood function). Because $\beta\Phi^T\Phi$ is a positive definite matrix, it will have positive eigenvalues, and so the ratio $\lambda_i/(\lambda_i + \alpha)$ will lie between 0 and 1. Consequently, the quantity γ defined by (3.91) will lie in the range $0 \leq \gamma \leq M$. For directions in which $\lambda_i \gg \alpha$, the corresponding parameter w_i will be close to its maximum likelihood value, and the ratio $\lambda_i/(\lambda_i + \alpha)$ will be close to 1. Such parameters are called *well determined* because their values are tightly constrained by the data. Conversely, for directions in which $\lambda_i \ll \alpha$, the corresponding parameters w_i will be close to zero, as will the ratios $\lambda_i/(\lambda_i + \alpha)$. These are directions in which the likelihood function is relatively insensitive to the parameter value and so the parameter has been set to a small value by the prior. The quantity γ defined by (3.91) therefore measures the effective total number of well determined parameters.

We can obtain some insight into the result (3.95) for re-estimating β by comparing it with the corresponding maximum likelihood result given by (3.21). Both of these formulae express the variance (the inverse precision) as an average of the squared differences between the targets and the model predictions. However, they differ in that the number of data points N in the denominator of the maximum likelihood result is replaced by $N - \gamma$ in the Bayesian result. We recall from (1.56) that the maximum likelihood estimate of the variance for a Gaussian distribution over a

single variable x is given by

$$\sigma_{\text{ML}}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \mu_{\text{ML}})^2 \quad (3.96)$$

and that this estimate is biased because the maximum likelihood solution μ_{ML} for the mean has fitted some of the noise on the data. In effect, this has used up one degree of freedom in the model. The corresponding unbiased estimate is given by (1.59) and takes the form

$$\sigma_{\text{MAP}}^2 = \frac{1}{N-1} \sum_{n=1}^N (x_n - \mu_{\text{ML}})^2. \quad (3.97)$$

We shall see in Section 10.1.3 that this result can be obtained from a Bayesian treatment in which we marginalize over the unknown mean. The factor of $N-1$ in the denominator of the Bayesian result takes account of the fact that one degree of freedom has been used in fitting the mean and removes the bias of maximum likelihood. Now consider the corresponding results for the linear regression model. The mean of the target distribution is now given by the function $\mathbf{w}^T \phi(\mathbf{x})$, which contains M parameters. However, not all of these parameters are tuned to the data. The effective number of parameters that are determined by the data is γ , with the remaining $M-\gamma$ parameters set to small values by the prior. This is reflected in the Bayesian result for the variance that has a factor $N-\gamma$ in the denominator, thereby correcting for the bias of the maximum likelihood result.

We can illustrate the evidence framework for setting hyperparameters using the sinusoidal synthetic data set from Section 1.1, together with the Gaussian basis function model comprising 9 basis functions, so that the total number of parameters in the model is given by $M = 10$ including the bias. Here, for simplicity of illustration, we have set β to its true value of 11.1 and then used the evidence framework to determine α , as shown in Figure 3.16.

We can also see how the parameter α controls the magnitude of the parameters $\{w_i\}$, by plotting the individual parameters versus the effective number γ of parameters, as shown in Figure 3.17.

If we consider the limit $N \gg M$ in which the number of data points is large in relation to the number of parameters, then from (3.87) all of the parameters will be well determined by the data because $\Phi^T \Phi$ involves an implicit sum over data points, and so the eigenvalues λ_i increase with the size of the data set. In this case, $\gamma = M$, and the re-estimation equations for α and β become

$$\alpha = \frac{M}{2E_W(\mathbf{m}_N)} \quad (3.98)$$

$$\beta = \frac{N}{2E_D(\mathbf{m}_N)} \quad (3.99)$$

where E_W and E_D are defined by (3.25) and (3.26), respectively. These results can be used as an easy-to-compute approximation to the full evidence re-estimation

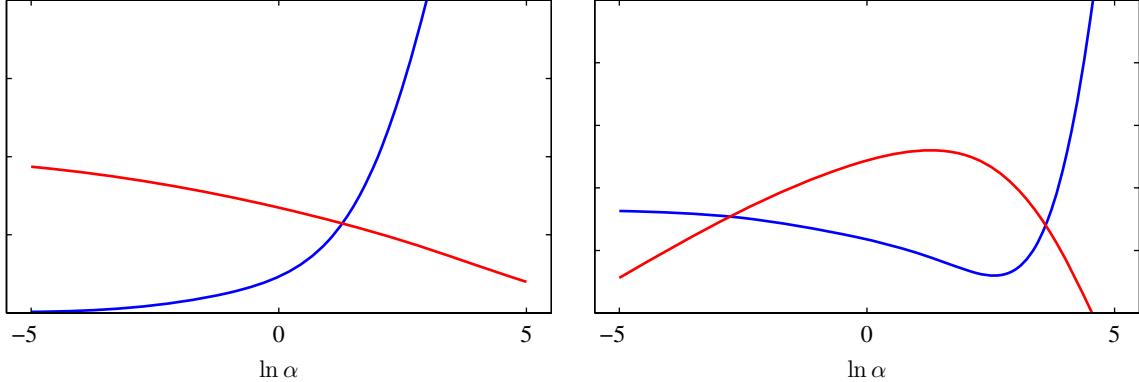
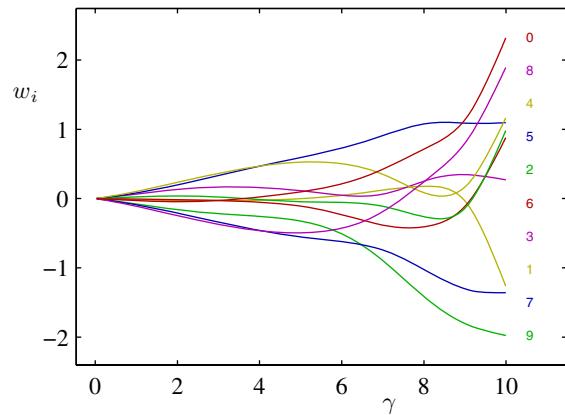


Figure 3.16 The left plot shows γ (red curve) and $2\alpha E_W(m_N)$ (blue curve) versus $\ln \alpha$ for the sinusoidal synthetic data set. It is the intersection of these two curves that defines the optimum value for α given by the evidence procedure. The right plot shows the corresponding graph of log evidence $\ln p(t|\alpha, \beta)$ versus $\ln \alpha$ (red curve) showing that the peak coincides with the crossing point of the curves in the left plot. Also shown is the test set error (blue curve) showing that the evidence maximum occurs close to the point of best generalization.

formulae, because they do not require evaluation of the eigenvalue spectrum of the Hessian.

Figure 3.17 Plot of the 10 parameters w_i from the Gaussian basis function model versus the effective number of parameters γ , in which the hyperparameter α is varied in the range $0 \leq \alpha \leq \infty$ causing γ to vary in the range $0 \leq \gamma \leq M$.



3.6. Limitations of Fixed Basis Functions

Throughout this chapter, we have focussed on models comprising a linear combination of fixed, nonlinear basis functions. We have seen that the assumption of linearity in the parameters led to a range of useful properties including closed-form solutions to the least-squares problem, as well as a tractable Bayesian treatment. Furthermore, for a suitable choice of basis functions, we can model arbitrary nonlinearities in the

mapping from input variables to targets. In the next chapter, we shall study an analogous class of models for classification.

It might appear, therefore, that such linear models constitute a general purpose framework for solving problems in pattern recognition. Unfortunately, there are some significant shortcomings with linear models, which will cause us to turn in later chapters to more complex models such as support vector machines and neural networks.

The difficulty stems from the assumption that the basis functions $\phi_j(\mathbf{x})$ are fixed before the training data set is observed and is a manifestation of the curse of dimensionality discussed in Section 1.4. As a consequence, the number of basis functions needs to grow rapidly, often exponentially, with the dimensionality D of the input space.

Fortunately, there are two properties of real data sets that we can exploit to help alleviate this problem. First of all, the data vectors $\{\mathbf{x}_n\}$ typically lie close to a non-linear manifold whose intrinsic dimensionality is smaller than that of the input space as a result of strong correlations between the input variables. We will see an example of this when we consider images of handwritten digits in Chapter 12. If we are using localized basis functions, we can arrange that they are scattered in input space only in regions containing data. This approach is used in radial basis function networks and also in support vector and relevance vector machines. Neural network models, which use adaptive basis functions having sigmoidal nonlinearities, can adapt the parameters so that the regions of input space over which the basis functions vary corresponds to the data manifold. The second property is that target variables may have significant dependence on only a small number of possible directions within the data manifold. Neural networks can exploit this property by choosing the directions in input space to which the basis functions respond.

Exercises

- 3.1** (*) **WWW** Show that the ‘tanh’ function and the logistic sigmoid function (3.6) are related by

$$\tanh(a) = 2\sigma(2a) - 1. \quad (3.100)$$

Hence show that a general linear combination of logistic sigmoid functions of the form

$$y(x, \mathbf{w}) = w_0 + \sum_{j=1}^M w_j \sigma\left(\frac{x - \mu_j}{s}\right) \quad (3.101)$$

is equivalent to a linear combination of ‘tanh’ functions of the form

$$y(x, \mathbf{u}) = u_0 + \sum_{j=1}^M u_j \tanh\left(\frac{x - \mu_j}{s}\right) \quad (3.102)$$

and find expressions to relate the new parameters $\{u_1, \dots, u_M\}$ to the original parameters $\{w_1, \dots, w_M\}$.

3. LINEAR MODELS FOR REGRESSION

- 3.2** (**) Show that the matrix

$$\Phi(\Phi^T \Phi)^{-1} \Phi^T \quad (3.103)$$

takes any vector \mathbf{v} and projects it onto the space spanned by the columns of Φ . Use this result to show that the least-squares solution (3.15) corresponds to an orthogonal projection of the vector \mathbf{t} onto the manifold \mathcal{S} as shown in Figure 3.2.

- 3.3** (*) Consider a data set in which each data point t_n is associated with a weighting factor $r_n > 0$, so that the sum-of-squares error function becomes

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N r_n \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2. \quad (3.104)$$

Find an expression for the solution \mathbf{w}^* that minimizes this error function. Give two alternative interpretations of the weighted sum-of-squares error function in terms of (i) data dependent noise variance and (ii) replicated data points.

- 3.4** (*) **www** Consider a linear model of the form

$$y(x, \mathbf{w}) = w_0 + \sum_{i=1}^D w_i x_i \quad (3.105)$$

together with a sum-of-squares error function of the form

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2. \quad (3.106)$$

Now suppose that Gaussian noise ϵ_i with zero mean and variance σ^2 is added independently to each of the input variables x_i . By making use of $\mathbb{E}[\epsilon_i] = 0$ and $\mathbb{E}[\epsilon_i \epsilon_j] = \delta_{ij} \sigma^2$, show that minimizing E_D averaged over the noise distribution is equivalent to minimizing the sum-of-squares error for noise-free input variables with the addition of a weight-decay regularization term, in which the bias parameter w_0 is omitted from the regularizer.

- 3.5** (*) **www** Using the technique of Lagrange multipliers, discussed in Appendix E, show that minimization of the regularized error function (3.29) is equivalent to minimizing the unregularized sum-of-squares error (3.12) subject to the constraint (3.30). Discuss the relationship between the parameters η and λ .

- 3.6** (*) **www** Consider a linear basis function regression model for a multivariate target variable \mathbf{t} having a Gaussian distribution of the form

$$p(\mathbf{t}|\mathbf{W}, \Sigma) = \mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x}, \mathbf{W}), \Sigma) \quad (3.107)$$

where

$$\mathbf{y}(\mathbf{x}, \mathbf{W}) = \mathbf{W}^T \phi(\mathbf{x}) \quad (3.108)$$

together with a training data set comprising input basis vectors $\phi(\mathbf{x}_n)$ and corresponding target vectors \mathbf{t}_n , with $n = 1, \dots, N$. Show that the maximum likelihood solution \mathbf{W}_{ML} for the parameter matrix \mathbf{W} has the property that each column is given by an expression of the form (3.15), which was the solution for an isotropic noise distribution. Note that this is independent of the covariance matrix Σ . Show that the maximum likelihood solution for Σ is given by

$$\Sigma = \frac{1}{N} \sum_{n=1}^N (\mathbf{t}_n - \mathbf{W}_{\text{ML}}^T \phi(\mathbf{x}_n)) (\mathbf{t}_n - \mathbf{W}_{\text{ML}}^T \phi(\mathbf{x}_n))^T. \quad (3.109)$$

- 3.7** (*) By using the technique of completing the square, verify the result (3.49) for the posterior distribution of the parameters \mathbf{w} in the linear basis function model in which \mathbf{m}_N and \mathbf{S}_N are defined by (3.50) and (3.51) respectively.
- 3.8** (**) **www** Consider the linear basis function model in Section 3.1, and suppose that we have already observed N data points, so that the posterior distribution over \mathbf{w} is given by (3.49). This posterior can be regarded as the prior for the next observation. By considering an additional data point $(\mathbf{x}_{N+1}, t_{N+1})$, and by completing the square in the exponential, show that the resulting posterior distribution is again given by (3.49) but with \mathbf{S}_N replaced by \mathbf{S}_{N+1} and \mathbf{m}_N replaced by \mathbf{m}_{N+1} .
- 3.9** (**) Repeat the previous exercise but instead of completing the square by hand, make use of the general result for linear-Gaussian models given by (2.116).
- 3.10** (**) **www** By making use of the result (2.115) to evaluate the integral in (3.57), verify that the predictive distribution for the Bayesian linear regression model is given by (3.58) in which the input-dependent variance is given by (3.59).
- 3.11** (**) We have seen that, as the size of a data set increases, the uncertainty associated with the posterior distribution over model parameters decreases. Make use of the matrix identity (Appendix C)

$$(\mathbf{M} + \mathbf{v}\mathbf{v}^T)^{-1} = \mathbf{M}^{-1} - \frac{(\mathbf{M}^{-1}\mathbf{v})(\mathbf{v}^T\mathbf{M}^{-1})}{1 + \mathbf{v}^T\mathbf{M}^{-1}\mathbf{v}} \quad (3.110)$$

to show that the uncertainty $\sigma_N^2(\mathbf{x})$ associated with the linear regression function given by (3.59) satisfies

$$\sigma_{N+1}^2(\mathbf{x}) \leq \sigma_N^2(\mathbf{x}). \quad (3.111)$$

- 3.12** (**) We saw in Section 2.3.6 that the conjugate prior for a Gaussian distribution with unknown mean and unknown precision (inverse variance) is a normal-gamma distribution. This property also holds for the case of the conditional Gaussian distribution $p(t|\mathbf{x}, \mathbf{w}, \beta)$ of the linear regression model. If we consider the likelihood function (3.10), then the conjugate prior for \mathbf{w} and β is given by

$$p(\mathbf{w}, \beta) = \mathcal{N}(\mathbf{w} | \mathbf{m}_0, \beta^{-1} \mathbf{S}_0) \text{Gam}(\beta | a_0, b_0). \quad (3.112)$$

3. LINEAR MODELS FOR REGRESSION

Show that the corresponding posterior distribution takes the same functional form, so that

$$p(\mathbf{w}, \beta | \mathbf{t}) = \mathcal{N}(\mathbf{w} | \mathbf{m}_N, \beta^{-1} \mathbf{S}_N) \text{Gam}(\beta | a_N, b_N) \quad (3.113)$$

and find expressions for the posterior parameters \mathbf{m}_N , \mathbf{S}_N , a_N , and b_N .

- 3.13** (**) Show that the predictive distribution $p(t|\mathbf{x}, \mathbf{t})$ for the model discussed in Exercise 3.12 is given by a Student's t-distribution of the form

$$p(t|\mathbf{x}, \mathbf{t}) = \text{St}(t|\mu, \lambda, \nu) \quad (3.114)$$

and obtain expressions for μ , λ and ν .

- 3.14** (**) In this exercise, we explore in more detail the properties of the equivalent kernel defined by (3.62), where \mathbf{S}_N is defined by (3.54). Suppose that the basis functions $\phi_j(\mathbf{x})$ are linearly independent and that the number N of data points is greater than the number M of basis functions. Furthermore, let one of the basis functions be constant, say $\phi_0(\mathbf{x}) = 1$. By taking suitable linear combinations of these basis functions, we can construct a new basis set $\psi_j(\mathbf{x})$ spanning the same space but that are orthonormal, so that

$$\sum_{n=1}^N \psi_j(\mathbf{x}_n) \psi_k(\mathbf{x}_n) = I_{jk} \quad (3.115)$$

where I_{jk} is defined to be 1 if $j = k$ and 0 otherwise, and we take $\psi_0(\mathbf{x}) = 1$. Show that for $\alpha = 0$, the equivalent kernel can be written as $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\psi}(\mathbf{x})^T \boldsymbol{\psi}(\mathbf{x}')$ where $\boldsymbol{\psi} = (\psi_1, \dots, \psi_M)^T$. Use this result to show that the kernel satisfies the summation constraint

$$\sum_{n=1}^N k(\mathbf{x}, \mathbf{x}_n) = 1. \quad (3.116)$$

- 3.15** (*) **www** Consider a linear basis function model for regression in which the parameters α and β are set using the evidence framework. Show that the function $E(\mathbf{m}_N)$ defined by (3.82) satisfies the relation $2E(\mathbf{m}_N) = N$.
- 3.16** (**) Derive the result (3.86) for the log evidence function $p(\mathbf{t}|\alpha, \beta)$ of the linear regression model by making use of (2.115) to evaluate the integral (3.77) directly.
- 3.17** (*) Show that the evidence function for the Bayesian linear regression model can be written in the form (3.78) in which $E(\mathbf{w})$ is defined by (3.79).
- 3.18** (**) **www** By completing the square over \mathbf{w} , show that the error function (3.79) in Bayesian linear regression can be written in the form (3.80).
- 3.19** (**) Show that the integration over \mathbf{w} in the Bayesian linear regression model gives the result (3.85). Hence show that the log marginal likelihood is given by (3.86).

- 3.20** (**) **www** Starting from (3.86) verify all of the steps needed to show that maximization of the log marginal likelihood function (3.86) with respect to α leads to the re-estimation equation (3.92).
- 3.21** (**) An alternative way to derive the result (3.92) for the optimal value of α in the evidence framework is to make use of the identity

$$\frac{d}{d\alpha} \ln |\mathbf{A}| = \text{Tr} \left(\mathbf{A}^{-1} \frac{d}{d\alpha} \mathbf{A} \right). \quad (3.117)$$

Prove this identity by considering the eigenvalue expansion of a real, symmetric matrix \mathbf{A} , and making use of the standard results for the determinant and trace of \mathbf{A} expressed in terms of its eigenvalues (Appendix C). Then make use of (3.117) to derive (3.92) starting from (3.86).

- 3.22** (**) Starting from (3.86) verify all of the steps needed to show that maximization of the log marginal likelihood function (3.86) with respect to β leads to the re-estimation equation (3.95).
- 3.23** (**) **www** Show that the marginal probability of the data, in other words the model evidence, for the model described in Exercise 3.12 is given by

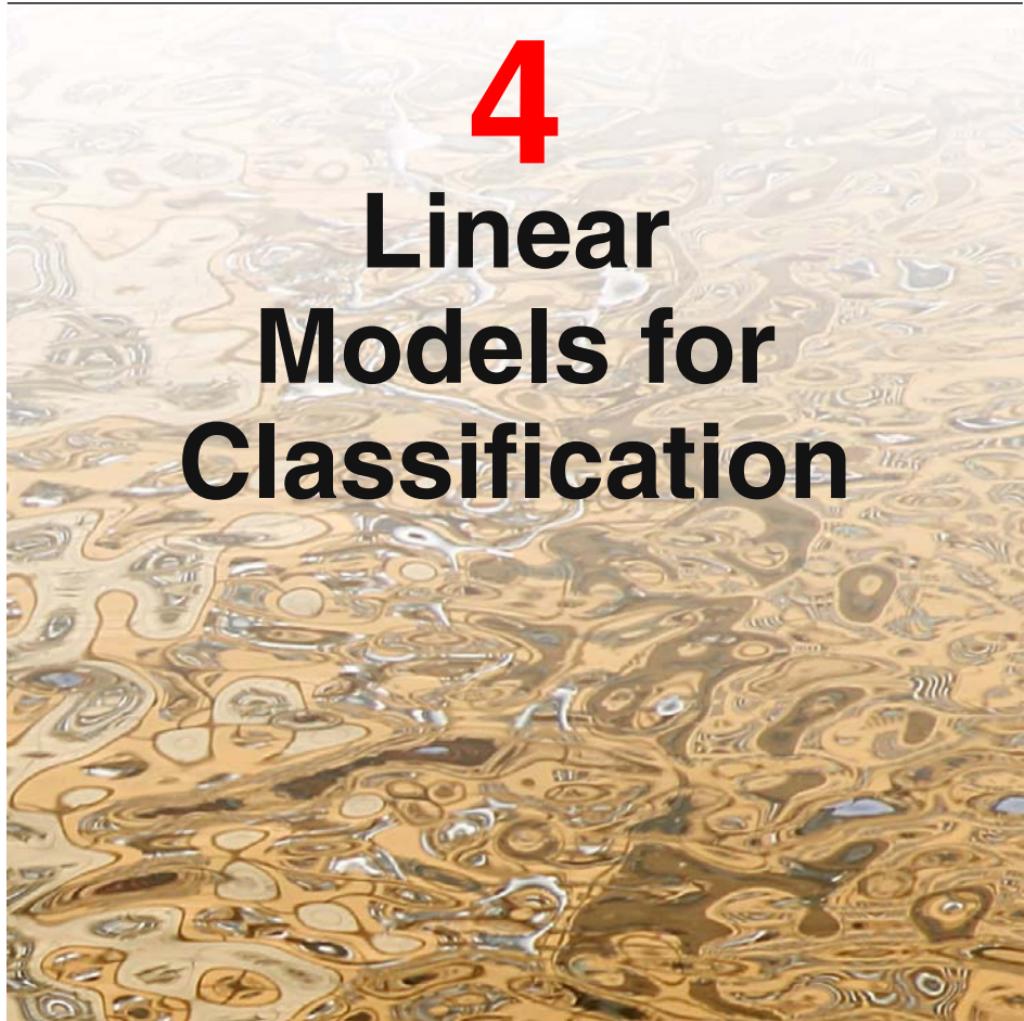
$$p(\mathbf{t}) = \frac{1}{(2\pi)^{N/2}} \frac{b_0^{a_0}}{b_N^{a_N}} \frac{\Gamma(a_N)}{\Gamma(a_0)} \frac{|\mathbf{S}_N|^{1/2}}{|\mathbf{S}_0|^{1/2}} \quad (3.118)$$

by first marginalizing with respect to \mathbf{w} and then with respect to β .

- 3.24** (**) Repeat the previous exercise but now use Bayes' theorem in the form

$$p(\mathbf{t}) = \frac{p(\mathbf{t}|\mathbf{w}, \beta)p(\mathbf{w}, \beta)}{p(\mathbf{w}, \beta|\mathbf{t})} \quad (3.119)$$

and then substitute for the prior and posterior distributions and the likelihood function in order to derive the result (3.118).



4 Linear Models for Classification

In the previous chapter, we explored a class of regression models having particularly simple analytical and computational properties. We now discuss an analogous class of models for solving classification problems. The goal in classification is to take an input vector \mathbf{x} and to assign it to one of K discrete classes \mathcal{C}_k where $k = 1, \dots, K$. In the most common scenario, the classes are taken to be disjoint, so that each input is assigned to one and only one class. The input space is thereby divided into *decision regions* whose boundaries are called *decision boundaries* or *decision surfaces*. In this chapter, we consider linear models for classification, by which we mean that the decision surfaces are linear functions of the input vector \mathbf{x} and hence are defined by $(D - 1)$ -dimensional hyperplanes within the D -dimensional input space. Data sets whose classes can be separated exactly by linear decision surfaces are said to be *linearly separable*.

For regression problems, the target variable t was simply the vector of real numbers whose values we wish to predict. In the case of classification, there are various

ways of using target values to represent class labels. For probabilistic models, the most convenient, in the case of two-class problems, is the binary representation in which there is a single target variable $t \in \{0, 1\}$ such that $t = 1$ represents class \mathcal{C}_1 and $t = 0$ represents class \mathcal{C}_2 . We can interpret the value of t as the probability that the class is \mathcal{C}_1 , with the values of probability taking only the extreme values of 0 and 1. For $K > 2$ classes, it is convenient to use a 1-of- K coding scheme in which \mathbf{t} is a vector of length K such that if the class is \mathcal{C}_j , then all elements t_k of \mathbf{t} are zero except element t_j , which takes the value 1. For instance, if we have $K = 5$ classes, then a pattern from class 2 would be given the target vector

$$\mathbf{t} = (0, 1, 0, 0, 0)^T. \quad (4.1)$$

Again, we can interpret the value of t_k as the probability that the class is \mathcal{C}_k . For nonprobabilistic models, alternative choices of target variable representation will sometimes prove convenient.

In Chapter 1, we identified three distinct approaches to the classification problem. The simplest involves constructing a *discriminant function* that directly assigns each vector \mathbf{x} to a specific class. A more powerful approach, however, models the conditional probability distribution $p(\mathcal{C}_k|\mathbf{x})$ in an inference stage, and then subsequently uses this distribution to make optimal decisions. By separating inference and decision, we gain numerous benefits, as discussed in Section 1.5.4. There are two different approaches to determining the conditional probabilities $p(\mathcal{C}_k|\mathbf{x})$. One technique is to model them directly, for example by representing them as parametric models and then optimizing the parameters using a training set. Alternatively, we can adopt a generative approach in which we model the class-conditional densities given by $p(\mathbf{x}|\mathcal{C}_k)$, together with the prior probabilities $p(\mathcal{C}_k)$ for the classes, and then we compute the required posterior probabilities using Bayes' theorem

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}. \quad (4.2)$$

We shall discuss examples of all three approaches in this chapter.

In the linear regression models considered in Chapter 3, the model prediction $y(\mathbf{x}, \mathbf{w})$ was given by a linear function of the parameters \mathbf{w} . In the simplest case, the model is also linear in the input variables and therefore takes the form $y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$, so that y is a real number. For classification problems, however, we wish to predict discrete class labels, or more generally posterior probabilities that lie in the range $(0, 1)$. To achieve this, we consider a generalization of this model in which we transform the linear function of \mathbf{w} using a nonlinear function $f(\cdot)$ so that

$$y(\mathbf{x}) = f(\mathbf{w}^T \mathbf{x} + w_0). \quad (4.3)$$

In the machine learning literature $f(\cdot)$ is known as an *activation function*, whereas its inverse is called a *link function* in the statistics literature. The decision surfaces correspond to $y(\mathbf{x}) = \text{constant}$, so that $\mathbf{w}^T \mathbf{x} + w_0 = \text{constant}$ and hence the decision surfaces are linear functions of \mathbf{x} , even if the function $f(\cdot)$ is nonlinear. For this reason, the class of models described by (4.3) are called *generalized linear models*.

(McCullagh and Nelder, 1989). Note, however, that in contrast to the models used for regression, they are no longer linear in the parameters due to the presence of the nonlinear function $f(\cdot)$. This will lead to more complex analytical and computational properties than for linear regression models. Nevertheless, these models are still relatively simple compared to the more general nonlinear models that will be studied in subsequent chapters.

The algorithms discussed in this chapter will be equally applicable if we first make a fixed nonlinear transformation of the input variables using a vector of basis functions $\phi(\mathbf{x})$ as we did for regression models in Chapter 3. We begin by considering classification directly in the original input space \mathbf{x} , while in Section 4.3 we shall find it convenient to switch to a notation involving basis functions for consistency with later chapters.

4.1. Discriminant Functions

A discriminant is a function that takes an input vector \mathbf{x} and assigns it to one of K classes, denoted \mathcal{C}_k . In this chapter, we shall restrict attention to *linear discriminants*, namely those for which the decision surfaces are hyperplanes. To simplify the discussion, we consider first the case of two classes and then investigate the extension to $K > 2$ classes.

4.1.1 Two classes

The simplest representation of a linear discriminant function is obtained by taking a linear function of the input vector so that

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 \quad (4.4)$$

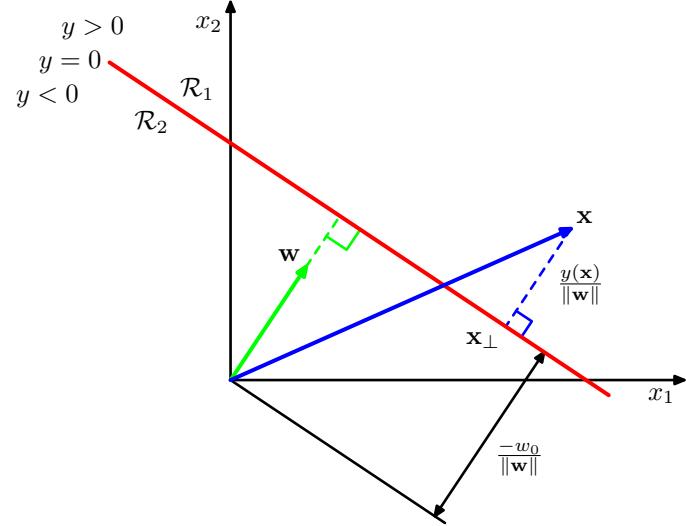
where \mathbf{w} is called a *weight vector*, and w_0 is a *bias* (not to be confused with bias in the statistical sense). The negative of the bias is sometimes called a *threshold*. An input vector \mathbf{x} is assigned to class \mathcal{C}_1 if $y(\mathbf{x}) \geq 0$ and to class \mathcal{C}_2 otherwise. The corresponding decision boundary is therefore defined by the relation $y(\mathbf{x}) = 0$, which corresponds to a $(D - 1)$ -dimensional hyperplane within the D -dimensional input space. Consider two points \mathbf{x}_A and \mathbf{x}_B both of which lie on the decision surface. Because $y(\mathbf{x}_A) = y(\mathbf{x}_B) = 0$, we have $\mathbf{w}^T(\mathbf{x}_A - \mathbf{x}_B) = 0$ and hence the vector \mathbf{w} is orthogonal to every vector lying within the decision surface, and so \mathbf{w} determines the orientation of the decision surface. Similarly, if \mathbf{x} is a point on the decision surface, then $y(\mathbf{x}) = 0$, and so the normal distance from the origin to the decision surface is given by

$$\frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} = -\frac{w_0}{\|\mathbf{w}\|}. \quad (4.5)$$

We therefore see that the bias parameter w_0 determines the location of the decision surface. These properties are illustrated for the case of $D = 2$ in Figure 4.1.

Furthermore, we note that the value of $y(\mathbf{x})$ gives a signed measure of the perpendicular distance r of the point \mathbf{x} from the decision surface. To see this, consider

Figure 4.1 Illustration of the geometry of a linear discriminant function in two dimensions. The decision surface, shown in red, is perpendicular to \mathbf{w} , and its displacement from the origin is controlled by the bias parameter w_0 . Also, the signed orthogonal distance of a general point \mathbf{x} from the decision surface is given by $y(\mathbf{x})/\|\mathbf{w}\|$.



an arbitrary point \mathbf{x} and let \mathbf{x}_\perp be its orthogonal projection onto the decision surface, so that

$$\mathbf{x} = \mathbf{x}_\perp + r \frac{\mathbf{w}}{\|\mathbf{w}\|}. \quad (4.6)$$

Multiplying both sides of this result by \mathbf{w}^T and adding w_0 , and making use of $y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$ and $y(\mathbf{x}_\perp) = \mathbf{w}^T \mathbf{x}_\perp + w_0 = 0$, we have

$$r = \frac{y(\mathbf{x})}{\|\mathbf{w}\|}. \quad (4.7)$$

This result is illustrated in Figure 4.1.

As with the linear regression models in Chapter 3, it is sometimes convenient to use a more compact notation in which we introduce an additional dummy ‘input’ value $x_0 = 1$ and then define $\tilde{\mathbf{w}} = (w_0, \mathbf{w})$ and $\tilde{\mathbf{x}} = (x_0, \mathbf{x})$ so that

$$y(\mathbf{x}) = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}. \quad (4.8)$$

In this case, the decision surfaces are D -dimensional hyperplanes passing through the origin of the $D + 1$ -dimensional expanded input space.

4.1.2 Multiple classes

Now consider the extension of linear discriminants to $K > 2$ classes. We might be tempted to build a K -class discriminant by combining a number of two-class discriminant functions. However, this leads to some serious difficulties (Duda and Hart, 1973) as we now show.

Consider the use of $K - 1$ classifiers each of which solves a two-class problem of separating points in a particular class C_k from points not in that class. This is known as a *one-versus-the-rest* classifier. The left-hand example in Figure 4.2 shows an

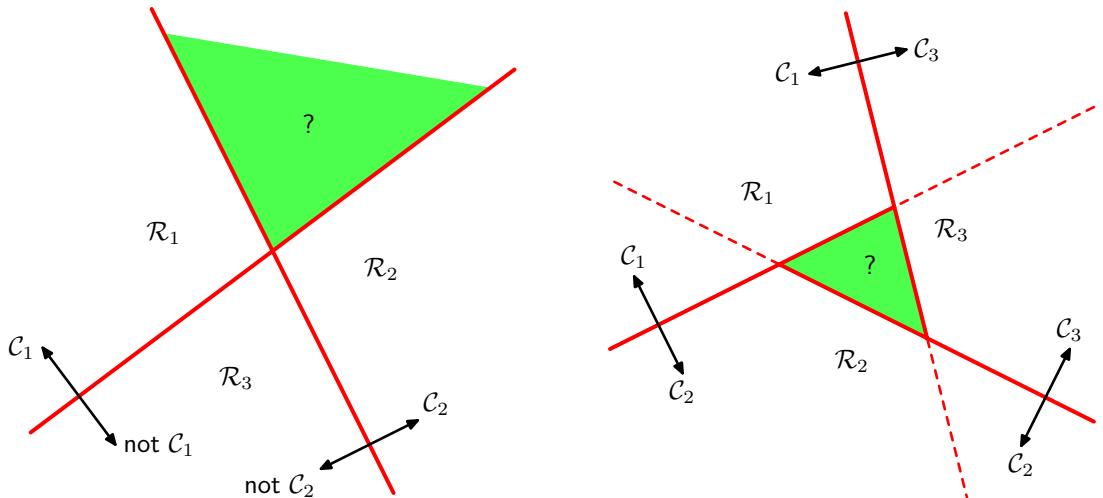


Figure 4.2 Attempting to construct a K class discriminant from a set of two class discriminants leads to ambiguous regions, shown in green. On the left is an example involving the use of two discriminants designed to distinguish points in class C_k from points not in class C_k . On the right is an example involving three discriminant functions each of which is used to separate a pair of classes C_k and C_j .

example involving three classes where this approach leads to regions of input space that are ambiguously classified.

An alternative is to introduce $K(K - 1)/2$ binary discriminant functions, one for every possible pair of classes. This is known as a *one-versus-one* classifier. Each point is then classified according to a majority vote amongst the discriminant functions. However, this too runs into the problem of ambiguous regions, as illustrated in the right-hand diagram of Figure 4.2.

We can avoid these difficulties by considering a single K -class discriminant comprising K linear functions of the form

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0} \quad (4.9)$$

and then assigning a point \mathbf{x} to class C_k if $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$. The decision boundary between class C_k and class C_j is therefore given by $y_k(\mathbf{x}) = y_j(\mathbf{x})$ and hence corresponds to a $(D - 1)$ -dimensional hyperplane defined by

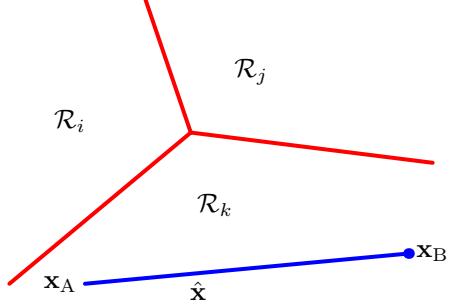
$$(\mathbf{w}_k - \mathbf{w}_j)^T \mathbf{x} + (w_{k0} - w_{j0}) = 0. \quad (4.10)$$

This has the same form as the decision boundary for the two-class case discussed in Section 4.1.1, and so analogous geometrical properties apply.

The decision regions of such a discriminant are always singly connected and convex. To see this, consider two points \mathbf{x}_A and \mathbf{x}_B both of which lie inside decision region \mathcal{R}_k , as illustrated in Figure 4.3. Any point $\hat{\mathbf{x}}$ that lies on the line connecting \mathbf{x}_A and \mathbf{x}_B can be expressed in the form

$$\hat{\mathbf{x}} = \lambda \mathbf{x}_A + (1 - \lambda) \mathbf{x}_B \quad (4.11)$$

Figure 4.3 Illustration of the decision regions for a multiclass linear discriminant, with the decision boundaries shown in red. If two points \mathbf{x}_A and \mathbf{x}_B both lie inside the same decision region \mathcal{R}_k , then any point $\hat{\mathbf{x}}$ that lies on the line connecting these two points must also lie in \mathcal{R}_k , and hence the decision region must be singly connected and convex.



where $0 \leq \lambda \leq 1$. From the linearity of the discriminant functions, it follows that

$$y_k(\hat{\mathbf{x}}) = \lambda y_k(\mathbf{x}_A) + (1 - \lambda)y_k(\mathbf{x}_B). \quad (4.12)$$

Because both \mathbf{x}_A and \mathbf{x}_B lie inside \mathcal{R}_k , it follows that $y_k(\mathbf{x}_A) > y_j(\mathbf{x}_A)$, and $y_k(\mathbf{x}_B) > y_j(\mathbf{x}_B)$, for all $j \neq k$, and hence $y_k(\hat{\mathbf{x}}) > y_j(\hat{\mathbf{x}})$, and so $\hat{\mathbf{x}}$ also lies inside \mathcal{R}_k . Thus \mathcal{R}_k is singly connected and convex.

Note that for two classes, we can either employ the formalism discussed here, based on two discriminant functions $y_1(\mathbf{x})$ and $y_2(\mathbf{x})$, or else use the simpler but equivalent formulation described in Section 4.1.1 based on a single discriminant function $y(\mathbf{x})$.

We now explore three approaches to learning the parameters of linear discriminant functions, based on least squares, Fisher's linear discriminant, and the perceptron algorithm.

4.1.3 Least squares for classification

In Chapter 3, we considered models that were linear functions of the parameters, and we saw that the minimization of a sum-of-squares error function led to a simple closed-form solution for the parameter values. It is therefore tempting to see if we can apply the same formalism to classification problems. Consider a general classification problem with K classes, with a 1-of- K binary coding scheme for the target vector \mathbf{t} . One justification for using least squares in such a context is that it approximates the conditional expectation $\mathbb{E}[\mathbf{t}|\mathbf{x}]$ of the target values given the input vector. For the binary coding scheme, this conditional expectation is given by the vector of posterior class probabilities. Unfortunately, however, these probabilities are typically approximated rather poorly, indeed the approximations can have values outside the range $(0, 1)$, due to the limited flexibility of a linear model as we shall see shortly.

Each class \mathcal{C}_k is described by its own linear model so that

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0} \quad (4.13)$$

where $k = 1, \dots, K$. We can conveniently group these together using vector notation so that

$$\mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^T \widetilde{\mathbf{x}} \quad (4.14)$$

where $\widetilde{\mathbf{W}}$ is a matrix whose k^{th} column comprises the $D + 1$ -dimensional vector $\widetilde{\mathbf{w}}_k = (w_{k0}, \mathbf{w}_k^T)^T$ and $\widetilde{\mathbf{x}}$ is the corresponding augmented input vector $(1, \mathbf{x}^T)^T$ with a dummy input $x_0 = 1$. This representation was discussed in detail in Section 3.1. A new input \mathbf{x} is then assigned to the class for which the output $y_k = \widetilde{\mathbf{w}}_k^T \widetilde{\mathbf{x}}$ is largest.

We now determine the parameter matrix $\widetilde{\mathbf{W}}$ by minimizing a sum-of-squares error function, as we did for regression in Chapter 3. Consider a training data set $\{\mathbf{x}_n, \mathbf{t}_n\}$ where $n = 1, \dots, N$, and define a matrix \mathbf{T} whose n^{th} row is the vector \mathbf{t}_n^T , together with a matrix $\widetilde{\mathbf{X}}$ whose n^{th} row is $\widetilde{\mathbf{x}}_n^T$. The sum-of-squares error function can then be written as

$$E_D(\widetilde{\mathbf{W}}) = \frac{1}{2} \text{Tr} \left\{ (\widetilde{\mathbf{X}} \widetilde{\mathbf{W}} - \mathbf{T})^T (\widetilde{\mathbf{X}} \widetilde{\mathbf{W}} - \mathbf{T}) \right\}. \quad (4.15)$$

Setting the derivative with respect to $\widetilde{\mathbf{W}}$ to zero, and rearranging, we then obtain the solution for $\widetilde{\mathbf{W}}$ in the form

$$\widetilde{\mathbf{W}} = (\widetilde{\mathbf{X}}^T \widetilde{\mathbf{X}})^{-1} \widetilde{\mathbf{X}}^T \mathbf{T} = \widetilde{\mathbf{X}}^\dagger \mathbf{T} \quad (4.16)$$

where $\widetilde{\mathbf{X}}^\dagger$ is the pseudo-inverse of the matrix $\widetilde{\mathbf{X}}$, as discussed in Section 3.1.1. We then obtain the discriminant function in the form

$$\mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^T \widetilde{\mathbf{x}} = \mathbf{T}^T \left(\widetilde{\mathbf{X}}^\dagger \right)^T \widetilde{\mathbf{x}}. \quad (4.17)$$

An interesting property of least-squares solutions with multiple target variables is that if every target vector in the training set satisfies some linear constraint

$$\mathbf{a}^T \mathbf{t}_n + b = 0 \quad (4.18)$$

for some constants \mathbf{a} and b , then the model prediction for any value of \mathbf{x} will satisfy the same constraint so that

$$\mathbf{a}^T \mathbf{y}(\mathbf{x}) + b = 0. \quad (4.19)$$

Thus if we use a 1-of- K coding scheme for K classes, then the predictions made by the model will have the property that the elements of $\mathbf{y}(\mathbf{x})$ will sum to 1 for any value of \mathbf{x} . However, this summation constraint alone is not sufficient to allow the model outputs to be interpreted as probabilities because they are not constrained to lie within the interval $(0, 1)$.

The least-squares approach gives an exact closed-form solution for the discriminant function parameters. However, even as a discriminant function (where we use it to make decisions directly and dispense with any probabilistic interpretation) it suffers from some severe problems. We have already seen that least-squares solutions lack robustness to outliers, and this applies equally to the classification application, as illustrated in Figure 4.4. Here we see that the additional data points in the right-hand figure produce a significant change in the location of the decision boundary, even though these point would be correctly classified by the original decision boundary in the left-hand figure. The sum-of-squares error function penalizes predictions that are ‘too correct’ in that they lie a long way on the correct side of the decision

Exercise 4.2

Section 2.3.7

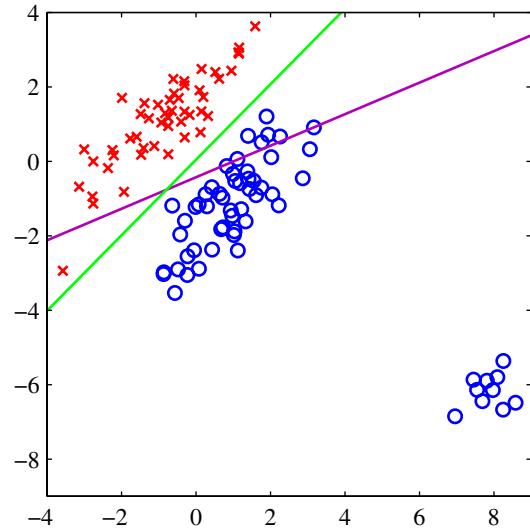
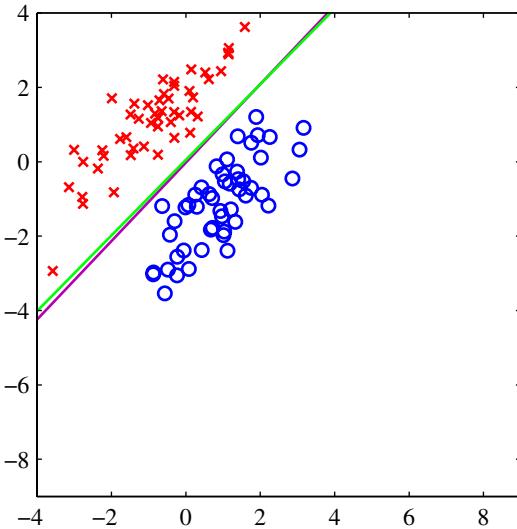


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

boundary. In Section 7.1.2, we shall consider several alternative error functions for classification and we shall see that they do not suffer from this difficulty.

However, problems with least squares can be more severe than simply lack of robustness, as illustrated in Figure 4.5. This shows a synthetic data set drawn from three classes in a two-dimensional input space (x_1, x_2) , having the property that linear decision boundaries can give excellent separation between the classes. Indeed, the technique of logistic regression, described later in this chapter, gives a satisfactory solution as seen in the right-hand plot. However, the least-squares solution gives poor results, with only a small region of the input space assigned to the green class.

The failure of least squares should not surprise us when we recall that it corresponds to maximum likelihood under the assumption of a Gaussian conditional distribution, whereas binary target vectors clearly have a distribution that is far from Gaussian. By adopting more appropriate probabilistic models, we shall obtain classification techniques with much better properties than least squares. For the moment, however, we continue to explore alternative nonprobabilistic methods for setting the parameters in the linear classification models.

4.1.4 Fisher's linear discriminant

One way to view a linear classification model is in terms of dimensionality reduction. Consider first the case of two classes, and suppose we take the D -

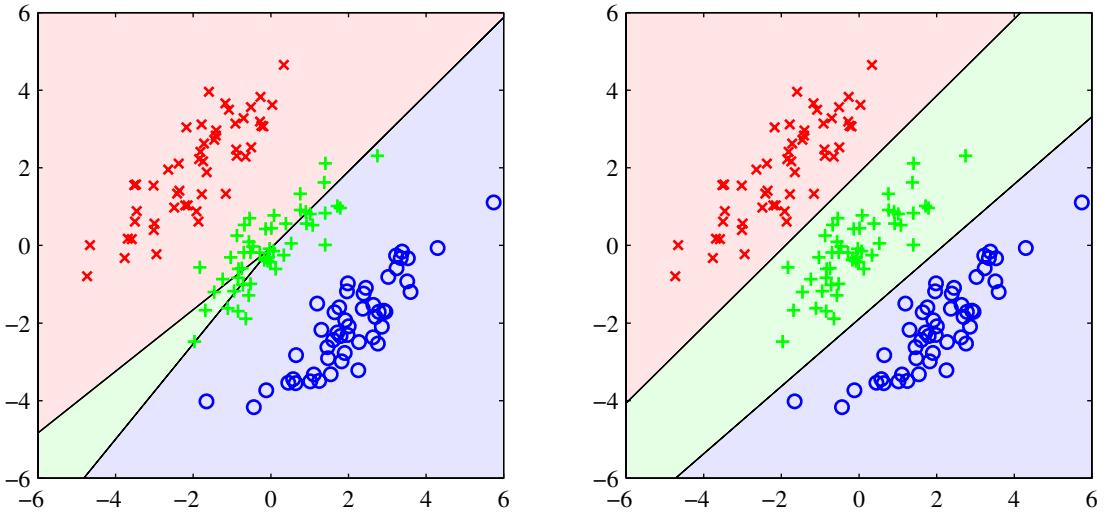


Figure 4.5 Example of a synthetic data set comprising three classes, with training data points denoted in red (\times), green ($+$), and blue (\circ). Lines denote the decision boundaries, and the background colours denote the respective classes of the decision regions. On the left is the result of using a least-squares discriminant. We see that the region of input space assigned to the green class is too small and so most of the points from this class are misclassified. On the right is the result of using logistic regressions as described in Section 4.3.2 showing correct classification of the training data.

dimensional input vector \mathbf{x} and project it down to one dimension using

$$y = \mathbf{w}^T \mathbf{x}. \quad (4.20)$$

If we place a threshold on y and classify $y \geq -w_0$ as class \mathcal{C}_1 , and otherwise class \mathcal{C}_2 , then we obtain our standard linear classifier discussed in the previous section. In general, the projection onto one dimension leads to a considerable loss of information, and classes that are well separated in the original D -dimensional space may become strongly overlapping in one dimension. However, by adjusting the components of the weight vector \mathbf{w} , we can select a projection that maximizes the class separation. To begin with, consider a two-class problem in which there are N_1 points of class \mathcal{C}_1 and N_2 points of class \mathcal{C}_2 , so that the mean vectors of the two classes are given by

$$\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in \mathcal{C}_1} \mathbf{x}_n, \quad \mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in \mathcal{C}_2} \mathbf{x}_n. \quad (4.21)$$

The simplest measure of the separation of the classes, when projected onto \mathbf{w} , is the separation of the projected class means. This suggests that we might choose \mathbf{w} so as to maximize

$$m_2 - m_1 = \mathbf{w}^T (\mathbf{m}_2 - \mathbf{m}_1) \quad (4.22)$$

where

$$m_k = \mathbf{w}^T \mathbf{m}_k \quad (4.23)$$

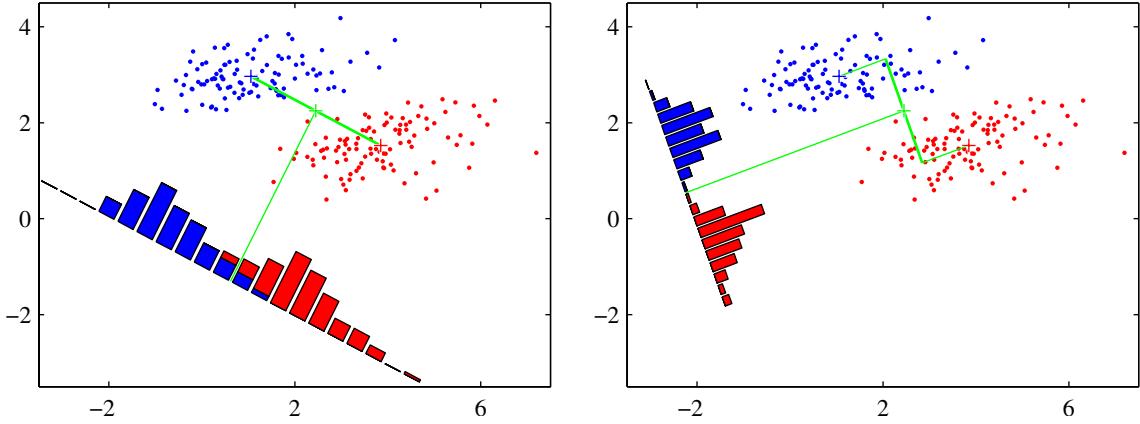


Figure 4.6 The left plot shows samples from two classes (depicted in red and blue) along with the histograms resulting from projection onto the line joining the class means. Note that there is considerable class overlap in the projected space. The right plot shows the corresponding projection based on the Fisher linear discriminant, showing the greatly improved class separation.

Appendix E Exercise 4.4

is the mean of the projected data from class \mathcal{C}_k . However, this expression can be made arbitrarily large simply by increasing the magnitude of \mathbf{w} . To solve this problem, we could constrain \mathbf{w} to have unit length, so that $\sum_i w_i^2 = 1$. Using a Lagrange multiplier to perform the constrained maximization, we then find that $\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1)$. There is still a problem with this approach, however, as illustrated in Figure 4.6. This shows two classes that are well separated in the original two-dimensional space (x_1, x_2) but that have considerable overlap when projected onto the line joining their means. This difficulty arises from the strongly nondiagonal covariances of the class distributions. The idea proposed by Fisher is to maximize a function that will give a large separation between the projected class means while also giving a small variance within each class, thereby minimizing the class overlap.

The projection formula (4.20) transforms the set of labelled data points in \mathbf{x} into a labelled set in the one-dimensional space y . The within-class variance of the transformed data from class \mathcal{C}_k is therefore given by

$$s_k^2 = \sum_{n \in \mathcal{C}_k} (y_n - m_k)^2 \quad (4.24)$$

where $y_n = \mathbf{w}^\top \mathbf{x}_n$. We can define the total within-class variance for the whole data set to be simply $s_1^2 + s_2^2$. The Fisher criterion is defined to be the ratio of the between-class variance to the within-class variance and is given by

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}. \quad (4.25)$$

Exercise 4.5

We can make the dependence on \mathbf{w} explicit by using (4.20), (4.23), and (4.24) to rewrite the Fisher criterion in the form

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \quad (4.26)$$

where \mathbf{S}_B is the *between-class* covariance matrix and is given by

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T \quad (4.27)$$

and \mathbf{S}_W is the total *within-class* covariance matrix, given by

$$\mathbf{S}_W = \sum_{n \in \mathcal{C}_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in \mathcal{C}_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T. \quad (4.28)$$

Differentiating (4.26) with respect to \mathbf{w} , we find that $J(\mathbf{w})$ is maximized when

$$(\mathbf{w}^T \mathbf{S}_B \mathbf{w}) \mathbf{S}_W \mathbf{w} = (\mathbf{w}^T \mathbf{S}_W \mathbf{w}) \mathbf{S}_B \mathbf{w}. \quad (4.29)$$

From (4.27), we see that $\mathbf{S}_B \mathbf{w}$ is always in the direction of $(\mathbf{m}_2 - \mathbf{m}_1)$. Furthermore, we do not care about the magnitude of \mathbf{w} , only its direction, and so we can drop the scalar factors $(\mathbf{w}^T \mathbf{S}_B \mathbf{w})$ and $(\mathbf{w}^T \mathbf{S}_W \mathbf{w})$. Multiplying both sides of (4.29) by \mathbf{S}_W^{-1} we then obtain

$$\mathbf{w} \propto \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1). \quad (4.30)$$

Note that if the within-class covariance is isotropic, so that \mathbf{S}_W is proportional to the unit matrix, we find that \mathbf{w} is proportional to the difference of the class means, as discussed above.

The result (4.30) is known as *Fisher's linear discriminant*, although strictly it is not a discriminant but rather a specific choice of direction for projection of the data down to one dimension. However, the projected data can subsequently be used to construct a discriminant, by choosing a threshold y_0 so that we classify a new point as belonging to \mathcal{C}_1 if $y(\mathbf{x}) \geq y_0$ and classify it as belonging to \mathcal{C}_2 otherwise. For example, we can model the class-conditional densities $p(y|\mathcal{C}_k)$ using Gaussian distributions and then use the techniques of Section 1.2.4 to find the parameters of the Gaussian distributions by maximum likelihood. Having found Gaussian approximations to the projected classes, the formalism of Section 1.5.1 then gives an expression for the optimal threshold. Some justification for the Gaussian assumption comes from the central limit theorem by noting that $y = \mathbf{w}^T \mathbf{x}$ is the sum of a set of random variables.

4.1.5 Relation to least squares

The least-squares approach to the determination of a linear discriminant was based on the goal of making the model predictions as close as possible to a set of target values. By contrast, the Fisher criterion was derived by requiring maximum class separation in the output space. It is interesting to see the relationship between these two approaches. In particular, we shall show that, for the two-class problem, the Fisher criterion can be obtained as a special case of least squares.

So far we have considered 1-of- K coding for the target values. If, however, we adopt a slightly different target coding scheme, then the least-squares solution for

the weights becomes equivalent to the Fisher solution (Duda and Hart, 1973). In particular, we shall take the targets for class \mathcal{C}_1 to be N/N_1 , where N_1 is the number of patterns in class \mathcal{C}_1 , and N is the total number of patterns. This target value approximates the reciprocal of the prior probability for class \mathcal{C}_1 . For class \mathcal{C}_2 , we shall take the targets to be $-N/N_2$, where N_2 is the number of patterns in class \mathcal{C}_2 .

The sum-of-squares error function can be written

$$E = \frac{1}{2} \sum_{n=1}^N (\mathbf{w}^T \mathbf{x}_n + w_0 - t_n)^2. \quad (4.31)$$

Setting the derivatives of E with respect to w_0 and \mathbf{w} to zero, we obtain respectively

$$\sum_{n=1}^N (\mathbf{w}^T \mathbf{x}_n + w_0 - t_n) = 0 \quad (4.32)$$

$$\sum_{n=1}^N (\mathbf{w}^T \mathbf{x}_n + w_0 - t_n) \mathbf{x}_n = 0. \quad (4.33)$$

From (4.32), and making use of our choice of target coding scheme for the t_n , we obtain an expression for the bias in the form

$$w_0 = -\mathbf{w}^T \mathbf{m} \quad (4.34)$$

where we have used

$$\sum_{n=1}^N t_n = N_1 \frac{N}{N_1} - N_2 \frac{N}{N_2} = 0 \quad (4.35)$$

and where \mathbf{m} is the mean of the total data set and is given by

$$\mathbf{m} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n = \frac{1}{N} (N_1 \mathbf{m}_1 + N_2 \mathbf{m}_2). \quad (4.36)$$

After some straightforward algebra, and again making use of the choice of t_n , the second equation (4.33) becomes

$$\left(\mathbf{S}_W + \frac{N_1 N_2}{N} \mathbf{S}_B \right) \mathbf{w} = N(\mathbf{m}_1 - \mathbf{m}_2) \quad (4.37)$$

where \mathbf{S}_W is defined by (4.28), \mathbf{S}_B is defined by (4.27), and we have substituted for the bias using (4.34). Using (4.27), we note that $\mathbf{S}_B \mathbf{w}$ is always in the direction of $(\mathbf{m}_2 - \mathbf{m}_1)$. Thus we can write

$$\mathbf{w} \propto \mathbf{S}_W^{-1} (\mathbf{m}_2 - \mathbf{m}_1) \quad (4.38)$$

where we have ignored irrelevant scale factors. Thus the weight vector coincides with that found from the Fisher criterion. In addition, we have also found an expression for the bias value w_0 given by (4.34). This tells us that a new vector \mathbf{x} should be classified as belonging to class \mathcal{C}_1 if $y(\mathbf{x}) = \mathbf{w}^T (\mathbf{x} - \mathbf{m}) > 0$ and class \mathcal{C}_2 otherwise.

Exercise 4.6

4.1.6 Fisher's discriminant for multiple classes

We now consider the generalization of the Fisher discriminant to $K > 2$ classes, and we shall assume that the dimensionality D of the input space is greater than the number K of classes. Next, we introduce $D' > 1$ linear ‘features’ $y_k = \mathbf{w}_k^T \mathbf{x}$, where $k = 1, \dots, D'$. These feature values can conveniently be grouped together to form a vector \mathbf{y} . Similarly, the weight vectors $\{\mathbf{w}_k\}$ can be considered to be the columns of a matrix \mathbf{W} , so that

$$\mathbf{y} = \mathbf{W}^T \mathbf{x}. \quad (4.39)$$

Note that again we are not including any bias parameters in the definition of \mathbf{y} . The generalization of the within-class covariance matrix to the case of K classes follows from (4.28) to give

$$\mathbf{S}_W = \sum_{k=1}^K \mathbf{S}_k \quad (4.40)$$

where

$$\mathbf{S}_k = \sum_{n \in \mathcal{C}_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T \quad (4.41)$$

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{n \in \mathcal{C}_k} \mathbf{x}_n \quad (4.42)$$

and N_k is the number of patterns in class \mathcal{C}_k . In order to find a generalization of the between-class covariance matrix, we follow Duda and Hart (1973) and consider first the total covariance matrix

$$\mathbf{S}_T = \sum_{n=1}^N (\mathbf{x}_n - \mathbf{m})(\mathbf{x}_n - \mathbf{m})^T \quad (4.43)$$

where \mathbf{m} is the mean of the total data set

$$\mathbf{m} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n = \frac{1}{N} \sum_{k=1}^K N_k \mathbf{m}_k \quad (4.44)$$

and $N = \sum_k N_k$ is the total number of data points. The total covariance matrix can be decomposed into the sum of the within-class covariance matrix, given by (4.40) and (4.41), plus an additional matrix \mathbf{S}_B , which we identify as a measure of the between-class covariance

$$\mathbf{S}_T = \mathbf{S}_W + \mathbf{S}_B \quad (4.45)$$

where

$$\mathbf{S}_B = \sum_{k=1}^K N_k (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T. \quad (4.46)$$

These covariance matrices have been defined in the original \mathbf{x} -space. We can now define similar matrices in the projected D' -dimensional \mathbf{y} -space

$$\mathbf{s}_W = \sum_{k=1}^K \sum_{n \in \mathcal{C}_k} (\mathbf{y}_n - \boldsymbol{\mu}_k)(\mathbf{y}_n - \boldsymbol{\mu}_k)^T \quad (4.47)$$

and

$$\mathbf{s}_B = \sum_{k=1}^K N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T \quad (4.48)$$

where

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n \in \mathcal{C}_k} \mathbf{y}_n, \quad \boldsymbol{\mu} = \frac{1}{N} \sum_{k=1}^K N_k \boldsymbol{\mu}_k. \quad (4.49)$$

Again we wish to construct a scalar that is large when the between-class covariance is large and when the within-class covariance is small. There are now many possible choices of criterion (Fukunaga, 1990). One example is given by

$$J(\mathbf{W}) = \text{Tr} \left\{ \mathbf{s}_W^{-1} \mathbf{s}_B \right\}. \quad (4.50)$$

This criterion can then be rewritten as an explicit function of the projection matrix \mathbf{W} in the form

$$J(\mathbf{w}) = \text{Tr} \left\{ (\mathbf{W} \mathbf{S}_W \mathbf{W}^T)^{-1} (\mathbf{W} \mathbf{S}_B \mathbf{W}^T) \right\}. \quad (4.51)$$

Maximization of such criteria is straightforward, though somewhat involved, and is discussed at length in Fukunaga (1990). The weight values are determined by those eigenvectors of $\mathbf{S}_W^{-1} \mathbf{S}_B$ that correspond to the D' largest eigenvalues.

There is one important result that is common to all such criteria, which is worth emphasizing. We first note from (4.46) that \mathbf{S}_B is composed of the sum of K matrices, each of which is an outer product of two vectors and therefore of rank 1. In addition, only $(K - 1)$ of these matrices are independent as a result of the constraint (4.44). Thus, \mathbf{S}_B has rank at most equal to $(K - 1)$ and so there are at most $(K - 1)$ nonzero eigenvalues. This shows that the projection onto the $(K - 1)$ -dimensional subspace spanned by the eigenvectors of \mathbf{S}_B does not alter the value of $J(\mathbf{w})$, and so we are therefore unable to find more than $(K - 1)$ linear ‘features’ by this means (Fukunaga, 1990).

4.1.7 The perceptron algorithm

Another example of a linear discriminant model is the perceptron of Rosenblatt (1962), which occupies an important place in the history of pattern recognition algorithms. It corresponds to a two-class model in which the input vector \mathbf{x} is first transformed using a fixed nonlinear transformation to give a feature vector $\phi(\mathbf{x})$, and this is then used to construct a generalized linear model of the form

$$y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x})) \quad (4.52)$$

where the nonlinear activation function $f(\cdot)$ is given by a step function of the form

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0. \end{cases} \quad (4.53)$$

The vector $\phi(\mathbf{x})$ will typically include a bias component $\phi_0(\mathbf{x}) = 1$. In earlier discussions of two-class classification problems, we have focussed on a target coding scheme in which $t \in \{0, 1\}$, which is appropriate in the context of probabilistic models. For the perceptron, however, it is more convenient to use target values $t = +1$ for class C_1 and $t = -1$ for class C_2 , which matches the choice of activation function.

The algorithm used to determine the parameters \mathbf{w} of the perceptron can most easily be motivated by error function minimization. A natural choice of error function would be the total number of misclassified patterns. However, this does not lead to a simple learning algorithm because the error is a piecewise constant function of \mathbf{w} , with discontinuities wherever a change in \mathbf{w} causes the decision boundary to move across one of the data points. Methods based on changing \mathbf{w} using the gradient of the error function cannot then be applied, because the gradient is zero almost everywhere.

We therefore consider an alternative error function known as the *perceptron criterion*. To derive this, we note that we are seeking a weight vector \mathbf{w} such that patterns \mathbf{x}_n in class C_1 will have $\mathbf{w}^T \phi(\mathbf{x}_n) > 0$, whereas patterns \mathbf{x}_n in class C_2 have $\mathbf{w}^T \phi(\mathbf{x}_n) < 0$. Using the $t \in \{-1, +1\}$ target coding scheme it follows that we would like all patterns to satisfy $\mathbf{w}^T \phi(\mathbf{x}_n) t_n > 0$. The perceptron criterion associates zero error with any pattern that is correctly classified, whereas for a misclassified pattern \mathbf{x}_n it tries to minimize the quantity $-\mathbf{w}^T \phi(\mathbf{x}_n) t_n$. The perceptron criterion is therefore given by

$$E_P(\mathbf{w}) = - \sum_{n \in \mathcal{M}} \mathbf{w}^T \phi_n t_n \quad (4.54)$$



Frank Rosenblatt

1928–1969

Rosenblatt's perceptron played an important role in the history of machine learning. Initially, Rosenblatt simulated the perceptron on an IBM 704 computer at Cornell in 1957, but by the early 1960s he had built special-purpose hardware that provided a direct, parallel implementation of perceptron learning. Many of his ideas were encapsulated in "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" published in 1962. Rosenblatt's work was criticized by Marvin Minsky, whose objections were published in the book "Perceptrons", co-authored with

Seymour Papert. This book was widely misinterpreted at the time as showing that neural networks were fatally flawed and could only learn solutions for linearly separable problems. In fact, it only proved such limitations in the case of single-layer networks such as the perceptron and merely conjectured (incorrectly) that they applied to more general network models. Unfortunately, however, this book contributed to the substantial decline in research funding for neural computing, a situation that was not reversed until the mid-1980s. Today, there are many hundreds, if not thousands, of applications of neural networks in widespread use, with examples in areas such as handwriting recognition and information retrieval being used routinely by millions of people.

where \mathcal{M} denotes the set of all misclassified patterns. The contribution to the error associated with a particular misclassified pattern is a linear function of \mathbf{w} in regions of \mathbf{w} space where the pattern is misclassified and zero in regions where it is correctly classified. The total error function is therefore piecewise linear.

Section 3.1.3

We now apply the stochastic gradient descent algorithm to this error function. The change in the weight vector \mathbf{w} is then given by

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta \phi_n t_n \quad (4.55)$$

where η is the learning rate parameter and τ is an integer that indexes the steps of the algorithm. Because the perceptron function $y(\mathbf{x}, \mathbf{w})$ is unchanged if we multiply \mathbf{w} by a constant, we can set the learning rate parameter η equal to 1 without loss of generality. Note that, as the weight vector evolves during training, the set of patterns that are misclassified will change.

The perceptron learning algorithm has a simple interpretation, as follows. We cycle through the training patterns in turn, and for each pattern \mathbf{x}_n we evaluate the perceptron function (4.52). If the pattern is correctly classified, then the weight vector remains unchanged, whereas if it is incorrectly classified, then for class C_1 we add the vector $\phi(\mathbf{x}_n)$ onto the current estimate of weight vector \mathbf{w} while for class C_2 we subtract the vector $\phi(\mathbf{x}_n)$ from \mathbf{w} . The perceptron learning algorithm is illustrated in Figure 4.7.

If we consider the effect of a single update in the perceptron learning algorithm, we see that the contribution to the error from a misclassified pattern will be reduced because from (4.55) we have

$$-\mathbf{w}^{(\tau+1)\top} \phi_n t_n = -\mathbf{w}^{(\tau)\top} \phi_n t_n - (\phi_n t_n)^\top \phi_n t_n < -\mathbf{w}^{(\tau)\top} \phi_n t_n \quad (4.56)$$

where we have set $\eta = 1$, and made use of $\|\phi_n t_n\|^2 > 0$. Of course, this does not imply that the contribution to the error function from the other misclassified patterns will have been reduced. Furthermore, the change in weight vector may have caused some previously correctly classified patterns to become misclassified. Thus the perceptron learning rule is not guaranteed to reduce the total error function at each stage.

However, the *perceptron convergence theorem* states that if there exists an exact solution (in other words, if the training data set is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps. Proofs of this theorem can be found for example in Rosenblatt (1962), Block (1962), Nilsson (1965), Minsky and Papert (1969), Hertz *et al.* (1991), and Bishop (1995a). Note, however, that the number of steps required to achieve convergence could still be substantial, and in practice, until convergence is achieved, we will not be able to distinguish between a nonseparable problem and one that is simply slow to converge.

Even when the data set is linearly separable, there may be many solutions, and which one is found will depend on the initialization of the parameters and on the order of presentation of the data points. Furthermore, for data sets that are not linearly separable, the perceptron learning algorithm will never converge.

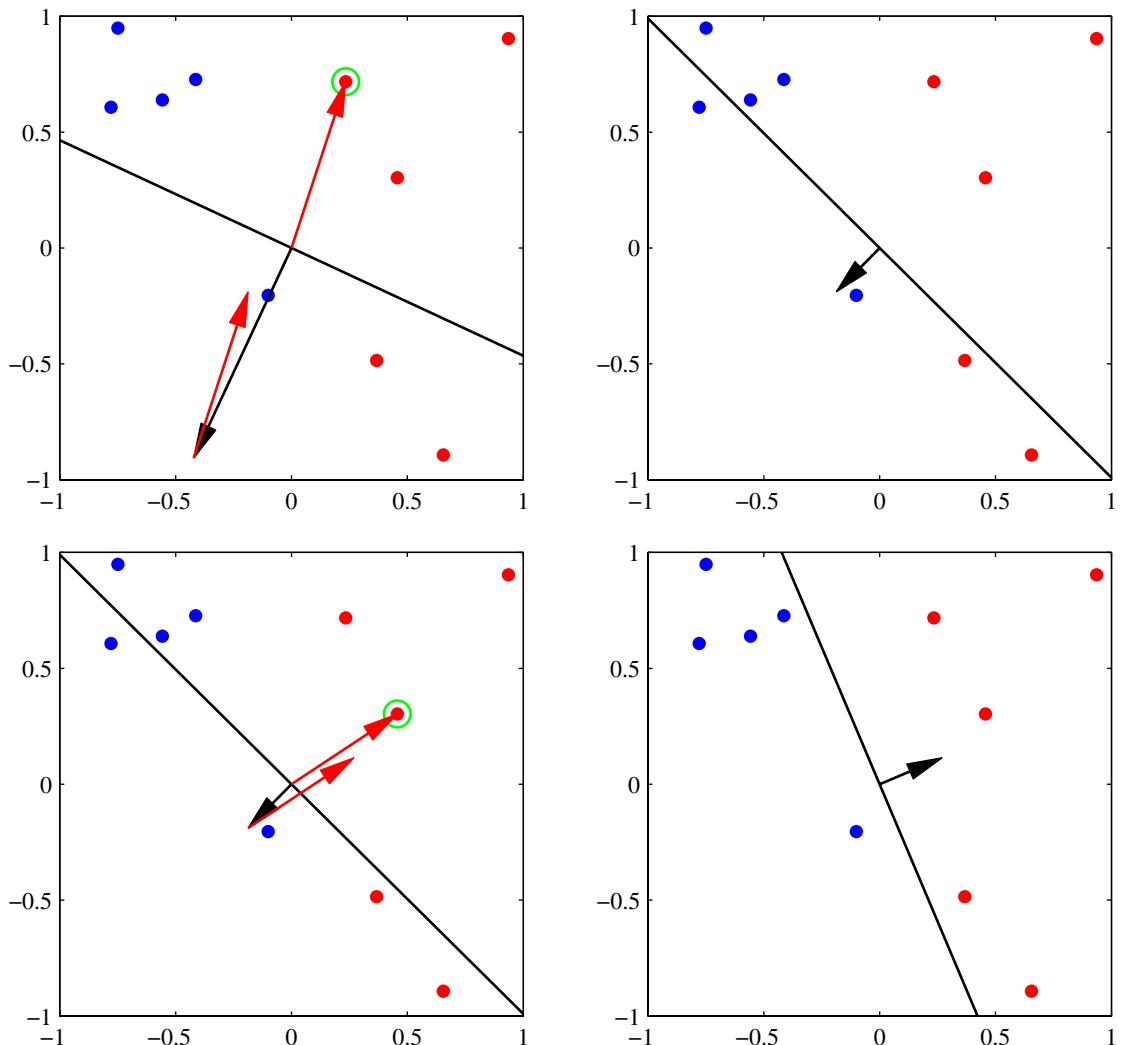


Figure 4.7 Illustration of the convergence of the perceptron learning algorithm, showing data points from two classes (red and blue) in a two-dimensional feature space (ϕ_1, ϕ_2) . The top left plot shows the initial parameter vector \mathbf{w} shown as a black arrow together with the corresponding decision boundary (black line), in which the arrow points towards the decision region which classified as belonging to the red class. The data point circled in green is misclassified and so its feature vector is added to the current weight vector, giving the new decision boundary shown in the top right plot. The bottom left plot shows the next misclassified point to be considered, indicated by the green circle, and its feature vector is again added to the weight vector giving the decision boundary shown in the bottom right plot for which all data points are correctly classified.



Figure 4.8 Illustration of the Mark 1 perceptron hardware. The photograph on the left shows how the inputs were obtained using a simple camera system in which an input scene, in this case a printed character, was illuminated by powerful lights, and an image focussed onto a 20×20 array of cadmium sulphide photocells, giving a primitive 400 pixel image. The perceptron also had a patch board, shown in the middle photograph, which allowed different configurations of input features to be tried. Often these were wired up at random to demonstrate the ability of the perceptron to learn without the need for precise wiring, in contrast to a modern digital computer. The photograph on the right shows one of the racks of adaptive weights. Each weight was implemented using a rotary variable resistor, also called a potentiometer, driven by an electric motor thereby allowing the value of the weight to be adjusted automatically by the learning algorithm.

Aside from difficulties with the learning algorithm, the perceptron does not provide probabilistic outputs, nor does it generalize readily to $K > 2$ classes. The most important limitation, however, arises from the fact that (in common with all of the models discussed in this chapter and the previous one) it is based on linear combinations of fixed basis functions. More detailed discussions of the limitations of perceptrons can be found in Minsky and Papert (1969) and Bishop (1995a).

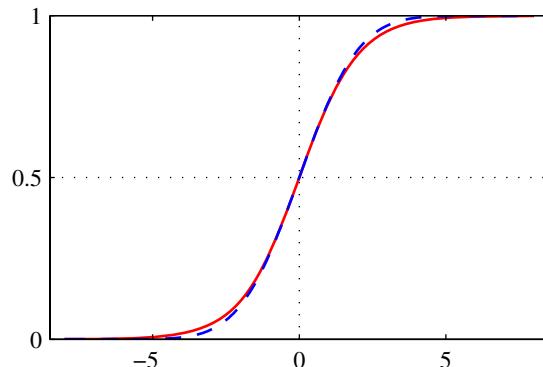
Analogue hardware implementations of the perceptron were built by Rosenblatt, based on motor-driven variable resistors to implement the adaptive parameters w_j . These are illustrated in Figure 4.8. The inputs were obtained from a simple camera system based on an array of photo-sensors, while the basis functions ϕ could be chosen in a variety of ways, for example based on simple fixed functions of randomly chosen subsets of pixels from the input image. Typical applications involved learning to discriminate simple shapes or characters.

At the same time that the perceptron was being developed, a closely related system called the *adaline*, which is short for ‘adaptive linear element’, was being explored by Widrow and co-workers. The functional form of the model was the same as for the perceptron, but a different approach to training was adopted (Widrow and Hoff, 1960; Widrow and Lehr, 1990).

4.2. Probabilistic Generative Models

We turn next to a probabilistic view of classification and show how models with linear decision boundaries arise from simple assumptions about the distribution of the data. In Section 1.5.4, we discussed the distinction between the discriminative and the generative approaches to classification. Here we shall adopt a generative

Figure 4.9 Plot of the logistic sigmoid function $\sigma(a)$ defined by (4.59), shown in red, together with the scaled probit function $\Phi(\lambda a)$, for $\lambda^2 = \pi/8$, shown in dashed blue, where $\Phi(a)$ is defined by (4.114). The scaling factor $\pi/8$ is chosen so that the derivatives of the two curves are equal for $a = 0$.



approach in which we model the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$, as well as the class priors $p(\mathcal{C}_k)$, and then use these to compute posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$ through Bayes' theorem.

Consider first of all the case of two classes. The posterior probability for class \mathcal{C}_1 can be written as

$$\begin{aligned} p(\mathcal{C}_1|\mathbf{x}) &= \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} \\ &= \frac{1}{1 + \exp(-a)} = \sigma(a) \end{aligned} \quad (4.57)$$

where we have defined

$$a = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} \quad (4.58)$$

and $\sigma(a)$ is the *logistic sigmoid* function defined by

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \quad (4.59)$$

which is plotted in Figure 4.9. The term ‘sigmoid’ means S-shaped. This type of function is sometimes also called a ‘squashing function’ because it maps the whole real axis into a finite interval. The logistic sigmoid has been encountered already in earlier chapters and plays an important role in many classification algorithms. It satisfies the following symmetry property

$$\sigma(-a) = 1 - \sigma(a) \quad (4.60)$$

as is easily verified. The inverse of the logistic sigmoid is given by

$$a = \ln \left(\frac{\sigma}{1 - \sigma} \right) \quad (4.61)$$

and is known as the *logit* function. It represents the log of the ratio of probabilities $\ln [p(\mathcal{C}_1|\mathbf{x})/p(\mathcal{C}_2|\mathbf{x})]$ for the two classes, also known as the *log odds*.

Note that in (4.57) we have simply rewritten the posterior probabilities in an equivalent form, and so the appearance of the logistic sigmoid may seem rather vacuous. However, it will have significance provided $a(\mathbf{x})$ takes a simple functional form. We shall shortly consider situations in which $a(\mathbf{x})$ is a linear function of \mathbf{x} , in which case the posterior probability is governed by a generalized linear model.

For the case of $K > 2$ classes, we have

$$\begin{aligned} p(\mathcal{C}_k|\mathbf{x}) &= \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)} \\ &= \frac{\exp(a_k)}{\sum_j \exp(a_j)} \end{aligned} \quad (4.62)$$

which is known as the *normalized exponential* and can be regarded as a multiclass generalization of the logistic sigmoid. Here the quantities a_k are defined by

$$a_k = \ln p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k). \quad (4.63)$$

The normalized exponential is also known as the *softmax function*, as it represents a smoothed version of the ‘max’ function because, if $a_k \gg a_j$ for all $j \neq k$, then $p(\mathcal{C}_k|\mathbf{x}) \simeq 1$, and $p(\mathcal{C}_j|\mathbf{x}) \simeq 0$.

We now investigate the consequences of choosing specific forms for the class-conditional densities, looking first at continuous input variables \mathbf{x} and then discussing briefly the case of discrete inputs.

4.2.1 Continuous inputs

Let us assume that the class-conditional densities are Gaussian and then explore the resulting form for the posterior probabilities. To start with, we shall assume that all classes share the same covariance matrix. Thus the density for class \mathcal{C}_k is given by

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right\}. \quad (4.64)$$

Consider first the case of two classes. From (4.57) and (4.58), we have

$$p(\mathcal{C}_1|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + w_0) \quad (4.65)$$

where we have defined

$$\mathbf{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \quad (4.66)$$

$$w_0 = -\frac{1}{2} \boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1 + \frac{1}{2} \boldsymbol{\mu}_2^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_2 + \ln \frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}. \quad (4.67)$$

We see that the quadratic terms in \mathbf{x} from the exponents of the Gaussian densities have cancelled (due to the assumption of common covariance matrices) leading to a linear function of \mathbf{x} in the argument of the logistic sigmoid. This result is illustrated for the case of a two-dimensional input space \mathbf{x} in Figure 4.10. The resulting

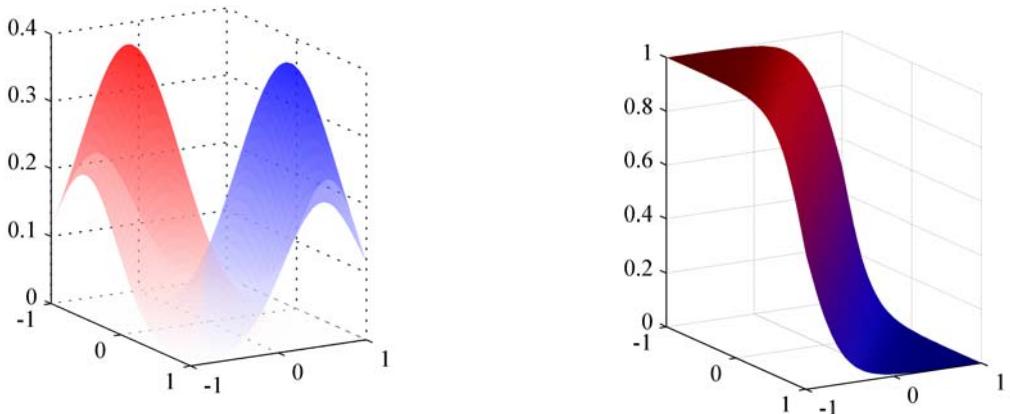


Figure 4.10 The left-hand plot shows the class-conditional densities for two classes, denoted red and blue. On the right is the corresponding posterior probability $p(\mathcal{C}_1|\mathbf{x})$, which is given by a logistic sigmoid of a linear function of \mathbf{x} . The surface in the right-hand plot is coloured using a proportion of red ink given by $p(\mathcal{C}_1|\mathbf{x})$ and a proportion of blue ink given by $p(\mathcal{C}_2|\mathbf{x}) = 1 - p(\mathcal{C}_1|\mathbf{x})$.

decision boundaries correspond to surfaces along which the posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$ are constant and so will be given by linear functions of \mathbf{x} , and therefore the decision boundaries are linear in input space. The prior probabilities $p(\mathcal{C}_k)$ enter only through the bias parameter w_0 so that changes in the priors have the effect of making parallel shifts of the decision boundary and more generally of the parallel contours of constant posterior probability.

For the general case of K classes we have, from (4.62) and (4.63),

$$a_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0} \quad (4.68)$$

where we have defined

$$\mathbf{w}_k = \Sigma^{-1} \boldsymbol{\mu}_k \quad (4.69)$$

$$w_{k0} = -\frac{1}{2} \boldsymbol{\mu}_k^T \Sigma^{-1} \boldsymbol{\mu}_k + \ln p(\mathcal{C}_k). \quad (4.70)$$

We see that the $a_k(\mathbf{x})$ are again linear functions of \mathbf{x} as a consequence of the cancellation of the quadratic terms due to the shared covariances. The resulting decision boundaries, corresponding to the minimum misclassification rate, will occur when two of the posterior probabilities (the two largest) are equal, and so will be defined by linear functions of \mathbf{x} , and so again we have a generalized linear model.

If we relax the assumption of a shared covariance matrix and allow each class-conditional density $p(\mathbf{x}|\mathcal{C}_k)$ to have its own covariance matrix Σ_k , then the earlier cancellations will no longer occur, and we will obtain quadratic functions of \mathbf{x} , giving rise to a *quadratic discriminant*. The linear and quadratic decision boundaries are illustrated in Figure 4.11.

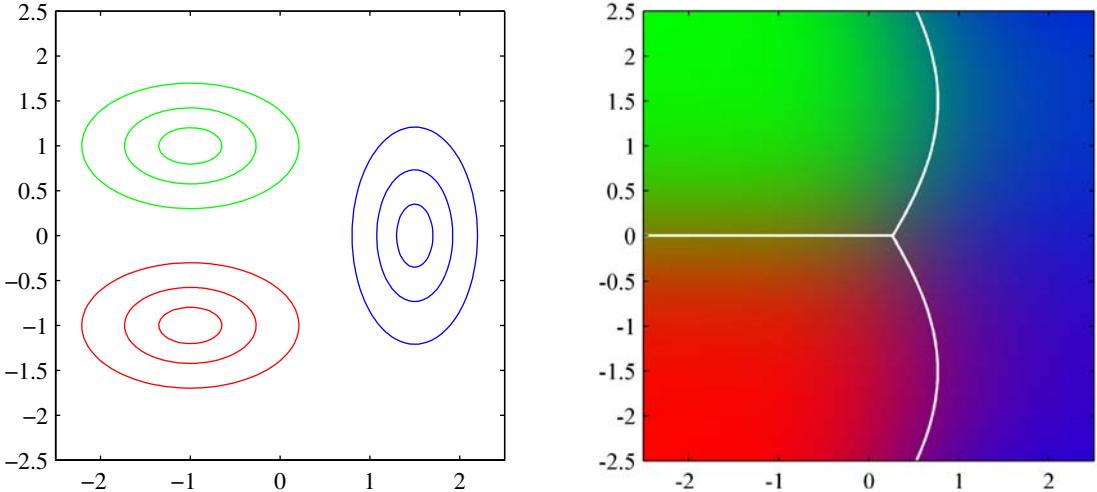


Figure 4.11 The left-hand plot shows the class-conditional densities for three classes each having a Gaussian distribution, coloured red, green, and blue, in which the red and green classes have the same covariance matrix. The right-hand plot shows the corresponding posterior probabilities, in which the RGB colour vector represents the posterior probabilities for the respective three classes. The decision boundaries are also shown. Notice that the boundary between the red and green classes, which have the same covariance matrix, is linear, whereas those between the other pairs of classes are quadratic.

4.2.2 Maximum likelihood solution

Once we have specified a parametric functional form for the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$, we can then determine the values of the parameters, together with the prior class probabilities $p(\mathcal{C}_k)$, using maximum likelihood. This requires a data set comprising observations of \mathbf{x} along with their corresponding class labels.

Consider first the case of two classes, each having a Gaussian class-conditional density with a shared covariance matrix, and suppose we have a data set $\{\mathbf{x}_n, t_n\}$ where $n = 1, \dots, N$. Here $t_n = 1$ denotes class \mathcal{C}_1 and $t_n = 0$ denotes class \mathcal{C}_2 . We denote the prior class probability $p(\mathcal{C}_1) = \pi$, so that $p(\mathcal{C}_2) = 1 - \pi$. For a data point \mathbf{x}_n from class \mathcal{C}_1 , we have $t_n = 1$ and hence

$$p(\mathbf{x}_n, \mathcal{C}_1) = p(\mathcal{C}_1)p(\mathbf{x}_n|\mathcal{C}_1) = \pi \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_1, \Sigma).$$

Similarly for class \mathcal{C}_2 , we have $t_n = 0$ and hence

$$p(\mathbf{x}_n, \mathcal{C}_2) = p(\mathcal{C}_2)p(\mathbf{x}_n|\mathcal{C}_2) = (1 - \pi) \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_2, \Sigma).$$

Thus the likelihood function is given by

$$p(\mathbf{t}|\pi, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma) = \prod_{n=1}^N [\pi \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_1, \Sigma)]^{t_n} [(1 - \pi) \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_2, \Sigma)]^{1-t_n} \quad (4.71)$$

where $\mathbf{t} = (t_1, \dots, t_N)^T$. As usual, it is convenient to maximize the log of the likelihood function. Consider first the maximization with respect to π . The terms in

the log likelihood function that depend on π are

$$\sum_{n=1}^N \{t_n \ln \pi + (1 - t_n) \ln(1 - \pi)\}. \quad (4.72)$$

Setting the derivative with respect to π equal to zero and rearranging, we obtain

$$\pi = \frac{1}{N} \sum_{n=1}^N t_n = \frac{N_1}{N} = \frac{N_1}{N_1 + N_2} \quad (4.73)$$

where N_1 denotes the total number of data points in class \mathcal{C}_1 , and N_2 denotes the total number of data points in class \mathcal{C}_2 . Thus the maximum likelihood estimate for π is simply the fraction of points in class \mathcal{C}_1 as expected. This result is easily generalized to the multiclass case where again the maximum likelihood estimate of the prior probability associated with class \mathcal{C}_k is given by the fraction of the training set points assigned to that class.

Exercise 4.9 Now consider the maximization with respect to $\boldsymbol{\mu}_1$. Again we can pick out of the log likelihood function those terms that depend on $\boldsymbol{\mu}_1$ giving

$$\sum_{n=1}^N t_n \ln \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}) = -\frac{1}{2} \sum_{n=1}^N t_n (\mathbf{x}_n - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_1) + \text{const.} \quad (4.74)$$

Setting the derivative with respect to $\boldsymbol{\mu}_1$ to zero and rearranging, we obtain

$$\boldsymbol{\mu}_1 = \frac{1}{N_1} \sum_{n=1}^N t_n \mathbf{x}_n \quad (4.75)$$

which is simply the mean of all the input vectors \mathbf{x}_n assigned to class \mathcal{C}_1 . By a similar argument, the corresponding result for $\boldsymbol{\mu}_2$ is given by

$$\boldsymbol{\mu}_2 = \frac{1}{N_2} \sum_{n=1}^N (1 - t_n) \mathbf{x}_n \quad (4.76)$$

which again is the mean of all the input vectors \mathbf{x}_n assigned to class \mathcal{C}_2 .

Finally, consider the maximum likelihood solution for the shared covariance matrix $\boldsymbol{\Sigma}$. Picking out the terms in the log likelihood function that depend on $\boldsymbol{\Sigma}$, we have

$$\begin{aligned} & -\frac{1}{2} \sum_{n=1}^N t_n \ln |\boldsymbol{\Sigma}| - \frac{1}{2} \sum_{n=1}^N t_n (\mathbf{x}_n - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_1) \\ & - \frac{1}{2} \sum_{n=1}^N (1 - t_n) \ln |\boldsymbol{\Sigma}| - \frac{1}{2} \sum_{n=1}^N (1 - t_n) (\mathbf{x}_n - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_2) \\ & = -\frac{N}{2} \ln |\boldsymbol{\Sigma}| - \frac{N}{2} \text{Tr} \{ \boldsymbol{\Sigma}^{-1} \mathbf{S} \} \end{aligned} \quad (4.77)$$

where we have defined

$$\mathbf{S} = \frac{N_1}{N} \mathbf{S}_1 + \frac{N_2}{N} \mathbf{S}_2 \quad (4.78)$$

$$\mathbf{S}_1 = \frac{1}{N_1} \sum_{n \in \mathcal{C}_1} (\mathbf{x}_n - \boldsymbol{\mu}_1)(\mathbf{x}_n - \boldsymbol{\mu}_1)^T \quad (4.79)$$

$$\mathbf{S}_2 = \frac{1}{N_2} \sum_{n \in \mathcal{C}_2} (\mathbf{x}_n - \boldsymbol{\mu}_2)(\mathbf{x}_n - \boldsymbol{\mu}_2)^T. \quad (4.80)$$

Using the standard result for the maximum likelihood solution for a Gaussian distribution, we see that $\Sigma = \mathbf{S}$, which represents a weighted average of the covariance matrices associated with each of the two classes separately.

This result is easily extended to the K class problem to obtain the corresponding maximum likelihood solutions for the parameters in which each class-conditional density is Gaussian with a shared covariance matrix. Note that the approach of fitting Gaussian distributions to the classes is not robust to outliers, because the maximum likelihood estimation of a Gaussian is not robust.

Exercise 4.10

Section 2.3.7

Section 8.2.2

Exercise 4.11

4.2.3 Discrete features

Let us now consider the case of discrete feature values x_i . For simplicity, we begin by looking at binary feature values $x_i \in \{0, 1\}$ and discuss the extension to more general discrete features shortly. If there are D inputs, then a general distribution would correspond to a table of 2^D numbers for each class, containing $2^D - 1$ independent variables (due to the summation constraint). Because this grows exponentially with the number of features, we might seek a more restricted representation. Here we will make the *naive Bayes* assumption in which the feature values are treated as independent, conditioned on the class \mathcal{C}_k . Thus we have class-conditional distributions of the form

$$p(\mathbf{x}|\mathcal{C}_k) = \prod_{i=1}^D \mu_{ki}^{x_i} (1 - \mu_{ki})^{1-x_i} \quad (4.81)$$

which contain D independent parameters for each class. Substituting into (4.63) then gives

$$a_k(\mathbf{x}) = \sum_{i=1}^D \{x_i \ln \mu_{ki} + (1 - x_i) \ln(1 - \mu_{ki})\} + \ln p(\mathcal{C}_k) \quad (4.82)$$

which again are linear functions of the input values x_i . For the case of $K = 2$ classes, we can alternatively consider the logistic sigmoid formulation given by (4.57). Analogous results are obtained for discrete variables each of which can take $M > 2$ states.

4.2.4 Exponential family

As we have seen, for both Gaussian distributed and discrete inputs, the posterior class probabilities are given by generalized linear models with logistic sigmoid ($K = 2$)

2 classes) or softmax ($K \geq 2$ classes) activation functions. These are particular cases of a more general result obtained by assuming that the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ are members of the exponential family of distributions.

Using the form (2.194) for members of the exponential family, we see that the distribution of \mathbf{x} can be written in the form

$$p(\mathbf{x}|\boldsymbol{\lambda}_k) = h(\mathbf{x})g(\boldsymbol{\lambda}_k) \exp \left\{ \boldsymbol{\lambda}_k^T \mathbf{u}(\mathbf{x}) \right\}. \quad (4.83)$$

We now restrict attention to the subclass of such distributions for which $\mathbf{u}(\mathbf{x}) = \mathbf{x}$. Then we make use of (2.236) to introduce a scaling parameter s , so that we obtain the restricted set of exponential family class-conditional densities of the form

$$p(\mathbf{x}|\boldsymbol{\lambda}_k, s) = \frac{1}{s} h\left(\frac{1}{s}\mathbf{x}\right) g(\boldsymbol{\lambda}_k) \exp \left\{ \frac{1}{s} \boldsymbol{\lambda}_k^T \mathbf{x} \right\}. \quad (4.84)$$

Note that we are allowing each class to have its own parameter vector $\boldsymbol{\lambda}_k$ but we are assuming that the classes share the same scale parameter s .

For the two-class problem, we substitute this expression for the class-conditional densities into (4.58) and we see that the posterior class probability is again given by a logistic sigmoid acting on a linear function $a(\mathbf{x})$ which is given by

$$a(\mathbf{x}) = (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_2)^T \mathbf{x} + \ln g(\boldsymbol{\lambda}_1) - \ln g(\boldsymbol{\lambda}_2) + \ln p(\mathcal{C}_1) - \ln p(\mathcal{C}_2). \quad (4.85)$$

Similarly, for the K -class problem, we substitute the class-conditional density expression into (4.63) to give

$$a_k(\mathbf{x}) = \boldsymbol{\lambda}_k^T \mathbf{x} + \ln g(\boldsymbol{\lambda}_k) + \ln p(\mathcal{C}_k) \quad (4.86)$$

and so again is a linear function of \mathbf{x} .

4.3. Probabilistic Discriminative Models

For the two-class classification problem, we have seen that the posterior probability of class \mathcal{C}_1 can be written as a logistic sigmoid acting on a linear function of \mathbf{x} , for a wide choice of class-conditional distributions $p(\mathbf{x}|\mathcal{C}_k)$. Similarly, for the multiclass case, the posterior probability of class \mathcal{C}_k is given by a softmax transformation of a linear function of \mathbf{x} . For specific choices of the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$, we have used maximum likelihood to determine the parameters of the densities as well as the class priors $p(\mathcal{C}_k)$ and then used Bayes' theorem to find the posterior class probabilities.

However, an alternative approach is to use the functional form of the generalized linear model explicitly and to determine its parameters directly by using maximum likelihood. We shall see that there is an efficient algorithm finding such solutions known as *iterative reweighted least squares*, or *IRLS*.

The indirect approach to finding the parameters of a generalized linear model, by fitting class-conditional densities and class priors separately and then applying

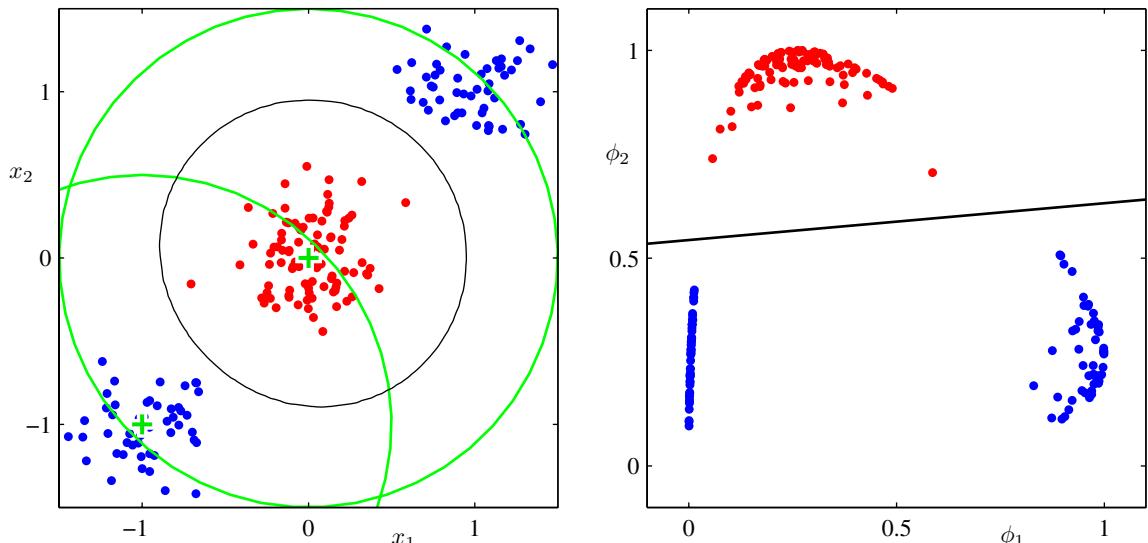


Figure 4.12 Illustration of the role of nonlinear basis functions in linear classification models. The left plot shows the original input space (x_1, x_2) together with data points from two classes labelled red and blue. Two ‘Gaussian’ basis functions $\phi_1(\mathbf{x})$ and $\phi_2(\mathbf{x})$ are defined in this space with centres shown by the green crosses and with contours shown by the green circles. The right-hand plot shows the corresponding feature space (ϕ_1, ϕ_2) together with the linear decision boundary obtained given by a logistic regression model of the form discussed in Section 4.3.2. This corresponds to a nonlinear decision boundary in the original input space, shown by the black curve in the left-hand plot.

Bayes’ theorem, represents an example of *generative* modelling, because we could take such a model and generate synthetic data by drawing values of \mathbf{x} from the marginal distribution $p(\mathbf{x})$. In the direct approach, we are maximizing a likelihood function defined through the conditional distribution $p(C_k|\mathbf{x})$, which represents a form of *discriminative* training. One advantage of the discriminative approach is that there will typically be fewer adaptive parameters to be determined, as we shall see shortly. It may also lead to improved predictive performance, particularly when the class-conditional density assumptions give a poor approximation to the true distributions.

4.3.1 Fixed basis functions

So far in this chapter, we have considered classification models that work directly with the original input vector \mathbf{x} . However, all of the algorithms are equally applicable if we first make a fixed nonlinear transformation of the inputs using a vector of basis functions $\phi(\mathbf{x})$. The resulting decision boundaries will be linear in the feature space ϕ , and these correspond to nonlinear decision boundaries in the original \mathbf{x} space, as illustrated in Figure 4.12. Classes that are linearly separable in the feature space $\phi(\mathbf{x})$ need not be linearly separable in the original observation space \mathbf{x} . Note that as in our discussion of linear models for regression, one of the

basis functions is typically set to a constant, say $\phi_0(\mathbf{x}) = 1$, so that the corresponding parameter w_0 plays the role of a bias. For the remainder of this chapter, we shall include a fixed basis function transformation $\phi(\mathbf{x})$, as this will highlight some useful similarities to the regression models discussed in Chapter 3.

For many problems of practical interest, there is significant overlap between the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$. This corresponds to posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$, which, for at least some values of \mathbf{x} , are not 0 or 1. In such cases, the optimal solution is obtained by modelling the posterior probabilities accurately and then applying standard decision theory, as discussed in Chapter 1. Note that nonlinear transformations $\phi(\mathbf{x})$ cannot remove such class overlap. Indeed, they can increase the level of overlap, or create overlap where none existed in the original observation space. However, suitable choices of nonlinearity can make the process of modelling the posterior probabilities easier.

Such fixed basis function models have important limitations, and these will be resolved in later chapters by allowing the basis functions themselves to adapt to the data. Notwithstanding these limitations, models with fixed nonlinear basis functions play an important role in applications, and a discussion of such models will introduce many of the key concepts needed for an understanding of their more complex counterparts.

4.3.2 Logistic regression

We begin our treatment of generalized linear models by considering the problem of two-class classification. In our discussion of generative approaches in Section 4.2, we saw that under rather general assumptions, the posterior probability of class \mathcal{C}_1 can be written as a logistic sigmoid acting on a linear function of the feature vector ϕ so that

$$p(\mathcal{C}_1|\phi) = y(\phi) = \sigma(\mathbf{w}^T \phi) \quad (4.87)$$

with $p(\mathcal{C}_2|\phi) = 1 - p(\mathcal{C}_1|\phi)$. Here $\sigma(\cdot)$ is the *logistic sigmoid* function defined by (4.59). In the terminology of statistics, this model is known as *logistic regression*, although it should be emphasized that this is a model for classification rather than regression.

For an M -dimensional feature space ϕ , this model has M adjustable parameters. By contrast, if we had fitted Gaussian class conditional densities using maximum likelihood, we would have used $2M$ parameters for the means and $M(M + 1)/2$ parameters for the (shared) covariance matrix. Together with the class prior $p(\mathcal{C}_1)$, this gives a total of $M(M + 5)/2 + 1$ parameters, which grows quadratically with M , in contrast to the linear dependence on M of the number of parameters in logistic regression. For large values of M , there is a clear advantage in working with the logistic regression model directly.

We now use maximum likelihood to determine the parameters of the logistic regression model. To do this, we shall make use of the derivative of the logistic sigmoid function, which can conveniently be expressed in terms of the sigmoid function itself

$$\frac{d\sigma}{da} = \sigma(1 - \sigma). \quad (4.88)$$

Section 3.6

Exercise 4.12

For a data set $\{\phi_n, t_n\}$, where $t_n \in \{0, 1\}$ and $\phi_n = \phi(\mathbf{x}_n)$, with $n = 1, \dots, N$, the likelihood function can be written

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^N y_n^{t_n} \{1 - y_n\}^{1-t_n} \quad (4.89)$$

where $\mathbf{t} = (t_1, \dots, t_N)^T$ and $y_n = p(\mathcal{C}_1|\phi_n)$. As usual, we can define an error function by taking the negative logarithm of the likelihood, which gives the *cross-entropy* error function in the form

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \quad (4.90)$$

where $y_n = \sigma(a_n)$ and $a_n = \mathbf{w}^T \phi_n$. Taking the gradient of the error function with respect to \mathbf{w} , we obtain

$$\nabla E(\mathbf{w}) = \sum_{n=1}^N (y_n - t_n) \phi_n \quad (4.91)$$

where we have made use of (4.88). We see that the factor involving the derivative of the logistic sigmoid has cancelled, leading to a simplified form for the gradient of the log likelihood. In particular, the contribution to the gradient from data point n is given by the ‘error’ $y_n - t_n$ between the target value and the prediction of the model, times the basis function vector ϕ_n . Furthermore, comparison with (3.13) shows that this takes precisely the same form as the gradient of the sum-of-squares error function for the linear regression model.

If desired, we could make use of the result (4.91) to give a sequential algorithm in which patterns are presented one at a time, in which each of the weight vectors is updated using (3.22) in which ∇E_n is the n^{th} term in (4.91).

It is worth noting that maximum likelihood can exhibit severe over-fitting for data sets that are linearly separable. This arises because the maximum likelihood solution occurs when the hyperplane corresponding to $\sigma = 0.5$, equivalent to $\mathbf{w}^T \phi = 0$, separates the two classes and the magnitude of \mathbf{w} goes to infinity. In this case, the logistic sigmoid function becomes infinitely steep in feature space, corresponding to a Heaviside step function, so that every training point from each class k is assigned a posterior probability $p(\mathcal{C}_k|\mathbf{x}) = 1$. Furthermore, there is typically a continuum of such solutions because any separating hyperplane will give rise to the same posterior probabilities at the training data points, as will be seen later in Figure 10.13. Maximum likelihood provides no way to favour one such solution over another, and which solution is found in practice will depend on the choice of optimization algorithm and on the parameter initialization. Note that the problem will arise even if the number of data points is large compared with the number of parameters in the model, so long as the training data set is linearly separable. The singularity can be avoided by inclusion of a prior and finding a MAP solution for \mathbf{w} , or equivalently by adding a regularization term to the error function.

Exercise 4.13

Section 3.1.1

Exercise 4.14

4.3.3 Iterative reweighted least squares

In the case of the linear regression models discussed in Chapter 3, the maximum likelihood solution, on the assumption of a Gaussian noise model, leads to a closed-form solution. This was a consequence of the quadratic dependence of the log likelihood function on the parameter vector \mathbf{w} . For logistic regression, there is no longer a closed-form solution, due to the nonlinearity of the logistic sigmoid function. However, the departure from a quadratic form is not substantial. To be precise, the error function is concave, as we shall see shortly, and hence has a unique minimum. Furthermore, the error function can be minimized by an efficient iterative technique based on the *Newton-Raphson* iterative optimization scheme, which uses a local quadratic approximation to the log likelihood function. The Newton-Raphson update, for minimizing a function $E(\mathbf{w})$, takes the form (Fletcher, 1987; Bishop and Nabney, 2008)

$$\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \mathbf{H}^{-1} \nabla E(\mathbf{w}). \quad (4.92)$$

where \mathbf{H} is the Hessian matrix whose elements comprise the second derivatives of $E(\mathbf{w})$ with respect to the components of \mathbf{w} .

Let us first of all apply the Newton-Raphson method to the linear regression model (3.3) with the sum-of-squares error function (3.12). The gradient and Hessian of this error function are given by

$$\nabla E(\mathbf{w}) = \sum_{n=1}^N (\mathbf{w}^T \boldsymbol{\phi}_n - t_n) \boldsymbol{\phi}_n = \boldsymbol{\Phi}^T \boldsymbol{\Phi} \mathbf{w} - \boldsymbol{\Phi}^T \mathbf{t} \quad (4.93)$$

$$\mathbf{H} = \nabla \nabla E(\mathbf{w}) = \sum_{n=1}^N \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T = \boldsymbol{\Phi}^T \boldsymbol{\Phi} \quad (4.94)$$

Section 3.1.1

where $\boldsymbol{\Phi}$ is the $N \times M$ design matrix, whose n^{th} row is given by $\boldsymbol{\phi}_n^T$. The Newton-Raphson update then takes the form

$$\begin{aligned} \mathbf{w}^{(\text{new})} &= \mathbf{w}^{(\text{old})} - (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \{ \boldsymbol{\Phi}^T \boldsymbol{\Phi} \mathbf{w}^{(\text{old})} - \boldsymbol{\Phi}^T \mathbf{t} \} \\ &= (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \mathbf{t} \end{aligned} \quad (4.95)$$

which we recognize as the standard least-squares solution. Note that the error function in this case is quadratic and hence the Newton-Raphson formula gives the exact solution in one step.

Now let us apply the Newton-Raphson update to the cross-entropy error function (4.90) for the logistic regression model. From (4.91) we see that the gradient and Hessian of this error function are given by

$$\nabla E(\mathbf{w}) = \sum_{n=1}^N (y_n - t_n) \boldsymbol{\phi}_n = \boldsymbol{\Phi}^T (\mathbf{y} - \mathbf{t}) \quad (4.96)$$

$$\mathbf{H} = \nabla \nabla E(\mathbf{w}) = \sum_{n=1}^N y_n(1 - y_n) \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T = \boldsymbol{\Phi}^T \mathbf{R} \boldsymbol{\Phi} \quad (4.97)$$

where we have made use of (4.88). Also, we have introduced the $N \times N$ diagonal matrix \mathbf{R} with elements

$$R_{nn} = y_n(1 - y_n). \quad (4.98)$$

We see that the Hessian is no longer constant but depends on \mathbf{w} through the weighting matrix \mathbf{R} , corresponding to the fact that the error function is no longer quadratic. Using the property $0 < y_n < 1$, which follows from the form of the logistic sigmoid function, we see that $\mathbf{u}^T \mathbf{H} \mathbf{u} > 0$ for an arbitrary vector \mathbf{u} , and so the Hessian matrix \mathbf{H} is positive definite. It follows that the error function is a concave function of \mathbf{w} and hence has a unique minimum.

Exercise 4.15

The Newton-Raphson update formula for the logistic regression model then becomes

$$\begin{aligned} \mathbf{w}^{(\text{new})} &= \mathbf{w}^{(\text{old})} - (\Phi^T \mathbf{R} \Phi)^{-1} \Phi^T (\mathbf{y} - \mathbf{t}) \\ &= (\Phi^T \mathbf{R} \Phi)^{-1} \{ \Phi^T \mathbf{R} \Phi \mathbf{w}^{(\text{old})} - \Phi^T (\mathbf{y} - \mathbf{t}) \} \\ &= (\Phi^T \mathbf{R} \Phi)^{-1} \Phi^T \mathbf{R} \mathbf{z} \end{aligned} \quad (4.99)$$

where \mathbf{z} is an N -dimensional vector with elements

$$\mathbf{z} = \Phi \mathbf{w}^{(\text{old})} - \mathbf{R}^{-1} (\mathbf{y} - \mathbf{t}). \quad (4.100)$$

We see that the update formula (4.99) takes the form of a set of normal equations for a weighted least-squares problem. Because the weighing matrix \mathbf{R} is not constant but depends on the parameter vector \mathbf{w} , we must apply the normal equations iteratively, each time using the new weight vector \mathbf{w} to compute a revised weighing matrix \mathbf{R} . For this reason, the algorithm is known as *iterative reweighted least squares*, or *IRLS* (Rubin, 1983). As in the weighted least-squares problem, the elements of the diagonal weighting matrix \mathbf{R} can be interpreted as variances because the mean and variance of t in the logistic regression model are given by

$$\mathbb{E}[t] = \sigma(\mathbf{x}) = y \quad (4.101)$$

$$\text{var}[t] = \mathbb{E}[t^2] - \mathbb{E}[t]^2 = \sigma(\mathbf{x}) - \sigma(\mathbf{x})^2 = y(1 - y) \quad (4.102)$$

where we have used the property $t^2 = t$ for $t \in \{0, 1\}$. In fact, we can interpret IRLS as the solution to a linearized problem in the space of the variable $a = \mathbf{w}^T \phi$. The quantity z_n , which corresponds to the n^{th} element of \mathbf{z} , can then be given a simple interpretation as an effective target value in this space obtained by making a local linear approximation to the logistic sigmoid function around the current operating point $\mathbf{w}^{(\text{old})}$

$$\begin{aligned} a_n(\mathbf{w}) &\simeq a_n(\mathbf{w}^{(\text{old})}) + \left. \frac{da_n}{dy_n} \right|_{\mathbf{w}^{(\text{old})}} (t_n - y_n) \\ &= \phi_n^T \mathbf{w}^{(\text{old})} - \frac{(y_n - t_n)}{y_n(1 - y_n)} = z_n. \end{aligned} \quad (4.103)$$

4.3.4 Multiclass logistic regression

Section 4.2

In our discussion of generative models for multiclass classification, we have seen that for a large class of distributions, the posterior probabilities are given by a softmax transformation of linear functions of the feature variables, so that

$$p(\mathcal{C}_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \quad (4.104)$$

where the ‘activations’ a_k are given by

$$a_k = \mathbf{w}_k^T \phi. \quad (4.105)$$

There we used maximum likelihood to determine separately the class-conditional densities and the class priors and then found the corresponding posterior probabilities using Bayes’ theorem, thereby implicitly determining the parameters $\{\mathbf{w}_k\}$. Here we consider the use of maximum likelihood to determine the parameters $\{\mathbf{w}_k\}$ of this model directly. To do this, we will require the derivatives of y_k with respect to all of the activations a_j . These are given by

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j) \quad (4.106)$$

where I_{kj} are the elements of the identity matrix.

Next we write down the likelihood function. This is most easily done using the 1-of- K coding scheme in which the target vector \mathbf{t}_n for a feature vector ϕ_n belonging to class \mathcal{C}_k is a binary vector with all elements zero except for element k , which equals one. The likelihood function is then given by

$$p(\mathbf{T}|\mathbf{w}_1, \dots, \mathbf{w}_K) = \prod_{n=1}^N \prod_{k=1}^K p(\mathcal{C}_k|\phi_n)^{t_{nk}} = \prod_{n=1}^N \prod_{k=1}^K y_{nk}^{t_{nk}} \quad (4.107)$$

where $y_{nk} = y_k(\phi_n)$, and \mathbf{T} is an $N \times K$ matrix of target variables with elements t_{nk} . Taking the negative logarithm then gives

$$E(\mathbf{w}_1, \dots, \mathbf{w}_K) = -\ln p(\mathbf{T}|\mathbf{w}_1, \dots, \mathbf{w}_K) = -\sum_{n=1}^N \sum_{k=1}^K t_{nk} \ln y_{nk} \quad (4.108)$$

which is known as the *cross-entropy* error function for the multiclass classification problem.

We now take the gradient of the error function with respect to one of the parameter vectors \mathbf{w}_j . Making use of the result (4.106) for the derivatives of the softmax function, we obtain

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \dots, \mathbf{w}_K) = \sum_{n=1}^N (y_{nj} - t_{nj}) \phi_n \quad (4.109)$$

Exercise 4.17

where we have made use of $\sum_k t_{nk} = 1$. Once again, we see the same form arising for the gradient as was found for the sum-of-squares error function with the linear model and the cross-entropy error for the logistic regression model, namely the product of the error $(y_{nj} - t_{nj})$ times the basis function ϕ_n . Again, we could use this to formulate a sequential algorithm in which patterns are presented one at a time, in which each of the weight vectors is updated using (3.22).

We have seen that the derivative of the log likelihood function for a linear regression model with respect to the parameter vector w for a data point n took the form of the ‘error’ $y_n - t_n$ times the feature vector ϕ_n . Similarly, for the combination of logistic sigmoid activation function and cross-entropy error function (4.90), and for the softmax activation function with the multiclass cross-entropy error function (4.108), we again obtain this same simple form. This is an example of a more general result, as we shall see in Section 4.3.6.

To find a batch algorithm, we again appeal to the Newton-Raphson update to obtain the corresponding IRLS algorithm for the multiclass problem. This requires evaluation of the Hessian matrix that comprises blocks of size $M \times M$ in which block j, k is given by

$$\nabla_{w_k} \nabla_{w_j} E(w_1, \dots, w_K) = - \sum_{n=1}^N y_{nk} (I_{kj} - y_{nj}) \phi_n \phi_n^T. \quad (4.110)$$

As with the two-class problem, the Hessian matrix for the multiclass logistic regression model is positive definite and so the error function again has a unique minimum. Practical details of IRLS for the multiclass case can be found in Bishop and Nabney (2008).

4.3.5 Probit regression

We have seen that, for a broad range of class-conditional distributions, described by the exponential family, the resulting posterior class probabilities are given by a logistic (or softmax) transformation acting on a linear function of the feature variables. However, not all choices of class-conditional density give rise to such a simple form for the posterior probabilities (for instance, if the class-conditional densities are modelled using Gaussian mixtures). This suggests that it might be worth exploring other types of discriminative probabilistic model. For the purposes of this chapter, however, we shall return to the two-class case, and again remain within the framework of generalized linear models so that

$$p(t = 1|a) = f(a) \quad (4.111)$$

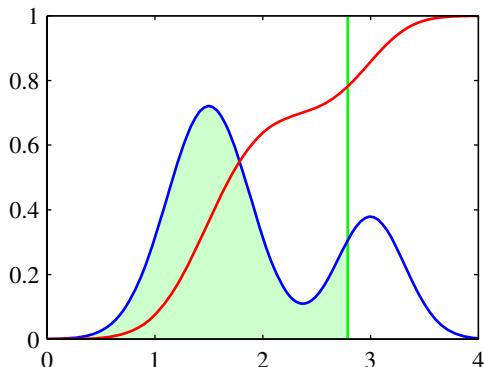
where $a = w^T \phi$, and $f(\cdot)$ is the activation function.

One way to motivate an alternative choice for the link function is to consider a noisy threshold model, as follows. For each input ϕ_n , we evaluate $a_n = w^T \phi_n$ and then we set the target value according to

$$\begin{cases} t_n = 1 & \text{if } a_n \geq \theta \\ t_n = 0 & \text{otherwise.} \end{cases} \quad (4.112)$$

Exercise 4.20

Figure 4.13 Schematic example of a probability density $p(\theta)$ shown by the blue curve, given in this example by a mixture of two Gaussians, along with its cumulative distribution function $f(a)$, shown by the red curve. Note that the value of the blue curve at any point, such as that indicated by the vertical green line, corresponds to the slope of the red curve at the same point. Conversely, the value of the red curve at this point corresponds to the area under the blue curve indicated by the shaded green region. In the stochastic threshold model, the class label takes the value $t = 1$ if the value of $a = \mathbf{w}^T \phi$ exceeds a threshold, otherwise it takes the value $t = 0$. This is equivalent to an activation function given by the cumulative distribution function $f(a)$.



If the value of θ is drawn from a probability density $p(\theta)$, then the corresponding activation function will be given by the cumulative distribution function

$$f(a) = \int_{-\infty}^a p(\theta) d\theta \quad (4.113)$$

as illustrated in Figure 4.13.

As a specific example, suppose that the density $p(\theta)$ is given by a zero mean, unit variance Gaussian. The corresponding cumulative distribution function is given by

$$\Phi(a) = \int_{-\infty}^a \mathcal{N}(\theta|0, 1) d\theta \quad (4.114)$$

which is known as the *probit* function. It has a sigmoidal shape and is compared with the logistic sigmoid function in Figure 4.9. Note that the use of a more general Gaussian distribution does not change the model because this is equivalent to a re-scaling of the linear coefficients \mathbf{w} . Many numerical packages provide for the evaluation of a closely related function defined by

$$\text{erf}(a) = \frac{2}{\sqrt{\pi}} \int_0^a \exp(-\theta^2/2) d\theta \quad (4.115)$$

and known as the *erf function* or *error function* (not to be confused with the error function of a machine learning model). It is related to the probit function by

$$\Phi(a) = \frac{1}{2} \left\{ 1 + \frac{1}{\sqrt{2}} \text{erf}(a) \right\}. \quad (4.116)$$

The generalized linear model based on a probit activation function is known as *probit regression*.

We can determine the parameters of this model using maximum likelihood, by a straightforward extension of the ideas discussed earlier. In practice, the results found using probit regression tend to be similar to those of logistic regression. We shall,

Exercise 4.21

however, find another use for the probit model when we discuss Bayesian treatments of logistic regression in Section 4.5.

One issue that can occur in practical applications is that of *outliers*, which can arise for instance through errors in measuring the input vector \mathbf{x} or through mislabelling of the target value t . Because such points can lie a long way to the wrong side of the ideal decision boundary, they can seriously distort the classifier. Note that the logistic and probit regression models behave differently in this respect because the tails of the logistic sigmoid decay asymptotically like $\exp(-x)$ for $x \rightarrow \infty$, whereas for the probit activation function they decay like $\exp(-x^2)$, and so the probit model can be significantly more sensitive to outliers.

However, both the logistic and the probit models assume the data is correctly labelled. The effect of mislabelling is easily incorporated into a probabilistic model by introducing a probability ϵ that the target value t has been flipped to the wrong value (Opper and Winther, 2000a), leading to a target value distribution for data point \mathbf{x} of the form

$$\begin{aligned} p(t|\mathbf{x}) &= (1 - \epsilon)\sigma(\mathbf{x}) + \epsilon(1 - \sigma(\mathbf{x})) \\ &= \epsilon + (1 - 2\epsilon)\sigma(\mathbf{x}) \end{aligned} \quad (4.117)$$

where $\sigma(\mathbf{x})$ is the activation function with input vector \mathbf{x} . Here ϵ may be set in advance, or it may be treated as a hyperparameter whose value is inferred from the data.

4.3.6 Canonical link functions

For the linear regression model with a Gaussian noise distribution, the error function, corresponding to the negative log likelihood, is given by (3.12). If we take the derivative with respect to the parameter vector \mathbf{w} of the contribution to the error function from a data point n , this takes the form of the ‘error’ $y_n - t_n$ times the feature vector ϕ_n , where $y_n = \mathbf{w}^T \phi_n$. Similarly, for the combination of the logistic sigmoid activation function and the cross-entropy error function (4.90), and for the softmax activation function with the multiclass cross-entropy error function (4.108), we again obtain this same simple form. We now show that this is a general result of assuming a conditional distribution for the target variable from the exponential family, along with a corresponding choice for the activation function known as the *canonical link function*.

We again make use of the restricted form (4.84) of exponential family distributions. Note that here we are applying the assumption of exponential family distribution to the target variable t , in contrast to Section 4.2.4 where we applied it to the input vector \mathbf{x} . We therefore consider conditional distributions of the target variable of the form

$$p(t|\eta, s) = \frac{1}{s} h\left(\frac{t}{s}\right) g(\eta) \exp\left\{\frac{\eta t}{s}\right\}. \quad (4.118)$$

Using the same line of argument as led to the derivation of the result (2.226), we see that the conditional mean of t , which we denote by y , is given by

$$y \equiv \mathbb{E}[t|\eta] = -s \frac{d}{d\eta} \ln g(\eta). \quad (4.119)$$

Thus y and η must related, and we denote this relation through $\eta = \psi(y)$.

Following Nelder and Wedderburn (1972), we define a *generalized linear model* to be one for which y is a nonlinear function of a linear combination of the input (or feature) variables so that

$$y = f(\mathbf{w}^T \boldsymbol{\phi}) \quad (4.120)$$

where $f(\cdot)$ is known as the *activation function* in the machine learning literature, and $f^{-1}(\cdot)$ is known as the *link function* in statistics.

Now consider the log likelihood function for this model, which, as a function of η , is given by

$$\ln p(\mathbf{t}|\eta, s) = \sum_{n=1}^N \ln p(t_n|\eta, s) = \sum_{n=1}^N \left\{ \ln g(\eta_n) + \frac{\eta_n t_n}{s} \right\} + \text{const} \quad (4.121)$$

where we are assuming that all observations share a common scale parameter (which corresponds to the noise variance for a Gaussian distribution for instance) and so s is independent of n . The derivative of the log likelihood with respect to the model parameters \mathbf{w} is then given by

$$\begin{aligned} \nabla_{\mathbf{w}} \ln p(\mathbf{t}|\eta, s) &= \sum_{n=1}^N \left\{ \frac{d}{d\eta_n} \ln g(\eta_n) + \frac{t_n}{s} \right\} \frac{d\eta_n}{dy_n} \frac{dy_n}{da_n} \nabla a_n \\ &= \sum_{n=1}^N \frac{1}{s} \{t_n - y_n\} \psi'(y_n) f'(a_n) \boldsymbol{\phi}_n \end{aligned} \quad (4.122)$$

where $a_n = \mathbf{w}^T \boldsymbol{\phi}_n$, and we have used $y_n = f(a_n)$ together with the result (4.119) for $\mathbb{E}[t|\eta]$. We now see that there is a considerable simplification if we choose a particular form for the link function $f^{-1}(y)$ given by

$$f^{-1}(y) = \psi(y) \quad (4.123)$$

which gives $f(\psi(y)) = y$ and hence $f'(\psi)\psi'(y) = 1$. Also, because $a = f^{-1}(y)$, we have $a = \psi$ and hence $f'(a)\psi'(y) = 1$. In this case, the gradient of the error function reduces to

$$\nabla \ln E(\mathbf{w}) = \frac{1}{s} \sum_{n=1}^N \{y_n - t_n\} \boldsymbol{\phi}_n. \quad (4.124)$$

For the Gaussian $s = \beta^{-1}$, whereas for the logistic model $s = 1$.

4.4. The Laplace Approximation

In Section 4.5 we shall discuss the Bayesian treatment of logistic regression. As we shall see, this is more complex than the Bayesian treatment of linear regression models, discussed in Sections 3.3 and 3.5. In particular, we cannot integrate exactly

over the parameter vector \mathbf{w} since the posterior distribution is no longer Gaussian. It is therefore necessary to introduce some form of approximation. Later in the book we shall consider a range of techniques based on analytical approximations and numerical sampling.

Here we introduce a simple, but widely used, framework called the Laplace approximation, that aims to find a Gaussian approximation to a probability density defined over a set of continuous variables. Consider first the case of a single continuous variable z , and suppose the distribution $p(z)$ is defined by

$$p(z) = \frac{1}{Z} f(z) \quad (4.125)$$

where $Z = \int f(z) dz$ is the normalization coefficient. We shall suppose that the value of Z is unknown. In the Laplace method the goal is to find a Gaussian approximation $q(z)$ which is centred on a mode of the distribution $p(z)$. The first step is to find a mode of $p(z)$, in other words a point z_0 such that $p'(z_0) = 0$, or equivalently

$$\left. \frac{df(z)}{dz} \right|_{z=z_0} = 0. \quad (4.126)$$

A Gaussian distribution has the property that its logarithm is a quadratic function of the variables. We therefore consider a Taylor expansion of $\ln f(z)$ centred on the mode z_0 so that

$$\ln f(z) \simeq \ln f(z_0) - \frac{1}{2} A(z - z_0)^2 \quad (4.127)$$

where

$$A = - \left. \frac{d^2}{dz^2} \ln f(z) \right|_{z=z_0}. \quad (4.128)$$

Note that the first-order term in the Taylor expansion does not appear since z_0 is a local maximum of the distribution. Taking the exponential we obtain

$$f(z) \simeq f(z_0) \exp \left\{ -\frac{A}{2}(z - z_0)^2 \right\}. \quad (4.129)$$

We can then obtain a normalized distribution $q(z)$ by making use of the standard result for the normalization of a Gaussian, so that

$$q(z) = \left(\frac{A}{2\pi} \right)^{1/2} \exp \left\{ -\frac{A}{2}(z - z_0)^2 \right\}. \quad (4.130)$$

The Laplace approximation is illustrated in Figure 4.14. Note that the Gaussian approximation will only be well defined if its precision $A > 0$, in other words the stationary point z_0 must be a local maximum, so that the second derivative of $f(z)$ at the point z_0 is negative.

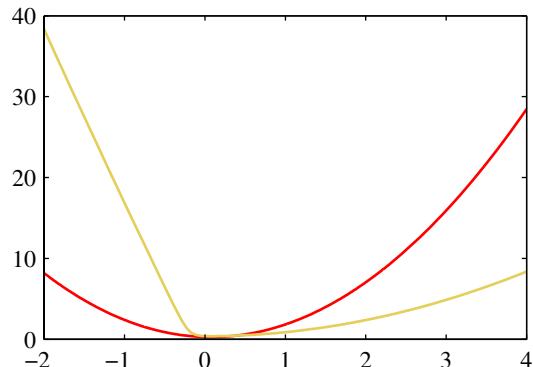
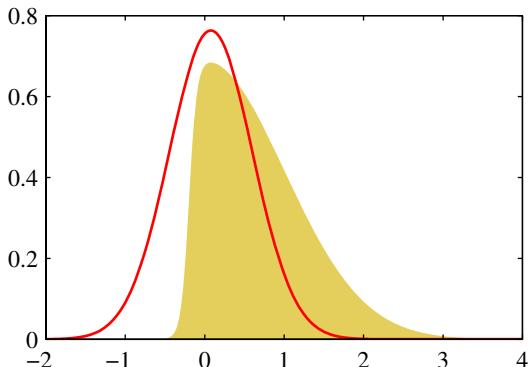


Figure 4.14 Illustration of the Laplace approximation applied to the distribution $p(z) \propto \exp(-z^2/2)\sigma(20z + 4)$, where $\sigma(z)$ is the logistic sigmoid function defined by $\sigma(z) = (1 + e^{-z})^{-1}$. The left plot shows the normalized distribution $p(z)$ in yellow, together with the Laplace approximation centred on the mode z_0 of $p(z)$ in red. The right plot shows the negative logarithms of the corresponding curves.

We can extend the Laplace method to approximate a distribution $p(\mathbf{z}) = f(\mathbf{z})/Z$ defined over an M -dimensional space \mathbf{z} . At a stationary point \mathbf{z}_0 the gradient $\nabla f(\mathbf{z})$ will vanish. Expanding around this stationary point we have

$$\ln f(\mathbf{z}) \simeq \ln f(\mathbf{z}_0) - \frac{1}{2}(\mathbf{z} - \mathbf{z}_0)^T \mathbf{A}(\mathbf{z} - \mathbf{z}_0) \quad (4.131)$$

where the $M \times M$ Hessian matrix \mathbf{A} is defined by

$$\mathbf{A} = -\nabla\nabla \ln f(\mathbf{z})|_{\mathbf{z}=\mathbf{z}_0} \quad (4.132)$$

and ∇ is the gradient operator. Taking the exponential of both sides we obtain

$$f(\mathbf{z}) \simeq f(\mathbf{z}_0) \exp \left\{ -\frac{1}{2}(\mathbf{z} - \mathbf{z}_0)^T \mathbf{A}(\mathbf{z} - \mathbf{z}_0) \right\}. \quad (4.133)$$

The distribution $q(\mathbf{z})$ is proportional to $f(\mathbf{z})$ and the appropriate normalization coefficient can be found by inspection, using the standard result (2.43) for a normalized multivariate Gaussian, giving

$$q(\mathbf{z}) = \frac{|\mathbf{A}|^{1/2}}{(2\pi)^{M/2}} \exp \left\{ -\frac{1}{2}(\mathbf{z} - \mathbf{z}_0)^T \mathbf{A}(\mathbf{z} - \mathbf{z}_0) \right\} = \mathcal{N}(\mathbf{z} | \mathbf{z}_0, \mathbf{A}^{-1}) \quad (4.134)$$

where $|\mathbf{A}|$ denotes the determinant of \mathbf{A} . This Gaussian distribution will be well defined provided its precision matrix, given by \mathbf{A} , is positive definite, which implies that the stationary point \mathbf{z}_0 must be a local maximum, not a minimum or a saddle point.

In order to apply the Laplace approximation we first need to find the mode \mathbf{z}_0 , and then evaluate the Hessian matrix at that mode. In practice a mode will typically be found by running some form of numerical optimization algorithm (Bishop

and Nabney, 2008). Many of the distributions encountered in practice will be multimodal and so there will be different Laplace approximations according to which mode is being considered. Note that the normalization constant Z of the true distribution does not need to be known in order to apply the Laplace method. As a result of the central limit theorem, the posterior distribution for a model is expected to become increasingly better approximated by a Gaussian as the number of observed data points is increased, and so we would expect the Laplace approximation to be most useful in situations where the number of data points is relatively large.

One major weakness of the Laplace approximation is that, since it is based on a Gaussian distribution, it is only directly applicable to real variables. In other cases it may be possible to apply the Laplace approximation to a transformation of the variable. For instance if $0 \leq \tau < \infty$ then we can consider a Laplace approximation of $\ln \tau$. The most serious limitation of the Laplace framework, however, is that it is based purely on the aspects of the true distribution at a specific value of the variable, and so can fail to capture important global properties. In Chapter 10 we shall consider alternative approaches which adopt a more global perspective.

4.4.1 Model comparison and BIC

As well as approximating the distribution $p(\mathbf{z})$ we can also obtain an approximation to the normalization constant Z . Using the approximation (4.133) we have

$$\begin{aligned} Z &= \int f(\mathbf{z}) d\mathbf{z} \\ &\simeq f(\mathbf{z}_0) \int \exp \left\{ -\frac{1}{2}(\mathbf{z} - \mathbf{z}_0)^T \mathbf{A} (\mathbf{z} - \mathbf{z}_0) \right\} d\mathbf{z} \\ &= f(\mathbf{z}_0) \frac{(2\pi)^{M/2}}{|\mathbf{A}|^{1/2}} \end{aligned} \quad (4.135)$$

where we have noted that the integrand is Gaussian and made use of the standard result (2.43) for a normalized Gaussian distribution. We can use the result (4.135) to obtain an approximation to the model evidence which, as discussed in Section 3.4, plays a central role in Bayesian model comparison.

Consider a data set \mathcal{D} and a set of models $\{\mathcal{M}_i\}$ having parameters $\{\boldsymbol{\theta}_i\}$. For each model we define a likelihood function $p(\mathcal{D}|\boldsymbol{\theta}_i, \mathcal{M}_i)$. If we introduce a prior $p(\boldsymbol{\theta}_i|\mathcal{M}_i)$ over the parameters, then we are interested in computing the model evidence $p(\mathcal{D}|\mathcal{M}_i)$ for the various models. From now on we omit the conditioning on \mathcal{M}_i to keep the notation uncluttered. From Bayes' theorem the model evidence is given by

$$p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (4.136)$$

Identifying $f(\boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ and $Z = p(\mathcal{D})$, and applying the result (4.135), we obtain

$$\ln p(\mathcal{D}) \simeq \ln p(\mathcal{D}|\boldsymbol{\theta}_{\text{MAP}}) + \ln p(\boldsymbol{\theta}_{\text{MAP}}) + \underbrace{\frac{M}{2} \ln(2\pi) - \frac{1}{2} \ln |\mathbf{A}|}_{\text{Occam factor}} \quad (4.137)$$

where $\boldsymbol{\theta}_{\text{MAP}}$ is the value of $\boldsymbol{\theta}$ at the mode of the posterior distribution, and \mathbf{A} is the *Hessian* matrix of second derivatives of the negative log posterior

$$\mathbf{A} = -\nabla\nabla \ln p(\mathcal{D}|\boldsymbol{\theta}_{\text{MAP}})p(\boldsymbol{\theta}_{\text{MAP}}) = -\nabla\nabla \ln p(\boldsymbol{\theta}_{\text{MAP}}|\mathcal{D}). \quad (4.138)$$

The first term on the right hand side of (4.137) represents the log likelihood evaluated using the optimized parameters, while the remaining three terms comprise the ‘Occam factor’ which penalizes model complexity.

If we assume that the Gaussian prior distribution over parameters is broad, and that the Hessian has full rank, then we can approximate (4.137) very roughly using

$$\ln p(\mathcal{D}) \simeq \ln p(\mathcal{D}|\boldsymbol{\theta}_{\text{MAP}}) - \frac{1}{2}M \ln N \quad (4.139)$$

where N is the number of data points, M is the number of parameters in $\boldsymbol{\theta}$ and we have omitted additive constants. This is known as the *Bayesian Information Criterion* (BIC) or the *Schwarz criterion* (Schwarz, 1978). Note that, compared to AIC given by (1.73), this penalizes model complexity more heavily.

Complexity measures such as AIC and BIC have the virtue of being easy to evaluate, but can also give misleading results. In particular, the assumption that the Hessian matrix has full rank is often not valid since many of the parameters are not ‘well-determined’. We can use the result (4.137) to obtain a more accurate estimate of the model evidence starting from the Laplace approximation, as we illustrate in the context of neural networks in Section 5.7.

Exercise 4.23

Section 3.5.3

4.5. Bayesian Logistic Regression

We now turn to a Bayesian treatment of logistic regression. Exact Bayesian inference for logistic regression is intractable. In particular, evaluation of the posterior distribution would require normalization of the product of a prior distribution and a likelihood function that itself comprises a product of logistic sigmoid functions, one for every data point. Evaluation of the predictive distribution is similarly intractable. Here we consider the application of the Laplace approximation to the problem of Bayesian logistic regression (Spiegelhalter and Lauritzen, 1990; MacKay, 1992b).

4.5.1 Laplace approximation

Recall from Section 4.4 that the Laplace approximation is obtained by finding the mode of the posterior distribution and then fitting a Gaussian centred at that mode. This requires evaluation of the second derivatives of the log posterior, which is equivalent to finding the Hessian matrix.

Because we seek a Gaussian representation for the posterior distribution, it is natural to begin with a Gaussian prior, which we write in the general form

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_0, \mathbf{S}_0) \quad (4.140)$$

where \mathbf{m}_0 and \mathbf{S}_0 are fixed hyperparameters. The posterior distribution over \mathbf{w} is given by

$$p(\mathbf{w}|\mathbf{t}) \propto p(\mathbf{w})p(\mathbf{t}|\mathbf{w}) \quad (4.141)$$

where $\mathbf{t} = (t_1, \dots, t_N)^T$. Taking the log of both sides, and substituting for the prior distribution using (4.140), and for the likelihood function using (4.89), we obtain

$$\begin{aligned} \ln p(\mathbf{w}|\mathbf{t}) &= -\frac{1}{2}(\mathbf{w} - \mathbf{m}_0)^T \mathbf{S}_0^{-1} (\mathbf{w} - \mathbf{m}_0) \\ &\quad + \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} + \text{const} \end{aligned} \quad (4.142)$$

where $y_n = \sigma(\mathbf{w}^T \phi_n)$. To obtain a Gaussian approximation to the posterior distribution, we first maximize the posterior distribution to give the MAP (maximum posterior) solution \mathbf{w}_{MAP} , which defines the mean of the Gaussian. The covariance is then given by the inverse of the matrix of second derivatives of the negative log likelihood, which takes the form

$$\mathbf{S}_N = -\nabla \nabla \ln p(\mathbf{w}|\mathbf{t}) = \mathbf{S}_0^{-1} + \sum_{n=1}^N y_n(1 - y_n) \phi_n \phi_n^T. \quad (4.143)$$

The Gaussian approximation to the posterior distribution therefore takes the form

$$q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\text{MAP}}, \mathbf{S}_N). \quad (4.144)$$

Having obtained a Gaussian approximation to the posterior distribution, there remains the task of marginalizing with respect to this distribution in order to make predictions.

4.5.2 Predictive distribution

The predictive distribution for class \mathcal{C}_1 , given a new feature vector $\phi(\mathbf{x})$, is obtained by marginalizing with respect to the posterior distribution $p(\mathbf{w}|\mathbf{t})$, which is itself approximated by a Gaussian distribution $q(\mathbf{w})$ so that

$$p(\mathcal{C}_1|\phi, \mathbf{t}) = \int p(\mathcal{C}_1|\phi, \mathbf{w}) p(\mathbf{w}|\mathbf{t}) d\mathbf{w} \simeq \int \sigma(\mathbf{w}^T \phi) q(\mathbf{w}) d\mathbf{w} \quad (4.145)$$

with the corresponding probability for class \mathcal{C}_2 given by $p(\mathcal{C}_2|\phi, \mathbf{t}) = 1 - p(\mathcal{C}_1|\phi, \mathbf{t})$. To evaluate the predictive distribution, we first note that the function $\sigma(\mathbf{w}^T \phi)$ depends on \mathbf{w} only through its projection onto ϕ . Denoting $a = \mathbf{w}^T \phi$, we have

$$\sigma(\mathbf{w}^T \phi) = \int \delta(a - \mathbf{w}^T \phi) \sigma(a) da \quad (4.146)$$

where $\delta(\cdot)$ is the Dirac delta function. From this we obtain

$$\int \sigma(\mathbf{w}^T \phi) q(\mathbf{w}) d\mathbf{w} = \int \sigma(a) p(a) da \quad (4.147)$$

where

$$p(a) = \int \delta(a - \mathbf{w}^T \boldsymbol{\phi}) q(\mathbf{w}) d\mathbf{w}. \quad (4.148)$$

We can evaluate $p(a)$ by noting that the delta function imposes a linear constraint on \mathbf{w} and so forms a marginal distribution from the joint distribution $q(\mathbf{w})$ by integrating out all directions orthogonal to $\boldsymbol{\phi}$. Because $q(\mathbf{w})$ is Gaussian, we know from Section 2.3.2 that the marginal distribution will also be Gaussian. We can evaluate the mean and covariance of this distribution by taking moments, and interchanging the order of integration over a and \mathbf{w} , so that

$$\mu_a = \mathbb{E}[a] = \int p(a) a da = \int q(\mathbf{w}) \mathbf{w}^T \boldsymbol{\phi} d\mathbf{w} = \mathbf{w}_{\text{MAP}}^T \boldsymbol{\phi} \quad (4.149)$$

where we have used the result (4.144) for the variational posterior distribution $q(\mathbf{w})$. Similarly

$$\begin{aligned} \sigma_a^2 &= \text{var}[a] = \int p(a) \{a^2 - \mathbb{E}[a]^2\} da \\ &= \int q(\mathbf{w}) \{(\mathbf{w}^T \boldsymbol{\phi})^2 - (\mathbf{m}_N^T \boldsymbol{\phi})^2\} d\mathbf{w} = \boldsymbol{\phi}^T \mathbf{S}_N \boldsymbol{\phi}. \end{aligned} \quad (4.150)$$

Note that the distribution of a takes the same form as the predictive distribution (3.58) for the linear regression model, with the noise variance set to zero. Thus our variational approximation to the predictive distribution becomes

$$p(\mathcal{C}_1 | \mathbf{t}) = \int \sigma(a) p(a) da = \int \sigma(a) \mathcal{N}(a | \mu_a, \sigma_a^2) da. \quad (4.151)$$

This result can also be derived directly by making use of the results for the marginal of a Gaussian distribution given in Section 2.3.2.

The integral over a represents the convolution of a Gaussian with a logistic sigmoid, and cannot be evaluated analytically. We can, however, obtain a good approximation (Spiegelhalter and Lauritzen, 1990; MacKay, 1992b; Barber and Bishop, 1998a) by making use of the close similarity between the logistic sigmoid function $\sigma(a)$ defined by (4.59) and the probit function $\Phi(a)$ defined by (4.114). In order to obtain the best approximation to the logistic function we need to re-scale the horizontal axis, so that we approximate $\sigma(a)$ by $\Phi(\lambda a)$. We can find a suitable value of λ by requiring that the two functions have the same slope at the origin, which gives $\lambda^2 = \pi/8$. The similarity of the logistic sigmoid and the probit function, for this choice of λ , is illustrated in Figure 4.9.

The advantage of using a probit function is that its convolution with a Gaussian can be expressed analytically in terms of another probit function. Specifically we can show that

$$\int \Phi(\lambda a) \mathcal{N}(a | \mu, \sigma^2) da = \Phi \left(\frac{\mu}{(\lambda^{-2} + \sigma^2)^{1/2}} \right). \quad (4.152)$$

Exercise 4.24

Exercise 4.25

Exercise 4.26

We now apply the approximation $\sigma(a) \simeq \Phi(\lambda a)$ to the probit functions appearing on both sides of this equation, leading to the following approximation for the convolution of a logistic sigmoid with a Gaussian

$$\int \sigma(a)\mathcal{N}(a|\mu, \sigma^2) da \simeq \sigma(\kappa(\sigma^2)\mu) \quad (4.153)$$

where we have defined

$$\kappa(\sigma^2) = (1 + \pi\sigma^2/8)^{-1/2}. \quad (4.154)$$

Applying this result to (4.151) we obtain the approximate predictive distribution in the form

$$p(\mathcal{C}_1|\phi, \mathbf{t}) = \sigma(\kappa(\sigma_a^2)\mu_a) \quad (4.155)$$

where μ_a and σ_a^2 are defined by (4.149) and (4.150), respectively, and $\kappa(\sigma_a^2)$ is defined by (4.154).

Note that the decision boundary corresponding to $p(\mathcal{C}_1|\phi, \mathbf{t}) = 0.5$ is given by $\mu_a = 0$, which is the same as the decision boundary obtained by using the MAP value for \mathbf{w} . Thus if the decision criterion is based on minimizing misclassification rate, with equal prior probabilities, then the marginalization over \mathbf{w} has no effect. However, for more complex decision criteria it will play an important role. Marginalization of the logistic sigmoid model under a Gaussian approximation to the posterior distribution will be illustrated in the context of variational inference in Figure 10.13.

Exercises

- 4.1** (**) Given a set of data points $\{\mathbf{x}_n\}$, we can define the *convex hull* to be the set of all points \mathbf{x} given by

$$\mathbf{x} = \sum_n \alpha_n \mathbf{x}_n \quad (4.156)$$

where $\alpha_n \geq 0$ and $\sum_n \alpha_n = 1$. Consider a second set of points $\{\mathbf{y}_n\}$ together with their corresponding convex hull. By definition, the two sets of points will be linearly separable if there exists a vector $\hat{\mathbf{w}}$ and a scalar w_0 such that $\hat{\mathbf{w}}^T \mathbf{x}_n + w_0 > 0$ for all \mathbf{x}_n , and $\hat{\mathbf{w}}^T \mathbf{y}_n + w_0 < 0$ for all \mathbf{y}_n . Show that if their convex hulls intersect, the two sets of points cannot be linearly separable, and conversely that if they are linearly separable, their convex hulls do not intersect.

- 4.2** (**) **www** Consider the minimization of a sum-of-squares error function (4.15), and suppose that all of the target vectors in the training set satisfy a linear constraint

$$\mathbf{a}^T \mathbf{t}_n + b = 0 \quad (4.157)$$

where \mathbf{t}_n corresponds to the n^{th} row of the matrix \mathbf{T} in (4.15). Show that as a consequence of this constraint, the elements of the model prediction $\mathbf{y}(\mathbf{x})$ given by the least-squares solution (4.17) also satisfy this constraint, so that

$$\mathbf{a}^T \mathbf{y}(\mathbf{x}) + b = 0. \quad (4.158)$$

To do so, assume that one of the basis functions $\phi_0(\mathbf{x}) = 1$ so that the corresponding parameter w_0 plays the role of a bias.

- 4.3** (**) Extend the result of Exercise 4.2 to show that if multiple linear constraints are satisfied simultaneously by the target vectors, then the same constraints will also be satisfied by the least-squares prediction of a linear model.
- 4.4** (*) **www** Show that maximization of the class separation criterion given by (4.23) with respect to \mathbf{w} , using a Lagrange multiplier to enforce the constraint $\mathbf{w}^T \mathbf{w} = 1$, leads to the result that $\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1)$.
- 4.5** (*) By making use of (4.20), (4.23), and (4.24), show that the Fisher criterion (4.25) can be written in the form (4.26).
- 4.6** (*) Using the definitions of the between-class and within-class covariance matrices given by (4.27) and (4.28), respectively, together with (4.34) and (4.36) and the choice of target values described in Section 4.1.5, show that the expression (4.33) that minimizes the sum-of-squares error function can be written in the form (4.37).
- 4.7** (*) **www** Show that the logistic sigmoid function (4.59) satisfies the property $\sigma(-a) = 1 - \sigma(a)$ and that its inverse is given by $\sigma^{-1}(y) = \ln\{y/(1-y)\}$.
- 4.8** (*) Using (4.57) and (4.58), derive the result (4.65) for the posterior class probability in the two-class generative model with Gaussian densities, and verify the results (4.66) and (4.67) for the parameters \mathbf{w} and w_0 .
- 4.9** (*) **www** Consider a generative classification model for K classes defined by prior class probabilities $p(\mathcal{C}_k) = \pi_k$ and general class-conditional densities $p(\phi|\mathcal{C}_k)$ where ϕ is the input feature vector. Suppose we are given a training data set $\{\phi_n, \mathbf{t}_n\}$ where $n = 1, \dots, N$, and \mathbf{t}_n is a binary target vector of length K that uses the 1-of- K coding scheme, so that it has components $t_{nj} = I_{jk}$ if pattern n is from class \mathcal{C}_k . Assuming that the data points are drawn independently from this model, show that the maximum-likelihood solution for the prior probabilities is given by

$$\pi_k = \frac{N_k}{N} \quad (4.159)$$

where N_k is the number of data points assigned to class \mathcal{C}_k .

- 4.10** (**) Consider the classification model of Exercise 4.9 and now suppose that the class-conditional densities are given by Gaussian distributions with a shared covariance matrix, so that

$$p(\phi|\mathcal{C}_k) = \mathcal{N}(\phi|\boldsymbol{\mu}_k, \Sigma). \quad (4.160)$$

Show that the maximum likelihood solution for the mean of the Gaussian distribution for class \mathcal{C}_k is given by

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N_k} t_{nk} \phi_n \quad (4.161)$$

which represents the mean of those feature vectors assigned to class \mathcal{C}_k . Similarly, show that the maximum likelihood solution for the shared covariance matrix is given by

$$\Sigma = \sum_{k=1}^K \frac{N_k}{N} \mathbf{S}_k \quad (4.162)$$

where

$$\mathbf{S}_k = \frac{1}{N_k} \sum_{n=1}^N t_{nk} (\phi_n - \mu_k)(\phi_n - \mu_k)^T. \quad (4.163)$$

Thus Σ is given by a weighted average of the covariances of the data associated with each class, in which the weighting coefficients are given by the prior probabilities of the classes.

- 4.11** (**) Consider a classification problem with K classes for which the feature vector ϕ has M components each of which can take L discrete states. Let the values of the components be represented by a 1-of- L binary coding scheme. Further suppose that, conditioned on the class \mathcal{C}_k , the M components of ϕ are independent, so that the class-conditional density factorizes with respect to the feature vector components. Show that the quantities a_k given by (4.63), which appear in the argument to the softmax function describing the posterior class probabilities, are linear functions of the components of ϕ . Note that this represents an example of the naive Bayes model which is discussed in Section 8.2.2.
- 4.12** (*) **www** Verify the relation (4.88) for the derivative of the logistic sigmoid function defined by (4.59).
- 4.13** (*) **www** By making use of the result (4.88) for the derivative of the logistic sigmoid, show that the derivative of the error function (4.90) for the logistic regression model is given by (4.91).
- 4.14** (*) Show that for a linearly separable data set, the maximum likelihood solution for the logistic regression model is obtained by finding a vector w whose decision boundary $w^T \phi(x) = 0$ separates the classes and then taking the magnitude of w to infinity.
- 4.15** (**) Show that the Hessian matrix H for the logistic regression model, given by (4.97), is positive definite. Here R is a diagonal matrix with elements $y_n(1 - y_n)$, and y_n is the output of the logistic regression model for input vector x_n . Hence show that the error function is a concave function of w and that it has a unique minimum.
- 4.16** (*) Consider a binary classification problem in which each observation x_n is known to belong to one of two classes, corresponding to $t = 0$ and $t = 1$, and suppose that the procedure for collecting training data is imperfect, so that training points are sometimes mislabelled. For every data point x_n , instead of having a value t for the class label, we have instead a value π_n representing the probability that $t_n = 1$. Given a probabilistic model $p(t = 1|\phi)$, write down the log likelihood function appropriate to such a data set.

- 4.17** (*) **www** Show that the derivatives of the softmax activation function (4.104), where the a_k are defined by (4.105), are given by (4.106).
- 4.18** (*) Using the result (4.91) for the derivatives of the softmax activation function, show that the gradients of the cross-entropy error (4.108) are given by (4.109).
- 4.19** (*) **www** Write down expressions for the gradient of the log likelihood, as well as the corresponding Hessian matrix, for the probit regression model defined in Section 4.3.5. These are the quantities that would be required to train such a model using IRLS.
- 4.20** (**) Show that the Hessian matrix for the multiclass logistic regression problem, defined by (4.110), is positive semidefinite. Note that the full Hessian matrix for this problem is of size $MK \times MK$, where M is the number of parameters and K is the number of classes. To prove the positive semidefinite property, consider the product $\mathbf{u}^T \mathbf{H} \mathbf{u}$ where \mathbf{u} is an arbitrary vector of length MK , and then apply Jensen's inequality.
- 4.21** (*) Show that the probit function (4.114) and the erf function (4.115) are related by (4.116).
- 4.22** (*) Using the result (4.135), derive the expression (4.137) for the log model evidence under the Laplace approximation.
- 4.23** (**) **www** In this exercise, we derive the BIC result (4.139) starting from the Laplace approximation to the model evidence given by (4.137). Show that if the prior over parameters is Gaussian of the form $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \mathbf{m}, \mathbf{V}_0)$, the log model evidence under the Laplace approximation takes the form
- $$\ln p(\mathcal{D}) \simeq \ln p(\mathcal{D} | \boldsymbol{\theta}_{\text{MAP}}) - \frac{1}{2} (\boldsymbol{\theta}_{\text{MAP}} - \mathbf{m})^T \mathbf{V}_0^{-1} (\boldsymbol{\theta}_{\text{MAP}} - \mathbf{m}) - \frac{1}{2} \ln |\mathbf{H}| + \text{const}$$
- where \mathbf{H} is the matrix of second derivatives of the log likelihood $\ln p(\mathcal{D} | \boldsymbol{\theta})$ evaluated at $\boldsymbol{\theta}_{\text{MAP}}$. Now assume that the prior is broad so that \mathbf{V}_0^{-1} is small and the second term on the right-hand side above can be neglected. Furthermore, consider the case of independent, identically distributed data so that \mathbf{H} is the sum of terms one for each data point. Show that the log model evidence can then be written approximately in the form of the BIC expression (4.139).
- 4.24** (**) Use the results from Section 2.3.2 to derive the result (4.151) for the marginalization of the logistic regression model with respect to a Gaussian posterior distribution over the parameters \mathbf{w} .
- 4.25** (**) Suppose we wish to approximate the logistic sigmoid $\sigma(a)$ defined by (4.59) by a scaled probit function $\Phi(\lambda a)$, where $\Phi(a)$ is defined by (4.114). Show that if λ is chosen so that the derivatives of the two functions are equal at $a = 0$, then $\lambda^2 = \pi/8$.

4.26 (**) In this exercise, we prove the relation (4.152) for the convolution of a probit function with a Gaussian distribution. To do this, show that the derivative of the left-hand side with respect to μ is equal to the derivative of the right-hand side, and then integrate both sides with respect to μ and then show that the constant of integration vanishes. Note that before differentiating the left-hand side, it is convenient first to introduce a change of variable given by $a = \mu + \sigma z$ so that the integral over a is replaced by an integral over z . When we differentiate the left-hand side of the relation (4.152), we will then obtain a Gaussian integral over z that can be evaluated analytically.

5

Neural Networks



In Chapters 3 and 4 we considered models for regression and classification that comprised linear combinations of fixed basis functions. We saw that such models have useful analytical and computational properties but that their practical applicability was limited by the curse of dimensionality. In order to apply such models to large-scale problems, it is necessary to adapt the basis functions to the data.

Support vector machines (SVMs), discussed in Chapter 7, address this by first defining basis functions that are centred on the training data points and then selecting a subset of these during training. One advantage of SVMs is that, although the training involves nonlinear optimization, the objective function is convex, and so the solution of the optimization problem is relatively straightforward. The number of basis functions in the resulting models is generally much smaller than the number of training points, although it is often still relatively large and typically increases with the size of the training set. The relevance vector machine, discussed in Section 7.2, also chooses a subset from a fixed set of basis functions and typically results in much

sparser models. Unlike the SVM it also produces probabilistic outputs, although this is at the expense of a nonconvex optimization during training.

An alternative approach is to fix the number of basis functions in advance but allow them to be adaptive, in other words to use parametric forms for the basis functions in which the parameter values are adapted during training. The most successful model of this type in the context of pattern recognition is the feed-forward neural network, also known as the *multilayer perceptron*, discussed in this chapter. In fact, ‘multilayer perceptron’ is really a misnomer, because the model comprises multiple layers of logistic regression models (with continuous nonlinearities) rather than multiple perceptrons (with discontinuous nonlinearities). For many applications, the resulting model can be significantly more compact, and hence faster to evaluate, than a support vector machine having the same generalization performance. The price to be paid for this compactness, as with the relevance vector machine, is that the likelihood function, which forms the basis for network training, is no longer a convex function of the model parameters. In practice, however, it is often worth investing substantial computational resources during the training phase in order to obtain a compact model that is fast at processing new data.

The term ‘neural network’ has its origins in attempts to find mathematical representations of information processing in biological systems (McCulloch and Pitts, 1943; Widrow and Hoff, 1960; Rosenblatt, 1962; Rumelhart *et al.*, 1986). Indeed, it has been used very broadly to cover a wide range of different models, many of which have been the subject of exaggerated claims regarding their biological plausibility. From the perspective of practical applications of pattern recognition, however, biological realism would impose entirely unnecessary constraints. Our focus in this chapter is therefore on neural networks as efficient models for statistical pattern recognition. In particular, we shall restrict our attention to the specific class of neural networks that have proven to be of greatest practical value, namely the multilayer perceptron.

We begin by considering the functional form of the network model, including the specific parameterization of the basis functions, and we then discuss the problem of determining the network parameters within a maximum likelihood framework, which involves the solution of a nonlinear optimization problem. This requires the evaluation of derivatives of the log likelihood function with respect to the network parameters, and we shall see how these can be obtained efficiently using the technique of *error backpropagation*. We shall also show how the backpropagation framework can be extended to allow other derivatives to be evaluated, such as the Jacobian and Hessian matrices. Next we discuss various approaches to regularization of neural network training and the relationships between them. We also consider some extensions to the neural network model, and in particular we describe a general framework for modelling conditional probability distributions known as *mixture density networks*. Finally, we discuss the use of Bayesian treatments of neural networks. Additional background on neural network models can be found in Bishop (1995a).

5.1. Feed-forward Network Functions

The linear models for regression and classification discussed in Chapters 3 and 4, respectively, are based on linear combinations of fixed nonlinear basis functions $\phi_j(\mathbf{x})$ and take the form

$$y(\mathbf{x}, \mathbf{w}) = f \left(\sum_{j=1}^M w_j \phi_j(\mathbf{x}) \right) \quad (5.1)$$

where $f(\cdot)$ is a nonlinear activation function in the case of classification and is the identity in the case of regression. Our goal is to extend this model by making the basis functions $\phi_j(\mathbf{x})$ depend on parameters and then to allow these parameters to be adjusted, along with the coefficients $\{w_j\}$, during training. There are, of course, many ways to construct parametric nonlinear basis functions. Neural networks use basis functions that follow the same form as (5.1), so that each basis function is itself a nonlinear function of a linear combination of the inputs, where the coefficients in the linear combination are adaptive parameters.

This leads to the basic neural network model, which can be described a series of functional transformations. First we construct M linear combinations of the input variables x_1, \dots, x_D in the form

$$a_j = \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \quad (5.2)$$

where $j = 1, \dots, M$, and the superscript (1) indicates that the corresponding parameters are in the first ‘layer’ of the network. We shall refer to the parameters $w_{ji}^{(1)}$ as *weights* and the parameters $w_{j0}^{(1)}$ as *biases*, following the nomenclature of Chapter 3. The quantities a_j are known as *activations*. Each of them is then transformed using a differentiable, nonlinear *activation function* $h(\cdot)$ to give

$$z_j = h(a_j). \quad (5.3)$$

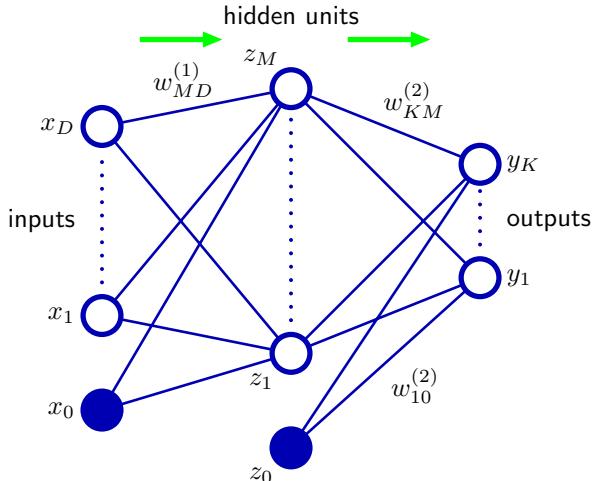
These quantities correspond to the outputs of the basis functions in (5.1) that, in the context of neural networks, are called *hidden units*. The nonlinear functions $h(\cdot)$ are generally chosen to be sigmoidal functions such as the logistic sigmoid or the ‘tanh’ function. Following (5.1), these values are again linearly combined to give *output unit activations*

$$a_k = \sum_{j=1}^M w_{kj}^{(2)} z_j + w_{k0}^{(2)} \quad (5.4)$$

where $k = 1, \dots, K$, and K is the total number of outputs. This transformation corresponds to the second layer of the network, and again the $w_{k0}^{(2)}$ are bias parameters. Finally, the output unit activations are transformed using an appropriate activation function to give a set of network outputs y_k . The choice of activation function is determined by the nature of the data and the assumed distribution of target variables

Exercise 5.1

Figure 5.1 Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables x_0 and z_0 . Arrows denote the direction of information flow through the network during forward propagation.



and follows the same considerations as for linear models discussed in Chapters 3 and 4. Thus for standard regression problems, the activation function is the identity so that $y_k = a_k$. Similarly, for multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that

$$y_k = \sigma(a_k) \quad (5.5)$$

where

$$\sigma(a) = \frac{1}{1 + \exp(-a)}. \quad (5.6)$$

Finally, for multiclass problems, a softmax activation function of the form (4.62) is used. The choice of output unit activation function is discussed in detail in Section 5.2.

We can combine these various stages to give the overall network function that, for sigmoidal output unit activation functions, takes the form

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\sum_{j=1}^M w_{kj}^{(2)} h \left(\sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \quad (5.7)$$

where the set of all weight and bias parameters have been grouped together into a vector \mathbf{w} . Thus the neural network model is simply a nonlinear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by a vector \mathbf{w} of adjustable parameters.

This function can be represented in the form of a network diagram as shown in Figure 5.1. The process of evaluating (5.7) can then be interpreted as a *forward propagation* of information through the network. It should be emphasized that these diagrams do not represent probabilistic graphical models of the kind to be considered in Chapter 8 because the internal nodes represent deterministic variables rather than stochastic ones. For this reason, we have adopted a slightly different graphical

notation for the two kinds of model. We shall see later how to give a probabilistic interpretation to a neural network.

As discussed in Section 3.1, the bias parameters in (5.2) can be absorbed into the set of weight parameters by defining an additional input variable x_0 whose value is clamped at $x_0 = 1$, so that (5.2) takes the form

$$a_j = \sum_{i=0}^D w_{ji}^{(1)} x_i. \quad (5.8)$$

We can similarly absorb the second-layer biases into the second-layer weights, so that the overall network function becomes

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\sum_{j=0}^M w_{kj}^{(2)} h \left(\sum_{i=0}^D w_{ji}^{(1)} x_i \right) \right). \quad (5.9)$$

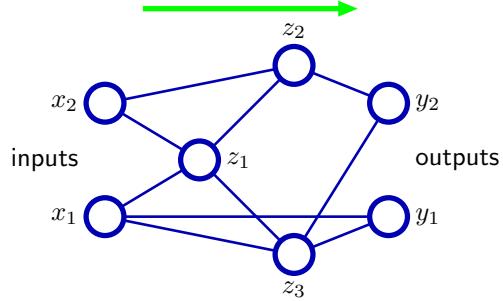
As can be seen from Figure 5.1, the neural network model comprises two stages of processing, each of which resembles the perceptron model of Section 4.1.7, and for this reason the neural network is also known as the *multilayer perceptron*, or MLP. A key difference compared to the perceptron, however, is that the neural network uses continuous sigmoidal nonlinearities in the hidden units, whereas the perceptron uses step-function nonlinearities. This means that the neural network function is differentiable with respect to the network parameters, and this property will play a central role in network training.

If the activation functions of all the hidden units in a network are taken to be linear, then for any such network we can always find an equivalent network without hidden units. This follows from the fact that the composition of successive linear transformations is itself a linear transformation. However, if the number of hidden units is smaller than either the number of input or output units, then the transformations that the network can generate are not the most general possible linear transformations from inputs to outputs because information is lost in the dimensionality reduction at the hidden units. In Section 12.4.2, we show that networks of linear units give rise to principal component analysis. In general, however, there is little interest in multilayer networks of linear units.

The network architecture shown in Figure 5.1 is the most commonly used one in practice. However, it is easily generalized, for instance by considering additional layers of processing each consisting of a weighted linear combination of the form (5.4) followed by an element-wise transformation using a nonlinear activation function. Note that there is some confusion in the literature regarding the terminology for counting the number of layers in such networks. Thus the network in Figure 5.1 may be described as a 3-layer network (which counts the number of layers of units, and treats the inputs as units) or sometimes as a single-hidden-layer network (which counts the number of layers of hidden units). We recommend a terminology in which Figure 5.1 is called a two-layer network, because it is the number of layers of adaptive weights that is important for determining the network properties.

Another generalization of the network architecture is to include *skip-layer* connections, each of which is associated with a corresponding adaptive parameter. For

Figure 5.2 Example of a neural network having a general feed-forward topology. Note that each hidden and output unit has an associated bias parameter (omitted for clarity).



instance, in a two-layer network these would go directly from inputs to outputs. In principle, a network with sigmoidal hidden units can always mimic skip layer connections (for bounded input values) by using a sufficiently small first-layer weight that, over its operating range, the hidden unit is effectively linear, and then compensating with a large weight value from the hidden unit to the output. In practice, however, it may be advantageous to include skip-layer connections explicitly.

Furthermore, the network can be sparse, with not all possible connections within a layer being present. We shall see an example of a sparse network architecture when we consider convolutional neural networks in Section 5.5.6.

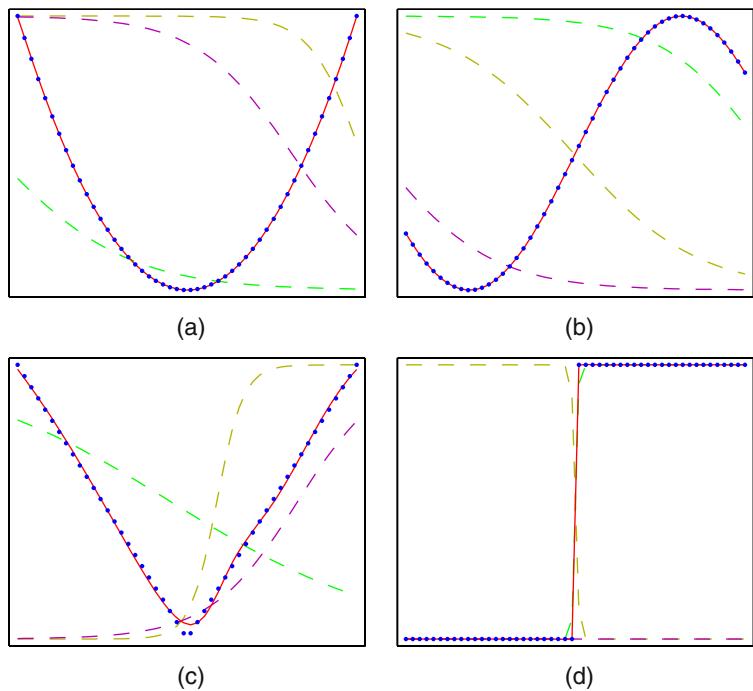
Because there is a direct correspondence between a network diagram and its mathematical function, we can develop more general network mappings by considering more complex network diagrams. However, these must be restricted to a *feed-forward* architecture, in other words to one having no closed directed cycles, to ensure that the outputs are deterministic functions of the inputs. This is illustrated with a simple example in Figure 5.2. Each (hidden or output) unit in such a network computes a function given by

$$z_k = h \left(\sum_j w_{kj} z_j \right) \quad (5.10)$$

where the sum runs over all units that send connections to unit k (and a bias parameter is included in the summation). For a given set of values applied to the inputs of the network, successive application of (5.10) allows the activations of all units in the network to be evaluated including those of the output units.

The approximation properties of feed-forward networks have been widely studied (Funahashi, 1989; Cybenko, 1989; Hornik *et al.*, 1989; Stinchcombe and White, 1989; Cotter, 1990; Ito, 1991; Hornik, 1991; Kreinovich, 1991; Ripley, 1996) and found to be very general. Neural networks are therefore said to be *universal approximators*. For example, a two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units. This result holds for a wide range of hidden unit activation functions, but excluding polynomials. Although such theorems are reassuring, the key problem is how to find suitable parameter values given a set of training data, and in later sections of this chapter we

Figure 5.3 Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c), $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown as blue dots, have been sampled uniformly in x over the interval $(-1, 1)$ and the corresponding values of $f(x)$ evaluated. These data points are then used to train a two-layer network having 3 hidden units with ‘tanh’ activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.



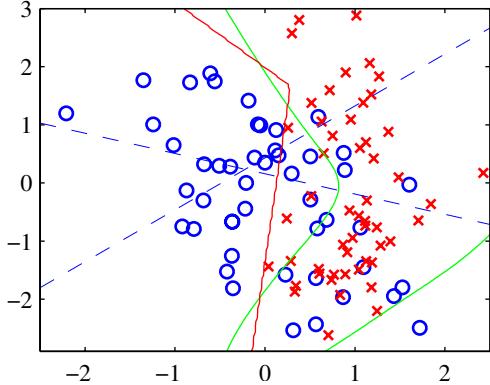
will show that there exist effective solutions to this problem based on both maximum likelihood and Bayesian approaches.

The capability of a two-layer network to model a broad range of functions is illustrated in Figure 5.3. This figure also shows how individual hidden units work collaboratively to approximate the final function. The role of hidden units in a simple classification problem is illustrated in Figure 5.4 using the synthetic classification data set described in Appendix A.

5.1.1 Weight-space symmetries

One property of feed-forward networks, which will play a role when we consider Bayesian model comparison, is that multiple distinct choices for the weight vector w can all give rise to the same mapping function from inputs to outputs (Chen *et al.*, 1993). Consider a two-layer network of the form shown in Figure 5.1 with M hidden units having ‘tanh’ activation functions and full connectivity in both layers. If we change the sign of all of the weights and the bias feeding into a particular hidden unit, then, for a given input pattern, the sign of the activation of the hidden unit will be reversed, because ‘tanh’ is an odd function, so that $\tanh(-a) = -\tanh(a)$. This transformation can be exactly compensated by changing the sign of all of the weights leading out of that hidden unit. Thus, by changing the signs of a particular group of weights (and a bias), the input–output mapping function represented by the network is unchanged, and so we have found two different weight vectors that give rise to the same mapping function. For M hidden units, there will be M such ‘sign-flip’

Figure 5.4 Example of the solution of a simple two-class classification problem involving synthetic data using a neural network having two inputs, two hidden units with ‘tanh’ activation functions, and a single output having a logistic sigmoid activation function. The dashed blue lines show the $z = 0.5$ contours for each of the hidden units, and the red line shows the $y = 0.5$ decision surface for the network. For comparison, the green line denotes the optimal decision boundary computed from the distributions used to generate the data.



symmetries, and thus any given weight vector will be one of a set 2^M equivalent weight vectors .

Similarly, imagine that we interchange the values of all of the weights (and the bias) leading both into and out of a particular hidden unit with the corresponding values of the weights (and bias) associated with a different hidden unit. Again, this clearly leaves the network input–output mapping function unchanged, but it corresponds to a different choice of weight vector. For M hidden units, any given weight vector will belong to a set of $M!$ equivalent weight vectors associated with this interchange symmetry, corresponding to the $M!$ different orderings of the hidden units. The network will therefore have an overall weight-space symmetry factor of $M!2^M$. For networks with more than two layers of weights, the total level of symmetry will be given by the product of such factors, one for each layer of hidden units.

It turns out that these factors account for all of the symmetries in weight space (except for possible accidental symmetries due to specific choices for the weight values). Furthermore, the existence of these symmetries is not a particular property of the ‘tanh’ function but applies to a wide range of activation functions (Kúrková and Kainen, 1994). In many cases, these symmetries in weight space are of little practical consequence, although in Section 5.7 we shall encounter a situation in which we need to take them into account.

5.2. Network Training

So far, we have viewed neural networks as a general class of parametric nonlinear functions from a vector \mathbf{x} of input variables to a vector \mathbf{y} of output variables. A simple approach to the problem of determining the network parameters is to make an analogy with the discussion of polynomial curve fitting in Section 1.1, and therefore to minimize a sum-of-squares error function. Given a training set comprising a set of input vectors $\{\mathbf{x}_n\}$, where $n = 1, \dots, N$, together with a corresponding set of

target vectors $\{\mathbf{t}_n\}$, we minimize the error function

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2. \quad (5.11)$$

However, we can provide a much more general view of network training by first giving a probabilistic interpretation to the network outputs. We have already seen many advantages of using probabilistic predictions in Section 1.5.4. Here it will also provide us with a clearer motivation both for the choice of output unit nonlinearity and the choice of error function.

We start by discussing regression problems, and for the moment we consider a single target variable t that can take any real value. Following the discussions in Section 1.2.5 and 3.1, we assume that t has a Gaussian distribution with an \mathbf{x} -dependent mean, which is given by the output of the neural network, so that

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \quad (5.12)$$

where β is the precision (inverse variance) of the Gaussian noise. Of course this is a somewhat restrictive assumption, and in Section 5.6 we shall see how to extend this approach to allow for more general conditional distributions. For the conditional distribution given by (5.12), it is sufficient to take the output unit activation function to be the identity, because such a network can approximate any continuous function from \mathbf{x} to y . Given a data set of N independent, identically distributed observations $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, along with corresponding target values $\mathbf{t} = \{t_1, \dots, t_N\}$, we can construct the corresponding likelihood function

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N p(t_n|\mathbf{x}_n, \mathbf{w}, \beta).$$

Taking the negative logarithm, we obtain the error function

$$\frac{\beta}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2} \ln \beta + \frac{N}{2} \ln(2\pi) \quad (5.13)$$

which can be used to learn the parameters \mathbf{w} and β . In Section 5.7, we shall discuss the Bayesian treatment of neural networks, while here we consider a maximum likelihood approach. Note that in the neural networks literature, it is usual to consider the minimization of an error function rather than the maximization of the (log) likelihood, and so here we shall follow this convention. Consider first the determination of \mathbf{w} . Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error function given by

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 \quad (5.14)$$

where we have discarded additive and multiplicative constants. The value of \mathbf{w} found by minimizing $E(\mathbf{w})$ will be denoted \mathbf{w}_{ML} because it corresponds to the maximum likelihood solution. In practice, the nonlinearity of the network function $y(\mathbf{x}_n, \mathbf{w})$ causes the error $E(\mathbf{w})$ to be nonconvex, and so in practice local maxima of the likelihood may be found, corresponding to local minima of the error function, as discussed in Section 5.2.1.

Having found \mathbf{w}_{ML} , the value of β can be found by minimizing the negative log likelihood to give

$$\frac{1}{\beta_{\text{ML}}} = \frac{1}{N} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}_{\text{ML}}) - t_n\}^2. \quad (5.15)$$

Note that this can be evaluated once the iterative optimization required to find \mathbf{w}_{ML} is completed. If we have multiple target variables, and we assume that they are independent conditional on \mathbf{x} and \mathbf{w} with shared noise precision β , then the conditional distribution of the target values is given by

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{t}|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\mathbf{I}). \quad (5.16)$$

Following the same argument as for a single target variable, we see that the maximum likelihood weights are determined by minimizing the sum-of-squares error function (5.11). The noise precision is then given by

$$\frac{1}{\beta_{\text{ML}}} = \frac{1}{NK} \sum_{n=1}^N \|y(\mathbf{x}_n, \mathbf{w}_{\text{ML}}) - \mathbf{t}_n\|^2 \quad (5.17)$$

where K is the number of target variables. The assumption of independence can be dropped at the expense of a slightly more complex optimization problem.

Recall from Section 4.3.6 that there is a natural pairing of the error function (given by the negative log likelihood) and the output unit activation function. In the regression case, we can view the network as having an output activation function that is the identity, so that $y_k = a_k$. The corresponding sum-of-squares error function has the property

$$\frac{\partial E}{\partial a_k} = y_k - t_k \quad (5.18)$$

which we shall make use of when discussing error backpropagation in Section 5.3.

Now consider the case of binary classification in which we have a single target variable t such that $t = 1$ denotes class \mathcal{C}_1 and $t = 0$ denotes class \mathcal{C}_2 . Following the discussion of canonical link functions in Section 4.3.6, we consider a network having a single output whose activation function is a logistic sigmoid

$$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)} \quad (5.19)$$

so that $0 \leq y(\mathbf{x}, \mathbf{w}) \leq 1$. We can interpret $y(\mathbf{x}, \mathbf{w})$ as the conditional probability $p(\mathcal{C}_1|\mathbf{x})$, with $p(\mathcal{C}_2|\mathbf{x})$ given by $1 - y(\mathbf{x}, \mathbf{w})$. The conditional distribution of targets given inputs is then a Bernoulli distribution of the form

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}. \quad (5.20)$$

Exercise 5.2

Exercise 5.3

If we consider a training set of independent observations, then the error function, which is given by the negative log likelihood, is then a *cross-entropy* error function of the form

$$E(\mathbf{w}) = - \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \quad (5.21)$$

where y_n denotes $y(\mathbf{x}_n, \mathbf{w})$. Note that there is no analogue of the noise precision β because the target values are assumed to be correctly labelled. However, the model is easily extended to allow for labelling errors. Simard *et al.* (2003) found that using the cross-entropy error function instead of the sum-of-squares for a classification problem leads to faster training as well as improved generalization.

If we have K separate binary classifications to perform, then we can use a network having K outputs each of which has a logistic sigmoid activation function. Associated with each output is a binary class label $t_k \in \{0, 1\}$, where $k = 1, \dots, K$. If we assume that the class labels are independent, given the input vector, then the conditional distribution of the targets is

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^K y_k(\mathbf{x}, \mathbf{w})^{t_k} [1 - y_k(\mathbf{x}, \mathbf{w})]^{1-t_k}. \quad (5.22)$$

Taking the negative logarithm of the corresponding likelihood function then gives the following error function

$$E(\mathbf{w}) = - \sum_{n=1}^N \sum_{k=1}^K \{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln(1 - y_{nk})\} \quad (5.23)$$

where y_{nk} denotes $y_k(\mathbf{x}_n, \mathbf{w})$. Again, the derivative of the error function with respect to the activation for a particular output unit takes the form (5.18) just as in the regression case.

It is interesting to contrast the neural network solution to this problem with the corresponding approach based on a linear classification model of the kind discussed in Chapter 4. Suppose that we are using a standard two-layer network of the kind shown in Figure 5.1. We see that the weight parameters in the first layer of the network are shared between the various outputs, whereas in the linear model each classification problem is solved independently. The first layer of the network can be viewed as performing a nonlinear feature extraction, and the sharing of features between the different outputs can save on computation and can also lead to improved generalization.

Finally, we consider the standard multiclass classification problem in which each input is assigned to one of K mutually exclusive classes. The binary target variables $t_k \in \{0, 1\}$ have a 1-of- K coding scheme indicating the class, and the network outputs are interpreted as $y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1|\mathbf{x})$, leading to the following error function

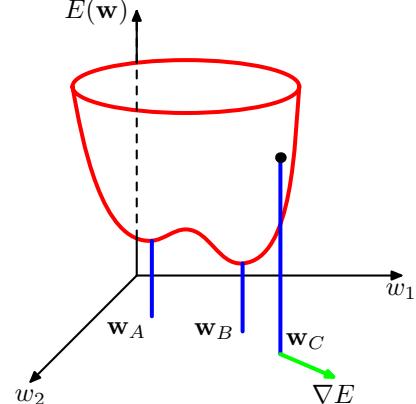
$$E(\mathbf{w}) = - \sum_{n=1}^N \sum_{k=1}^K t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}). \quad (5.24)$$

Exercise 5.4

Exercise 5.5

Exercise 5.6

Figure 5.5 Geometrical view of the error function $E(\mathbf{w})$ as a surface sitting over weight space. Point \mathbf{w}_A is a local minimum and \mathbf{w}_B is the global minimum. At any point \mathbf{w}_C , the local gradient of the error surface is given by the vector ∇E .



Following the discussion of Section 4.3.4, we see that the output unit activation function, which corresponds to the canonical link, is given by the softmax function

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))} \quad (5.25)$$

which satisfies $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$. Note that the $y_k(\mathbf{x}, \mathbf{w})$ are unchanged if a constant is added to all of the $a_k(\mathbf{x}, \mathbf{w})$, causing the error function to be constant for some directions in weight space. This degeneracy is removed if an appropriate regularization term (Section 5.5) is added to the error function.

Once again, the derivative of the error function with respect to the activation for a particular output unit takes the familiar form (5.18).

In summary, there is a natural choice of both output unit activation function and matching error function, according to the type of problem being solved. For regression we use linear outputs and a sum-of-squares error, for (multiple independent) binary classifications we use logistic sigmoid outputs and a cross-entropy error function, and for multiclass classification we use softmax outputs with the corresponding multiclass cross-entropy error function. For classification problems involving two classes, we can use a single logistic sigmoid output, or alternatively we can use a network with two outputs having a softmax output activation function.

5.2.1 Parameter optimization

We turn next to the task of finding a weight vector \mathbf{w} which minimizes the chosen function $E(\mathbf{w})$. At this point, it is useful to have a geometrical picture of the error function, which we can view as a surface sitting over weight space as shown in Figure 5.5. First note that if we make a small step in weight space from \mathbf{w} to $\mathbf{w} + \delta\mathbf{w}$ then the change in the error function is $\delta E \simeq \delta\mathbf{w}^T \nabla E(\mathbf{w})$, where the vector $\nabla E(\mathbf{w})$ points in the direction of greatest rate of increase of the error function. Because the error $E(\mathbf{w})$ is a smooth continuous function of \mathbf{w} , its smallest value will occur at a

Exercise 5.7

point in weight space such that the gradient of the error function vanishes, so that

$$\nabla E(\mathbf{w}) = 0 \quad (5.26)$$

as otherwise we could make a small step in the direction of $-\nabla E(\mathbf{w})$ and thereby further reduce the error. Points at which the gradient vanishes are called stationary points, and may be further classified into minima, maxima, and saddle points.

Our goal is to find a vector \mathbf{w} such that $E(\mathbf{w})$ takes its smallest value. However, the error function typically has a highly nonlinear dependence on the weights and bias parameters, and so there will be many points in weight space at which the gradient vanishes (or is numerically very small). Indeed, from the discussion in Section 5.1.1 we see that for any point \mathbf{w} that is a local minimum, there will be other points in weight space that are equivalent minima. For instance, in a two-layer network of the kind shown in Figure 5.1, with M hidden units, each point in weight space is a member of a family of $M!2^M$ equivalent points.

Furthermore, there will typically be multiple inequivalent stationary points and in particular multiple inequivalent minima. A minimum that corresponds to the smallest value of the error function for any weight vector is said to be a *global minimum*. Any other minima corresponding to higher values of the error function are said to be *local minima*. For a successful application of neural networks, it may not be necessary to find the global minimum (and in general it will not be known whether the global minimum has been found) but it may be necessary to compare several local minima in order to find a sufficiently good solution.

Because there is clearly no hope of finding an analytical solution to the equation $\nabla E(\mathbf{w}) = 0$ we resort to iterative numerical procedures. The optimization of continuous nonlinear functions is a widely studied problem and there exists an extensive literature on how to solve it efficiently. Most techniques involve choosing some initial value $\mathbf{w}^{(0)}$ for the weight vector and then moving through weight space in a succession of steps of the form

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \quad (5.27)$$

where τ labels the iteration step. Different algorithms involve different choices for the weight vector update $\Delta\mathbf{w}^{(\tau)}$. Many algorithms make use of gradient information and therefore require that, after each update, the value of $\nabla E(\mathbf{w})$ is evaluated at the new weight vector $\mathbf{w}^{(\tau+1)}$. In order to understand the importance of gradient information, it is useful to consider a local approximation to the error function based on a Taylor expansion.

5.2.2 Local quadratic approximation

Insight into the optimization problem, and into the various techniques for solving it, can be obtained by considering a local quadratic approximation to the error function.

Consider the Taylor expansion of $E(\mathbf{w})$ around some point $\hat{\mathbf{w}}$ in weight space

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^T \mathbf{b} + \frac{1}{2} (\mathbf{w} - \hat{\mathbf{w}})^T \mathbf{H} (\mathbf{w} - \hat{\mathbf{w}}) \quad (5.28)$$

Section 5.1.1

where cubic and higher terms have been omitted. Here \mathbf{b} is defined to be the gradient of E evaluated at $\hat{\mathbf{w}}$

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w}=\hat{\mathbf{w}}} \quad (5.29)$$

and the Hessian matrix $\mathbf{H} = \nabla \nabla E$ has elements

$$(\mathbf{H})_{ij} \equiv \left. \frac{\partial E}{\partial w_i \partial w_j} \right|_{\mathbf{w}=\hat{\mathbf{w}}}. \quad (5.30)$$

From (5.28), the corresponding local approximation to the gradient is given by

$$\nabla E \simeq \mathbf{b} + \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}}). \quad (5.31)$$

For points \mathbf{w} that are sufficiently close to $\hat{\mathbf{w}}$, these expressions will give reasonable approximations for the error and its gradient.

Consider the particular case of a local quadratic approximation around a point \mathbf{w}^* that is a minimum of the error function. In this case there is no linear term, because $\nabla E = 0$ at \mathbf{w}^* , and (5.28) becomes

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*) \quad (5.32)$$

where the Hessian \mathbf{H} is evaluated at \mathbf{w}^* . In order to interpret this geometrically, consider the eigenvalue equation for the Hessian matrix

$$\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i \quad (5.33)$$

where the eigenvectors \mathbf{u}_i form a complete orthonormal set (Appendix C) so that

$$\mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}. \quad (5.34)$$

We now expand $(\mathbf{w} - \mathbf{w}^*)$ as a linear combination of the eigenvectors in the form

$$\mathbf{w} - \mathbf{w}^* = \sum_i \alpha_i \mathbf{u}_i. \quad (5.35)$$

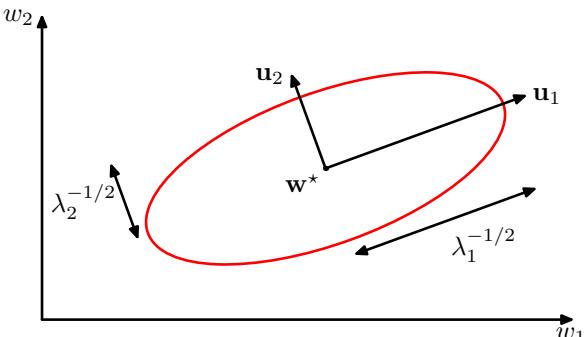
This can be regarded as a transformation of the coordinate system in which the origin is translated to the point \mathbf{w}^* , and the axes are rotated to align with the eigenvectors (through the orthogonal matrix whose columns are the \mathbf{u}_i), and is discussed in more detail in Appendix C. Substituting (5.35) into (5.32), and using (5.33) and (5.34), allows the error function to be written in the form

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2} \sum_i \lambda_i \alpha_i^2. \quad (5.36)$$

A matrix \mathbf{H} is said to be *positive definite* if, and only if,

$$\mathbf{v}^T \mathbf{H} \mathbf{v} > 0 \quad \text{for all } \mathbf{v}. \quad (5.37)$$

Figure 5.6 In the neighbourhood of a minimum w^* , the error function can be approximated by a quadratic. Contours of constant error are then ellipses whose axes are aligned with the eigenvectors u_i of the Hessian matrix, with lengths that are inversely proportional to the square roots of the corresponding eigenvalues λ_i .



Because the eigenvectors $\{u_i\}$ form a complete set, an arbitrary vector v can be written in the form

$$v = \sum_i c_i u_i. \quad (5.38)$$

From (5.33) and (5.34), we then have

$$v^T H v = \sum_i c_i^2 \lambda_i \quad (5.39)$$

Exercise 5.10

Exercise 5.11

and so H will be positive definite if, and only if, all of its eigenvalues are positive. In the new coordinate system, whose basis vectors are given by the eigenvectors $\{u_i\}$, the contours of constant E are ellipses centred on the origin, as illustrated in Figure 5.6. For a one-dimensional weight space, a stationary point w^* will be a minimum if

$$\left. \frac{\partial^2 E}{\partial w^2} \right|_{w^*} > 0. \quad (5.40)$$

Exercise 5.12

The corresponding result in D -dimensions is that the Hessian matrix, evaluated at w^* , should be positive definite.

5.2.3 Use of gradient information

As we shall see in Section 5.3, it is possible to evaluate the gradient of an error function efficiently by means of the backpropagation procedure. The use of this gradient information can lead to significant improvements in the speed with which the minima of the error function can be located. We can see why this is so, as follows.

In the quadratic approximation to the error function, given in (5.28), the error surface is specified by the quantities b and H , which contain a total of $W(W + 3)/2$ independent elements (because the matrix H is symmetric), where W is the dimensionality of w (i.e., the total number of adaptive parameters in the network). The location of the minimum of this quadratic approximation therefore depends on $O(W^2)$ parameters, and we should not expect to be able to locate the minimum until we have gathered $O(W^2)$ independent pieces of information. If we do not make use of gradient information, we would expect to have to perform $O(W^2)$ function

Exercise 5.13

evaluations, each of which would require $O(W)$ steps. Thus, the computational effort needed to find the minimum using such an approach would be $O(W^3)$.

Now compare this with an algorithm that makes use of the gradient information. Because each evaluation of ∇E brings W items of information, we might hope to find the minimum of the function in $O(W)$ gradient evaluations. As we shall see, by using error backpropagation, each such evaluation takes only $O(W)$ steps and so the minimum can now be found in $O(W^2)$ steps. For this reason, the use of gradient information forms the basis of practical algorithms for training neural networks.

5.2.4 Gradient descent optimization

The simplest approach to using gradient information is to choose the weight update in (5.27) to comprise a small step in the direction of the negative gradient, so that

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \quad (5.41)$$

where the parameter $\eta > 0$ is known as the *learning rate*. After each such update, the gradient is re-evaluated for the new weight vector and the process repeated. Note that the error function is defined with respect to a training set, and so each step requires that the entire training set be processed in order to evaluate ∇E . Techniques that use the whole data set at once are called *batch* methods. At each step the weight vector is moved in the direction of the greatest rate of decrease of the error function, and so this approach is known as *gradient descent* or *steepest descent*. Although such an approach might intuitively seem reasonable, in fact it turns out to be a poor algorithm, for reasons discussed in Bishop and Nabney (2008).

For batch optimization, there are more efficient methods, such as *conjugate gradients* and *quasi-Newton* methods, which are much more robust and much faster than simple gradient descent (Gill *et al.*, 1981; Fletcher, 1987; Nocedal and Wright, 1999). Unlike gradient descent, these algorithms have the property that the error function always decreases at each iteration unless the weight vector has arrived at a local or global minimum.

In order to find a sufficiently good minimum, it may be necessary to run a gradient-based algorithm multiple times, each time using a different randomly chosen starting point, and comparing the resulting performance on an independent validation set.

There is, however, an on-line version of gradient descent that has proved useful in practice for training neural networks on large data sets (Le Cun *et al.*, 1989). Error functions based on maximum likelihood for a set of independent observations comprise a sum of terms, one for each data point

$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w}). \quad (5.42)$$

On-line gradient descent, also known as *sequential gradient descent* or *stochastic gradient descent*, makes an update to the weight vector based on one data point at a time, so that

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}). \quad (5.43)$$

This update is repeated by cycling through the data either in sequence or by selecting points at random with replacement. There are of course intermediate scenarios in which the updates are based on batches of data points.

One advantage of on-line methods compared to batch methods is that the former handle redundancy in the data much more efficiently. To see, this consider an extreme example in which we take a data set and double its size by duplicating every data point. Note that this simply multiplies the error function by a factor of 2 and so is equivalent to using the original error function. Batch methods will require double the computational effort to evaluate the batch error function gradient, whereas on-line methods will be unaffected. Another property of on-line gradient descent is the possibility of escaping from local minima, since a stationary point with respect to the error function for the whole data set will generally not be a stationary point for each data point individually.

Nonlinear optimization algorithms, and their practical application to neural network training, are discussed in detail in Bishop and Nabney (2008).

5.3. Error Backpropagation

Our goal in this section is to find an efficient technique for evaluating the gradient of an error function $E(\mathbf{w})$ for a feed-forward neural network. We shall see that this can be achieved using a local message passing scheme in which information is sent alternately forwards and backwards through the network and is known as *error backpropagation*, or sometimes simply as *backprop*.

It should be noted that the term backpropagation is used in the neural computing literature to mean a variety of different things. For instance, the multilayer perceptron architecture is sometimes called a backpropagation network. The term backpropagation is also used to describe the training of a multilayer perceptron using gradient descent applied to a sum-of-squares error function. In order to clarify the terminology, it is useful to consider the nature of the training process more carefully. Most training algorithms involve an iterative procedure for minimization of an error function, with adjustments to the weights being made in a sequence of steps. At each such step, we can distinguish between two distinct stages. In the first stage, the derivatives of the error function with respect to the weights must be evaluated. As we shall see, the important contribution of the backpropagation technique is in providing a computationally efficient method for evaluating such derivatives. Because it is at this stage that errors are propagated backwards through the network, we shall use the term backpropagation specifically to describe the evaluation of derivatives. In the second stage, the derivatives are then used to compute the adjustments to be made to the weights. The simplest such technique, and the one originally considered by Rumelhart *et al.* (1986), involves gradient descent. It is important to recognize that the two stages are distinct. Thus, the first stage, namely the propagation of errors backwards through the network in order to evaluate derivatives, can be applied to many other kinds of network and not just the multilayer perceptron. It can also be applied to error functions other than just the simple sum-of-squares, and to the eval-

uation of other derivatives such as the Jacobian and Hessian matrices, as we shall see later in this chapter. Similarly, the second stage of weight adjustment using the calculated derivatives can be tackled using a variety of optimization schemes, many of which are substantially more powerful than simple gradient descent.

5.3.1 Evaluation of error-function derivatives

We now derive the backpropagation algorithm for a general network having arbitrary feed-forward topology, arbitrary differentiable nonlinear activation functions, and a broad class of error function. The resulting formulae will then be illustrated using a simple layered network structure having a single layer of sigmoidal hidden units together with a sum-of-squares error.

Many error functions of practical interest, for instance those defined by maximum likelihood for a set of i.i.d. data, comprise a sum of terms, one for each data point in the training set, so that

$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w}). \quad (5.44)$$

Here we shall consider the problem of evaluating $\nabla E_n(\mathbf{w})$ for one such term in the error function. This may be used directly for sequential optimization, or the results can be accumulated over the training set in the case of batch methods.

Consider first a simple linear model in which the outputs y_k are linear combinations of the input variables x_i so that

$$y_k = \sum_i w_{ki} x_i \quad (5.45)$$

together with an error function that, for a particular input pattern n , takes the form

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \quad (5.46)$$

where $y_{nk} = y_k(\mathbf{x}_n, \mathbf{w})$. The gradient of this error function with respect to a weight w_{ji} is given by

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \quad (5.47)$$

which can be interpreted as a ‘local’ computation involving the product of an ‘error signal’ $y_{nj} - t_{nj}$ associated with the output end of the link w_{ji} and the variable x_{ni} associated with the input end of the link. In Section 4.3.2, we saw how a similar formula arises with the logistic sigmoid activation function together with the cross-entropy error function, and similarly for the softmax activation function together with its matching cross-entropy error function. We shall now see how this simple result extends to the more complex setting of multilayer feed-forward networks.

In a general feed-forward network, each unit computes a weighted sum of its inputs of the form

$$a_j = \sum_i w_{ji} z_i \quad (5.48)$$

where z_i is the activation of a unit, or input, that sends a connection to unit j , and w_{ji} is the weight associated with that connection. In Section 5.1, we saw that biases can be included in this sum by introducing an extra unit, or input, with activation fixed at +1. We therefore do not need to deal with biases explicitly. The sum in (5.48) is transformed by a nonlinear activation function $h(\cdot)$ to give the activation z_j of unit j in the form

$$z_j = h(a_j). \quad (5.49)$$

Note that one or more of the variables z_i in the sum in (5.48) could be an input, and similarly, the unit j in (5.49) could be an output.

For each pattern in the training set, we shall suppose that we have supplied the corresponding input vector to the network and calculated the activations of all of the hidden and output units in the network by successive application of (5.48) and (5.49). This process is often called *forward propagation* because it can be regarded as a forward flow of information through the network.

Now consider the evaluation of the derivative of E_n with respect to a weight w_{ji} . The outputs of the various units will depend on the particular input pattern n . However, in order to keep the notation uncluttered, we shall omit the subscript n from the network variables. First we note that E_n depends on the weight w_{ji} only via the summed input a_j to unit j . We can therefore apply the chain rule for partial derivatives to give

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}. \quad (5.50)$$

We now introduce a useful notation

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \quad (5.51)$$

where the δ 's are often referred to as *errors* for reasons we shall see shortly. Using (5.48), we can write

$$\frac{\partial a_j}{\partial w_{ji}} = z_i. \quad (5.52)$$

Substituting (5.51) and (5.52) into (5.50), we then obtain

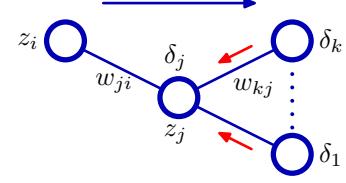
$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i. \quad (5.53)$$

Equation (5.53) tells us that the required derivative is obtained simply by multiplying the value of δ for the unit at the output end of the weight by the value of z for the unit at the input end of the weight (where $z = 1$ in the case of a bias). Note that this takes the same form as for the simple linear model considered at the start of this section. Thus, in order to evaluate the derivatives, we need only to calculate the value of δ_j for each hidden and output unit in the network, and then apply (5.53).

As we have seen already, for the output units, we have

$$\delta_k = y_k - t_k \quad (5.54)$$

Figure 5.7 Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ 's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.



provided we are using the canonical link as the output-unit activation function. To evaluate the δ 's for hidden units, we again make use of the chain rule for partial derivatives,

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \quad (5.55)$$

where the sum runs over all units k to which unit j sends connections. The arrangement of units and weights is illustrated in Figure 5.7. Note that the units labelled k could include other hidden units and/or output units. In writing down (5.55), we are making use of the fact that variations in a_j give rise to variations in the error function only through variations in the variables a_k . If we now substitute the definition of δ given by (5.51) into (5.55), and make use of (5.48) and (5.49), we obtain the following *backpropagation* formula

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \quad (5.56)$$

which tells us that the value of δ for a particular hidden unit can be obtained by propagating the δ 's backwards from units higher up in the network, as illustrated in Figure 5.7. Note that the summation in (5.56) is taken over the first index on w_{kj} (corresponding to backward propagation of information through the network), whereas in the forward propagation equation (5.10) it is taken over the second index. Because we already know the values of the δ 's for the output units, it follows that by recursively applying (5.56) we can evaluate the δ 's for all of the hidden units in a feed-forward network, regardless of its topology.

The backpropagation procedure can therefore be summarized as follows.

Error Backpropagation

1. Apply an input vector \mathbf{x}_n to the network and forward propagate through the network using (5.48) and (5.49) to find the activations of all the hidden and output units.
2. Evaluate the δ_k for all the output units using (5.54).
3. Backpropagate the δ 's using (5.56) to obtain δ_j for each hidden unit in the network.
4. Use (5.53) to evaluate the required derivatives.

For batch methods, the derivative of the total error E can then be obtained by repeating the above steps for each pattern in the training set and then summing over all patterns:

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}}. \quad (5.57)$$

In the above derivation we have implicitly assumed that each hidden or output unit in the network has the same activation function $h(\cdot)$. The derivation is easily generalized, however, to allow different units to have individual activation functions, simply by keeping track of which form of $h(\cdot)$ goes with which unit.

5.3.2 A simple example

The above derivation of the backpropagation procedure allowed for general forms for the error function, the activation functions, and the network topology. In order to illustrate the application of this algorithm, we shall consider a particular example. This is chosen both for its simplicity and for its practical importance, because many applications of neural networks reported in the literature make use of this type of network. Specifically, we shall consider a two-layer network of the form illustrated in Figure 5.1, together with a sum-of-squares error, in which the output units have linear activation functions, so that $y_k = a_k$, while the hidden units have logistic sigmoid activation functions given by

$$h(a) \equiv \tanh(a) \quad (5.58)$$

where

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}. \quad (5.59)$$

A useful feature of this function is that its derivative can be expressed in a particularly simple form:

$$h'(a) = 1 - h(a)^2. \quad (5.60)$$

We also consider a standard sum-of-squares error function, so that for pattern n the error is given by

$$E_n = \frac{1}{2} \sum_{k=1}^K (y_k - t_k)^2 \quad (5.61)$$

where y_k is the activation of output unit k , and t_k is the corresponding target, for a particular input pattern \mathbf{x}_n .

For each pattern in the training set in turn, we first perform a forward propagation using

$$a_j = \sum_{i=0}^D w_{ji}^{(1)} x_i \quad (5.62)$$

$$z_j = \tanh(a_j) \quad (5.63)$$

$$y_k = \sum_{j=0}^M w_{kj}^{(2)} z_j. \quad (5.64)$$

Next we compute the δ 's for each output unit using

$$\delta_k = y_k - t_k. \quad (5.65)$$

Then we backpropagate these to obtain δ s for the hidden units using

$$\delta_j = (1 - z_j^2) \sum_{k=1}^K w_{kj} \delta_k. \quad (5.66)$$

Finally, the derivatives with respect to the first-layer and second-layer weights are given by

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \quad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j. \quad (5.67)$$

5.3.3 Efficiency of backpropagation

One of the most important aspects of backpropagation is its computational efficiency. To understand this, let us examine how the number of computer operations required to evaluate the derivatives of the error function scales with the total number W of weights and biases in the network. A single evaluation of the error function (for a given input pattern) would require $O(W)$ operations, for sufficiently large W . This follows from the fact that, except for a network with very sparse connections, the number of weights is typically much greater than the number of units, and so the bulk of the computational effort in forward propagation is concerned with evaluating the sums in (5.48), with the evaluation of the activation functions representing a small overhead. Each term in the sum in (5.48) requires one multiplication and one addition, leading to an overall computational cost that is $O(W)$.

An alternative approach to backpropagation for computing the derivatives of the error function is to use finite differences. This can be done by perturbing each weight in turn, and approximating the derivatives by the expression

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji})}{\epsilon} + O(\epsilon) \quad (5.68)$$

where $\epsilon \ll 1$. In a software simulation, the accuracy of the approximation to the derivatives can be improved by making ϵ smaller, until numerical roundoff problems arise. The accuracy of the finite differences method can be improved significantly by using symmetrical *central differences* of the form

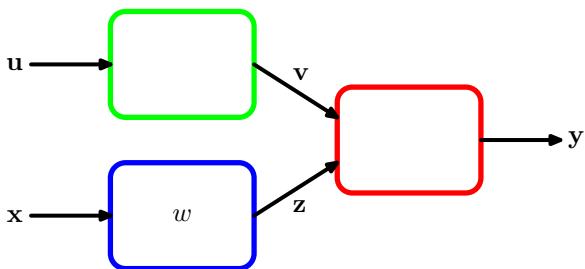
$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji} - \epsilon)}{2\epsilon} + O(\epsilon^2). \quad (5.69)$$

Exercise 5.14

In this case, the $O(\epsilon)$ corrections cancel, as can be verified by Taylor expansion on the right-hand side of (5.69), and so the residual corrections are $O(\epsilon^2)$. The number of computational steps is, however, roughly doubled compared with (5.68).

The main problem with numerical differentiation is that the highly desirable $O(W)$ scaling has been lost. Each forward propagation requires $O(W)$ steps, and

Figure 5.8 Illustration of a modular pattern recognition system in which the Jacobian matrix can be used to backpropagate error signals from the outputs through to earlier modules in the system.



there are W weights in the network each of which must be perturbed individually, so that the overall scaling is $O(W^2)$.

However, numerical differentiation plays an important role in practice, because a comparison of the derivatives calculated by backpropagation with those obtained using central differences provides a powerful check on the correctness of any software implementation of the backpropagation algorithm. When training networks in practice, derivatives should be evaluated using backpropagation, because this gives the greatest accuracy and numerical efficiency. However, the results should be compared with numerical differentiation using (5.69) for some test cases in order to check the correctness of the implementation.

5.3.4 The Jacobian matrix

We have seen how the derivatives of an error function with respect to the weights can be obtained by the propagation of errors backwards through the network. The technique of backpropagation can also be applied to the calculation of other derivatives. Here we consider the evaluation of the *Jacobian* matrix, whose elements are given by the derivatives of the network outputs with respect to the inputs

$$J_{ki} \equiv \frac{\partial y_k}{\partial x_i} \quad (5.70)$$

where each such derivative is evaluated with all other inputs held fixed. Jacobian matrices play a useful role in systems built from a number of distinct modules, as illustrated in Figure 5.8. Each module can comprise a fixed or adaptive function, which can be linear or nonlinear, so long as it is differentiable. Suppose we wish to minimize an error function E with respect to the parameter w in Figure 5.8. The derivative of the error function is given by

$$\frac{\partial E}{\partial w} = \sum_{k,j} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_j} \frac{\partial z_j}{\partial w} \quad (5.71)$$

in which the Jacobian matrix for the red module in Figure 5.8 appears in the middle term.

Because the Jacobian matrix provides a measure of the local sensitivity of the outputs to changes in each of the input variables, it also allows any known errors Δx_i

associated with the inputs to be propagated through the trained network in order to estimate their contribution Δy_k to the errors at the outputs, through the relation

$$\Delta y_k \simeq \sum_i \frac{\partial y_k}{\partial x_i} \Delta x_i \quad (5.72)$$

which is valid provided the $|\Delta x_i|$ are small. In general, the network mapping represented by a trained neural network will be nonlinear, and so the elements of the Jacobian matrix will not be constants but will depend on the particular input vector used. Thus (5.72) is valid only for small perturbations of the inputs, and the Jacobian itself must be re-evaluated for each new input vector.

The Jacobian matrix can be evaluated using a backpropagation procedure that is similar to the one derived earlier for evaluating the derivatives of an error function with respect to the weights. We start by writing the element J_{ki} in the form

$$\begin{aligned} J_{ki} = \frac{\partial y_k}{\partial x_i} &= \sum_j \frac{\partial y_k}{\partial a_j} \frac{\partial a_j}{\partial x_i} \\ &= \sum_j w_{ji} \frac{\partial y_k}{\partial a_j} \end{aligned} \quad (5.73)$$

where we have made use of (5.48). The sum in (5.73) runs over all units j to which the input unit i sends connections (for example, over all units in the first hidden layer in the layered topology considered earlier). We now write down a recursive backpropagation formula to determine the derivatives $\partial y_k / \partial a_j$

$$\begin{aligned} \frac{\partial y_k}{\partial a_j} &= \sum_l \frac{\partial y_k}{\partial a_l} \frac{\partial a_l}{\partial a_j} \\ &= h'(a_j) \sum_l w_{lj} \frac{\partial y_k}{\partial a_l} \end{aligned} \quad (5.74)$$

where the sum runs over all units l to which unit j sends connections (corresponding to the first index of w_{lj}). Again, we have made use of (5.48) and (5.49). This backpropagation starts at the output units for which the required derivatives can be found directly from the functional form of the output-unit activation function. For instance, if we have individual sigmoidal activation functions at each output unit, then

$$\frac{\partial y_k}{\partial a_j} = \delta_{kj} \sigma'(a_j) \quad (5.75)$$

whereas for softmax outputs we have

$$\frac{\partial y_k}{\partial a_j} = \delta_{kj} y_k - y_k y_j. \quad (5.76)$$

We can summarize the procedure for evaluating the Jacobian matrix as follows. Apply the input vector corresponding to the point in input space at which the Jacobian matrix is to be found, and forward propagate in the usual way to obtain the

activations of all of the hidden and output units in the network. Next, for each row k of the Jacobian matrix, corresponding to the output unit k , backpropagate using the recursive relation (5.74), starting with (5.75) or (5.76), for all of the hidden units in the network. Finally, use (5.73) to do the backpropagation to the inputs. The Jacobian can also be evaluated using an alternative *forward* propagation formalism, which can be derived in an analogous way to the backpropagation approach given here.

Again, the implementation of such algorithms can be checked by using numerical differentiation in the form

$$\frac{\partial y_k}{\partial x_i} = \frac{y_k(x_i + \epsilon) - y_k(x_i - \epsilon)}{2\epsilon} + O(\epsilon^2) \quad (5.77)$$

which involves $2D$ forward propagations for a network having D inputs.

5.4. The Hessian Matrix

We have shown how the technique of backpropagation can be used to obtain the first derivatives of an error function with respect to the weights in the network. Backpropagation can also be used to evaluate the second derivatives of the error, given by

$$\frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}}. \quad (5.78)$$

Note that it is sometimes convenient to consider all of the weight and bias parameters as elements w_i of a single vector, denoted \mathbf{w} , in which case the second derivatives form the elements H_{ij} of the *Hessian* matrix \mathbf{H} , where $i, j \in \{1, \dots, W\}$ and W is the total number of weights and biases. The Hessian plays an important role in many aspects of neural computing, including the following:

1. Several nonlinear optimization algorithms used for training neural networks are based on considerations of the second-order properties of the error surface, which are controlled by the Hessian matrix (Bishop and Nabney, 2008).
2. The Hessian forms the basis of a fast procedure for re-training a feed-forward network following a small change in the training data (Bishop, 1991).
3. The inverse of the Hessian has been used to identify the least significant weights in a network as part of network ‘pruning’ algorithms (Le Cun *et al.*, 1990).
4. The Hessian plays a central role in the Laplace approximation for a Bayesian neural network (see Section 5.7). Its inverse is used to determine the predictive distribution for a trained network, its eigenvalues determine the values of hyperparameters, and its determinant is used to evaluate the model evidence.

Various approximation schemes have been used to evaluate the Hessian matrix for a neural network. However, the Hessian can also be calculated exactly using an extension of the backpropagation technique.

Exercise 5.15

An important consideration for many applications of the Hessian is the efficiency with which it can be evaluated. If there are W parameters (weights and biases) in the network, then the Hessian matrix has dimensions $W \times W$ and so the computational effort needed to evaluate the Hessian will scale like $O(W^2)$ for each pattern in the data set. As we shall see, there are efficient methods for evaluating the Hessian whose scaling is indeed $O(W^2)$.

5.4.1 Diagonal approximation

Some of the applications for the Hessian matrix discussed above require the inverse of the Hessian, rather than the Hessian itself. For this reason, there has been some interest in using a diagonal approximation to the Hessian, in other words one that simply replaces the off-diagonal elements with zeros, because its inverse is trivial to evaluate. Again, we shall consider an error function that consists of a sum of terms, one for each pattern in the data set, so that $E = \sum_n E_n$. The Hessian can then be obtained by considering one pattern at a time, and then summing the results over all patterns. From (5.48), the diagonal elements of the Hessian, for pattern n , can be written

$$\frac{\partial^2 E_n}{\partial w_{ji}^2} = \frac{\partial^2 E_n}{\partial a_j^2} z_i^2. \quad (5.79)$$

Using (5.48) and (5.49), the second derivatives on the right-hand side of (5.79) can be found recursively using the chain rule of differential calculus to give a backpropagation equation of the form

$$\frac{\partial^2 E_n}{\partial a_j^2} = h'(a_j)^2 \sum_k \sum_{k'} w_{kj} w_{k'j} \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} + h''(a_j) \sum_k w_{kj} \frac{\partial E_n}{\partial a_k}. \quad (5.80)$$

If we now neglect off-diagonal elements in the second-derivative terms, we obtain (Becker and Le Cun, 1989; Le Cun *et al.*, 1990)

$$\frac{\partial^2 E_n}{\partial a_j^2} = h'(a_j)^2 \sum_k w_{kj}^2 \frac{\partial^2 E_n}{\partial a_k^2} + h''(a_j) \sum_k w_{kj} \frac{\partial E_n}{\partial a_k}. \quad (5.81)$$

Note that the number of computational steps required to evaluate this approximation is $O(W)$, where W is the total number of weight and bias parameters in the network, compared with $O(W^2)$ for the full Hessian.

Ricotti *et al.* (1988) also used the diagonal approximation to the Hessian, but they retained all terms in the evaluation of $\partial^2 E_n / \partial a_j^2$ and so obtained exact expressions for the diagonal terms. Note that this no longer has $O(W)$ scaling. The major problem with diagonal approximations, however, is that in practice the Hessian is typically found to be strongly nondiagonal, and so these approximations, which are driven mainly by computational convenience, must be treated with care.

5.4.2 Outer product approximation

When neural networks are applied to regression problems, it is common to use a sum-of-squares error function of the form

$$E = \frac{1}{2} \sum_{n=1}^N (y_n - t_n)^2 \quad (5.82)$$

where we have considered the case of a single output in order to keep the notation simple (the extension to several outputs is straightforward). We can then write the Hessian matrix in the form

$$\mathbf{H} = \nabla \nabla E = \sum_{n=1}^N \nabla y_n \nabla y_n + \sum_{n=1}^N (y_n - t_n) \nabla \nabla y_n. \quad (5.83)$$

If the network has been trained on the data set, and its outputs y_n happen to be very close to the target values t_n , then the second term in (5.83) will be small and can be neglected. More generally, however, it may be appropriate to neglect this term by the following argument. Recall from Section 1.5.5 that the optimal function that minimizes a sum-of-squares loss is the conditional average of the target data. The quantity $(y_n - t_n)$ is then a random variable with zero mean. If we assume that its value is uncorrelated with the value of the second derivative term on the right-hand side of (5.83), then the whole term will average to zero in the summation over n .

By neglecting the second term in (5.83), we arrive at the *Levenberg–Marquardt* approximation or *outer product* approximation (because the Hessian matrix is built up from a sum of outer products of vectors), given by

$$\mathbf{H} \simeq \sum_{n=1}^N \mathbf{b}_n \mathbf{b}_n^T \quad (5.84)$$

where $\mathbf{b}_n = \nabla y_n = \nabla a_n$ because the activation function for the output units is simply the identity. Evaluation of the outer product approximation for the Hessian is straightforward as it only involves first derivatives of the error function, which can be evaluated efficiently in $O(W)$ steps using standard backpropagation. The elements of the matrix can then be found in $O(W^2)$ steps by simple multiplication. It is important to emphasize that this approximation is only likely to be valid for a network that has been trained appropriately, and that for a general network mapping the second derivative terms on the right-hand side of (5.83) will typically not be negligible.

In the case of the cross-entropy error function for a network with logistic sigmoid output-unit activation functions, the corresponding approximation is given by

$$\mathbf{H} \simeq \sum_{n=1}^N y_n(1-y_n) \mathbf{b}_n \mathbf{b}_n^T. \quad (5.85)$$

An analogous result can be obtained for multiclass networks having softmax output-unit activation functions.

Exercise 5.16

Exercise 5.17

Exercise 5.19

Exercise 5.20

5.4.3 Inverse Hessian

We can use the outer-product approximation to develop a computationally efficient procedure for approximating the inverse of the Hessian (Hassibi and Stork, 1993). First we write the outer-product approximation in matrix notation as

$$\mathbf{H}_N = \sum_{n=1}^N \mathbf{b}_n \mathbf{b}_n^T \quad (5.86)$$

where $\mathbf{b}_n \equiv \nabla_{\mathbf{w}} a_n$ is the contribution to the gradient of the output unit activation arising from data point n . We now derive a sequential procedure for building up the Hessian by including data points one at a time. Suppose we have already obtained the inverse Hessian using the first L data points. By separating off the contribution from data point $L + 1$, we obtain

$$\mathbf{H}_{L+1} = \mathbf{H}_L + \mathbf{b}_{L+1} \mathbf{b}_{L+1}^T. \quad (5.87)$$

In order to evaluate the inverse of the Hessian, we now consider the matrix identity

$$(\mathbf{M} + \mathbf{v}\mathbf{v}^T)^{-1} = \mathbf{M}^{-1} - \frac{(\mathbf{M}^{-1}\mathbf{v})(\mathbf{v}^T\mathbf{M}^{-1})}{1 + \mathbf{v}^T\mathbf{M}^{-1}\mathbf{v}} \quad (5.88)$$

where \mathbf{I} is the unit matrix, which is simply a special case of the Woodbury identity (C.7). If we now identify \mathbf{H}_L with \mathbf{M} and \mathbf{b}_{L+1} with \mathbf{v} , we obtain

$$\mathbf{H}_{L+1}^{-1} = \mathbf{H}_L^{-1} - \frac{\mathbf{H}_L^{-1}\mathbf{b}_{L+1}\mathbf{b}_{L+1}^T\mathbf{H}_L^{-1}}{1 + \mathbf{b}_{L+1}^T\mathbf{H}_L^{-1}\mathbf{b}_{L+1}}. \quad (5.89)$$

In this way, data points are sequentially absorbed until $L + 1 = N$ and the whole data set has been processed. This result therefore represents a procedure for evaluating the inverse of the Hessian using a single pass through the data set. The initial matrix \mathbf{H}_0 is chosen to be $\alpha\mathbf{I}$, where α is a small quantity, so that the algorithm actually finds the inverse of $\mathbf{H} + \alpha\mathbf{I}$. The results are not particularly sensitive to the precise value of α . Extension of this algorithm to networks having more than one output is straightforward.

We note here that the Hessian matrix can sometimes be calculated indirectly as part of the network training algorithm. In particular, quasi-Newton nonlinear optimization algorithms gradually build up an approximation to the inverse of the Hessian during training. Such algorithms are discussed in detail in Bishop and Nabney (2008).

5.4.4 Finite differences

As in the case of the first derivatives of the error function, we can find the second derivatives by using finite differences, with accuracy limited by numerical precision. If we perturb each possible pair of weights in turn, we obtain

$$\begin{aligned} \frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}} &= \frac{1}{4\epsilon^2} \{E(w_{ji} + \epsilon, w_{lk} + \epsilon) - E(w_{ji} + \epsilon, w_{lk} - \epsilon) \\ &\quad - E(w_{ji} - \epsilon, w_{lk} + \epsilon) + E(w_{ji} - \epsilon, w_{lk} - \epsilon)\} + O(\epsilon^2). \end{aligned} \quad (5.90)$$

Exercise 5.21

Again, by using a symmetrical central differences formulation, we ensure that the residual errors are $O(\epsilon^2)$ rather than $O(\epsilon)$. Because there are W^2 elements in the Hessian matrix, and because the evaluation of each element requires four forward propagations each needing $O(W)$ operations (per pattern), we see that this approach will require $O(W^3)$ operations to evaluate the complete Hessian. It therefore has poor scaling properties, although in practice it is very useful as a check on the software implementation of backpropagation methods.

A more efficient version of numerical differentiation can be found by applying central differences to the first derivatives of the error function, which are themselves calculated using backpropagation. This gives

$$\frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}} = \frac{1}{2\epsilon} \left\{ \frac{\partial E}{\partial w_{ji}}(w_{lk} + \epsilon) - \frac{\partial E}{\partial w_{ji}}(w_{lk} - \epsilon) \right\} + O(\epsilon^2). \quad (5.91)$$

Because there are now only W weights to be perturbed, and because the gradients can be evaluated in $O(W)$ steps, we see that this method gives the Hessian in $O(W^2)$ operations.

5.4.5 Exact evaluation of the Hessian

So far, we have considered various approximation schemes for evaluating the Hessian matrix or its inverse. The Hessian can also be evaluated exactly, for a network of arbitrary feed-forward topology, using extension of the technique of back-propagation used to evaluate first derivatives, which shares many of its desirable features including computational efficiency (Bishop, 1991; Bishop, 1992). It can be applied to any differentiable error function that can be expressed as a function of the network outputs and to networks having arbitrary differentiable activation functions. The number of computational steps needed to evaluate the Hessian scales like $O(W^2)$. Similar algorithms have also been considered by Buntine and Weigend (1993).

Here we consider the specific case of a network having two layers of weights, for which the required equations are easily derived. We shall use indices i and i' to denote inputs, indices j and j' to denote hidden units, and indices k and k' to denote outputs. We first define

$$\delta_k = \frac{\partial E_n}{\partial a_k}, \quad M_{kk'} \equiv \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} \quad (5.92)$$

where E_n is the contribution to the error from data point n . The Hessian matrix for this network can then be considered in three separate blocks as follows.

1. Both weights in the second layer:

$$\frac{\partial^2 E_n}{\partial w_{kj}^{(2)} \partial w_{k'j'}^{(2)}} = z_j z_{j'} M_{kk'}. \quad (5.93)$$

Exercise 5.22

2. Both weights in the first layer:

$$\begin{aligned} \frac{\partial^2 E_n}{\partial w_{ji}^{(1)} \partial w_{j'i'}^{(1)}} &= x_i x_{i'} h''(a_{j'}) I_{jj'} \sum_k w_{kj'}^{(2)} \delta_k \\ &+ x_i x_{i'} h'(a_{j'}) h'(a_j) \sum_k \sum_{k'} w_{k'j'}^{(2)} w_{kj}^{(2)} M_{kk'}. \end{aligned} \quad (5.94)$$

3. One weight in each layer:

$$\frac{\partial^2 E_n}{\partial w_{ji}^{(1)} \partial w_{kj'}^{(2)}} = x_i h'(a_{j'}) \left\{ \delta_k I_{jj'} + z_j \sum_{k'} w_{k'j'}^{(2)} H_{kk'} \right\}. \quad (5.95)$$

Here $I_{jj'}$ is the j, j' element of the identity matrix. If one or both of the weights is a bias term, then the corresponding expressions are obtained simply by setting the appropriate activation(s) to 1. Inclusion of skip-layer connections is straightforward.

Exercise 5.23

5.4.6 Fast multiplication by the Hessian

For many applications of the Hessian, the quantity of interest is not the Hessian matrix \mathbf{H} itself but the product of \mathbf{H} with some vector \mathbf{v} . We have seen that the evaluation of the Hessian takes $O(W^2)$ operations, and it also requires storage that is $O(W^2)$. The vector $\mathbf{v}^T \mathbf{H}$ that we wish to calculate, however, has only W elements, so instead of computing the Hessian as an intermediate step, we can instead try to find an efficient approach to evaluating $\mathbf{v}^T \mathbf{H}$ directly in a way that requires only $O(W)$ operations.

To do this, we first note that

$$\mathbf{v}^T \mathbf{H} = \mathbf{v}^T \nabla(\nabla E) \quad (5.96)$$

where ∇ denotes the gradient operator in weight space. We can then write down the standard forward-propagation and backpropagation equations for the evaluation of ∇E and apply (5.96) to these equations to give a set of forward-propagation and backpropagation equations for the evaluation of $\mathbf{v}^T \nabla(\nabla E)$ (Møller, 1993; Pearlmutter, 1994). This corresponds to acting on the original forward-propagation and backpropagation equations with a differential operator $\mathbf{v}^T \nabla$. Pearlmutter (1994) used the notation $\mathcal{R}\{\cdot\}$ to denote the operator $\mathbf{v}^T \nabla$, and we shall follow this convention. The analysis is straightforward and makes use of the usual rules of differential calculus, together with the result

$$\mathcal{R}\{\mathbf{w}\} = \mathbf{v}. \quad (5.97)$$

The technique is best illustrated with a simple example, and again we choose a two-layer network of the form shown in Figure 5.1, with linear output units and a sum-of-squares error function. As before, we consider the contribution to the error function from one pattern in the data set. The required vector is then obtained as

usual by summing over the contributions from each of the patterns separately. For the two-layer network, the forward-propagation equations are given by

$$a_j = \sum_i w_{ji} x_i \quad (5.98)$$

$$z_j = h(a_j) \quad (5.99)$$

$$y_k = \sum_j w_{kj} z_j. \quad (5.100)$$

We now act on these equations using the $\mathcal{R}\{\cdot\}$ operator to obtain a set of forward propagation equations in the form

$$\mathcal{R}\{a_j\} = \sum_i v_{ji} x_i \quad (5.101)$$

$$\mathcal{R}\{z_j\} = h'(a_j) \mathcal{R}\{a_j\} \quad (5.102)$$

$$\mathcal{R}\{y_k\} = \sum_j w_{kj} \mathcal{R}\{z_j\} + \sum_j v_{kj} z_j \quad (5.103)$$

where v_{ji} is the element of the vector \mathbf{v} that corresponds to the weight w_{ji} . Quantities of the form $\mathcal{R}\{z_j\}$, $\mathcal{R}\{a_j\}$ and $\mathcal{R}\{y_k\}$ are to be regarded as new variables whose values are found using the above equations.

Because we are considering a sum-of-squares error function, we have the following standard backpropagation expressions:

$$\delta_k = y_k - t_k \quad (5.104)$$

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k. \quad (5.105)$$

Again, we act on these equations with the $\mathcal{R}\{\cdot\}$ operator to obtain a set of backpropagation equations in the form

$$\mathcal{R}\{\delta_k\} = \mathcal{R}\{y_k\} \quad (5.106)$$

$$\begin{aligned} \mathcal{R}\{\delta_j\} &= h''(a_j) \mathcal{R}\{a_j\} \sum_k w_{kj} \delta_k \\ &\quad + h'(a_j) \sum_k v_{kj} \delta_k + h'(a_j) \sum_k w_{kj} \mathcal{R}\{\delta_k\}. \end{aligned} \quad (5.107)$$

Finally, we have the usual equations for the first derivatives of the error

$$\frac{\partial E}{\partial w_{kj}} = \delta_k z_j \quad (5.108)$$

$$\frac{\partial E}{\partial w_{ji}} = \delta_j x_i \quad (5.109)$$

and acting on these with the $\mathcal{R}\{\cdot\}$ operator, we obtain expressions for the elements of the vector $\mathbf{v}^T \mathbf{H}$

$$\mathcal{R} \left\{ \frac{\partial E}{\partial w_{kj}} \right\} = \mathcal{R}\{\delta_k\} z_j + \delta_k \mathcal{R}\{z_j\} \quad (5.110)$$

$$\mathcal{R} \left\{ \frac{\partial E}{\partial w_{ji}} \right\} = x_i \mathcal{R}\{\delta_j\}. \quad (5.111)$$

The implementation of this algorithm involves the introduction of additional variables $\mathcal{R}\{a_j\}$, $\mathcal{R}\{z_j\}$ and $\mathcal{R}\{\delta_j\}$ for the hidden units and $\mathcal{R}\{\delta_k\}$ and $\mathcal{R}\{y_k\}$ for the output units. For each input pattern, the values of these quantities can be found using the above results, and the elements of $\mathbf{v}^T \mathbf{H}$ are then given by (5.110) and (5.111). An elegant aspect of this technique is that the equations for evaluating $\mathbf{v}^T \mathbf{H}$ mirror closely those for standard forward and backward propagation, and so the extension of existing software to compute this product is typically straightforward.

If desired, the technique can be used to evaluate the full Hessian matrix by choosing the vector \mathbf{v} to be given successively by a series of unit vectors of the form $(0, 0, \dots, 1, \dots, 0)$ each of which picks out one column of the Hessian. This leads to a formalism that is analytically equivalent to the backpropagation procedure of Bishop (1992), as described in Section 5.4.5, though with some loss of efficiency due to redundant calculations.

5.5. Regularization in Neural Networks

The number of input and outputs units in a neural network is generally determined by the dimensionality of the data set, whereas the number M of hidden units is a free parameter that can be adjusted to give the best predictive performance. Note that M controls the number of parameters (weights and biases) in the network, and so we might expect that in a maximum likelihood setting there will be an optimum value of M that gives the best generalization performance, corresponding to the optimum balance between under-fitting and over-fitting. Figure 5.9 shows an example of the effect of different values of M for the sinusoidal regression problem.

The generalization error, however, is not a simple function of M due to the presence of local minima in the error function, as illustrated in Figure 5.10. Here we see the effect of choosing multiple random initializations for the weight vector for a range of values of M . The overall best validation set performance in this case occurred for a particular solution having $M = 8$. In practice, one approach to choosing M is in fact to plot a graph of the kind shown in Figure 5.10 and then to choose the specific solution having the smallest validation set error.

There are, however, other ways to control the complexity of a neural network model in order to avoid over-fitting. From our discussion of polynomial curve fitting in Chapter 1, we see that an alternative approach is to choose a relatively large value for M and then to control complexity by the addition of a regularization term to the error function. The simplest regularizer is the quadratic, giving a regularized error

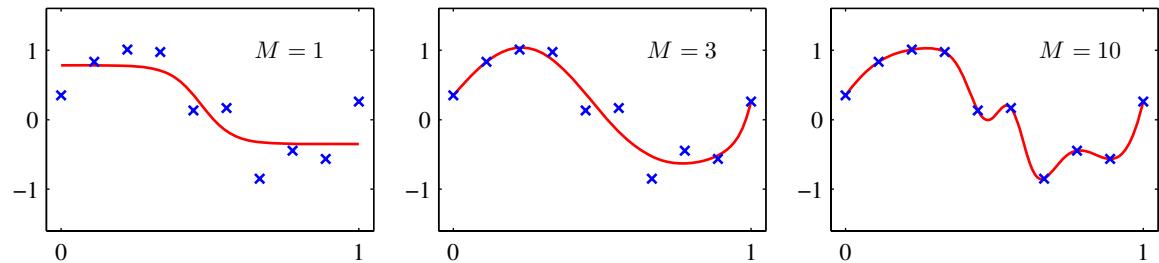


Figure 5.9 Examples of two-layer networks trained on 10 data points drawn from the sinusoidal data set. The graphs show the result of fitting networks having $M = 1, 3$ and 10 hidden units, respectively, by minimizing a sum-of-squares error function using a scaled conjugate-gradient algorithm.

of the form

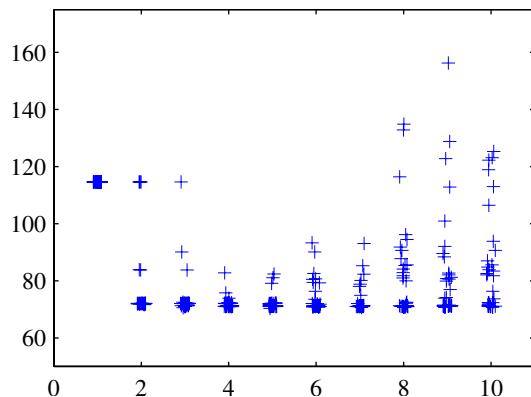
$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}. \quad (5.112)$$

This regularizer is also known as *weight decay* and has been discussed at length in Chapter 3. The effective model complexity is then determined by the choice of the regularization coefficient λ . As we have seen previously, this regularizer can be interpreted as the negative logarithm of a zero-mean Gaussian prior distribution over the weight vector \mathbf{w} .

5.5.1 Consistent Gaussian priors

One of the limitations of simple weight decay in the form (5.112) is that is inconsistent with certain scaling properties of network mappings. To illustrate this, consider a multilayer perceptron network having two layers of weights and linear output units, which performs a mapping from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$. The activations of the hidden units in the first hidden layer

Figure 5.10 Plot of the sum-of-squares test-set error for the polynomial data set versus the number of hidden units in the network, with 30 random starts for each network size, showing the effect of local minima. For each new start, the weight vector was initialized by sampling from an isotropic Gaussian distribution having a mean of zero and a variance of 10.



take the form

$$z_j = h \left(\sum_i w_{ji} x_i + w_{j0} \right) \quad (5.113)$$

while the activations of the output units are given by

$$y_k = \sum_j w_{kj} z_j + w_{k0}. \quad (5.114)$$

Suppose we perform a linear transformation of the input data of the form

$$x_i \rightarrow \tilde{x}_i = ax_i + b. \quad (5.115)$$

Then we can arrange for the mapping performed by the network to be unchanged by making a corresponding linear transformation of the weights and biases from the inputs to the units in the hidden layer of the form

$$w_{ji} \rightarrow \tilde{w}_{ji} = \frac{1}{a} w_{ji} \quad (5.116)$$

$$w_{j0} \rightarrow \tilde{w}_{j0} = w_{j0} - \frac{b}{a} \sum_i w_{ji}. \quad (5.117)$$

Similarly, a linear transformation of the output variables of the network of the form

$$y_k \rightarrow \tilde{y}_k = cy_k + d \quad (5.118)$$

can be achieved by making a transformation of the second-layer weights and biases using

$$w_{kj} \rightarrow \tilde{w}_{kj} = cw_{kj} \quad (5.119)$$

$$w_{k0} \rightarrow \tilde{w}_{k0} = cw_{k0} + d. \quad (5.120)$$

If we train one network using the original data and one network using data for which the input and/or target variables are transformed by one of the above linear transformations, then consistency requires that we should obtain equivalent networks that differ only by the linear transformation of the weights as given. Any regularizer should be consistent with this property, otherwise it arbitrarily favours one solution over another, equivalent one. Clearly, simple weight decay (5.112), that treats all weights and biases on an equal footing, does not satisfy this property.

We therefore look for a regularizer which is invariant under the linear transformations (5.116), (5.117), (5.119) and (5.120). These require that the regularizer should be invariant to re-scaling of the weights and to shifts of the biases. Such a regularizer is given by

$$\frac{\lambda_1}{2} \sum_{w \in \mathcal{W}_1} w^2 + \frac{\lambda_2}{2} \sum_{w \in \mathcal{W}_2} w^2 \quad (5.121)$$

where \mathcal{W}_1 denotes the set of weights in the first layer, \mathcal{W}_2 denotes the set of weights in the second layer, and biases are excluded from the summations. This regularizer

Exercise 5.24

will remain unchanged under the weight transformations provided the regularization parameters are re-scaled using $\lambda_1 \rightarrow a^{1/2}\lambda_1$ and $\lambda_2 \rightarrow c^{-1/2}\lambda_2$.

The regularizer (5.121) corresponds to a prior of the form

$$p(\mathbf{w}|\alpha_1, \alpha_2) \propto \exp\left(-\frac{\alpha_1}{2} \sum_{w \in \mathcal{W}_1} w^2 - \frac{\alpha_2}{2} \sum_{w \in \mathcal{W}_2} w^2\right). \quad (5.122)$$

Note that priors of this form are *improper* (they cannot be normalized) because the bias parameters are unconstrained. The use of improper priors can lead to difficulties in selecting regularization coefficients and in model comparison within the Bayesian framework, because the corresponding evidence is zero. It is therefore common to include separate priors for the biases (which then break shift invariance) having their own hyperparameters. We can illustrate the effect of the resulting four hyperparameters by drawing samples from the prior and plotting the corresponding network functions, as shown in Figure 5.11.

More generally, we can consider priors in which the weights are divided into any number of groups \mathcal{W}_k so that

$$p(\mathbf{w}) \propto \exp\left(-\frac{1}{2} \sum_k \alpha_k \|\mathbf{w}\|_k^2\right) \quad (5.123)$$

where

$$\|\mathbf{w}\|_k^2 = \sum_{j \in \mathcal{W}_k} w_j^2. \quad (5.124)$$

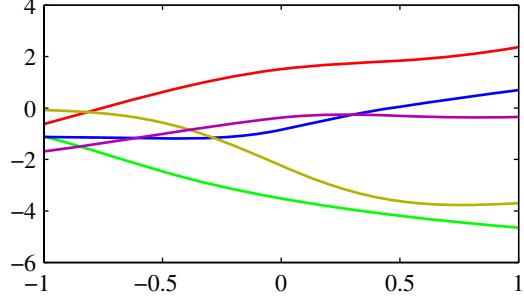
As a special case of this prior, if we choose the groups to correspond to the sets of weights associated with each of the input units, and we optimize the marginal likelihood with respect to the corresponding parameters α_k , we obtain *automatic relevance determination* as discussed in Section 7.2.2.

5.5.2 Early stopping

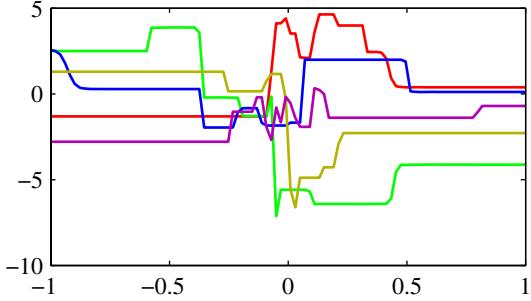
An alternative to regularization as a way of controlling the effective complexity of a network is the procedure of *early stopping*. The training of nonlinear network models corresponds to an iterative reduction of the error function defined with respect to a set of training data. For many of the optimization algorithms used for network training, such as conjugate gradients, the error is a nonincreasing function of the iteration index. However, the error measured with respect to independent data, generally called a validation set, often shows a decrease at first, followed by an increase as the network starts to over-fit. Training can therefore be stopped at the point of smallest error with respect to the validation data set, as indicated in Figure 5.12, in order to obtain a network having good generalization performance.

The behaviour of the network in this case is sometimes explained qualitatively in terms of the effective number of degrees of freedom in the network, in which this number starts out small and then grows during the training process, corresponding to a steady increase in the effective complexity of the model. Halting training before

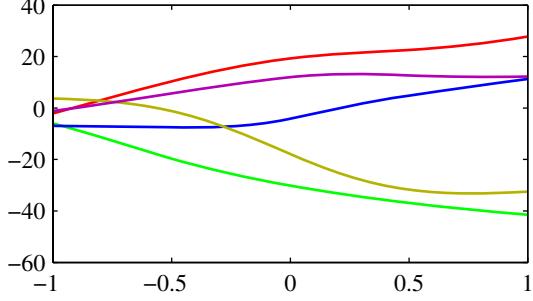
$$\alpha_1^w = 1, \alpha_1^b = 1, \alpha_2^w = 1, \alpha_2^b = 1$$



$$\alpha_1^w = 1000, \alpha_1^b = 100, \alpha_2^w = 1, \alpha_2^b = 1$$



$$\alpha_1^w = 1, \alpha_1^b = 1, \alpha_2^w = 10, \alpha_2^b = 1$$



$$\alpha_1^w = 1000, \alpha_1^b = 1000, \alpha_2^w = 1, \alpha_2^b = 1$$

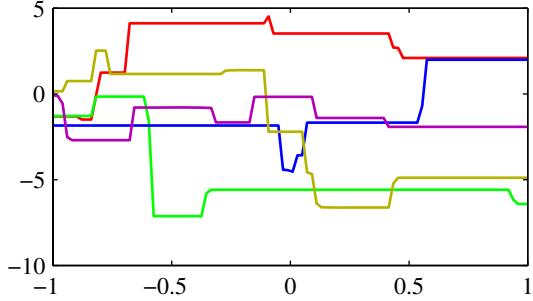


Figure 5.11 Illustration of the effect of the hyperparameters governing the prior distribution over weights and biases in a two-layer network having a single input, a single linear output, and 12 hidden units having ‘tanh’ activation functions. The priors are governed by four hyperparameters α_1^b , α_1^w , α_2^b , and α_2^w , which represent the precisions of the Gaussian distributions of the first-layer biases, first-layer weights, second-layer biases, and second-layer weights, respectively. We see that the parameter α_2^w governs the vertical scale of functions (note the different vertical axis ranges on the top two diagrams), α_1^w governs the horizontal scale of variations in the function values, and α_1^b governs the horizontal range over which variations occur. The parameter α_2^b , whose effect is not illustrated here, governs the range of vertical offsets of the functions.

a minimum of the training error has been reached then represents a way of limiting the effective network complexity.

In the case of a quadratic error function, we can verify this insight, and show that early stopping should exhibit similar behaviour to regularization using a simple weight-decay term. This can be understood from Figure 5.13, in which the axes in weight space have been rotated to be parallel to the eigenvectors of the Hessian matrix. If, in the absence of weight decay, the weight vector starts at the origin and proceeds during training along a path that follows the local negative gradient vector, then the weight vector will move initially parallel to the w_2 axis through a point corresponding roughly to \tilde{w} and then move towards the minimum of the error function w_{ML} . This follows from the shape of the error surface and the widely differing eigenvalues of the Hessian. Stopping at a point near \tilde{w} is therefore similar to weight decay. The relationship between early stopping and weight decay can be made quantitative, thereby showing that the quantity $\tau\eta$ (where τ is the iteration index, and η is the learning rate parameter) plays the role of the reciprocal of the regularization

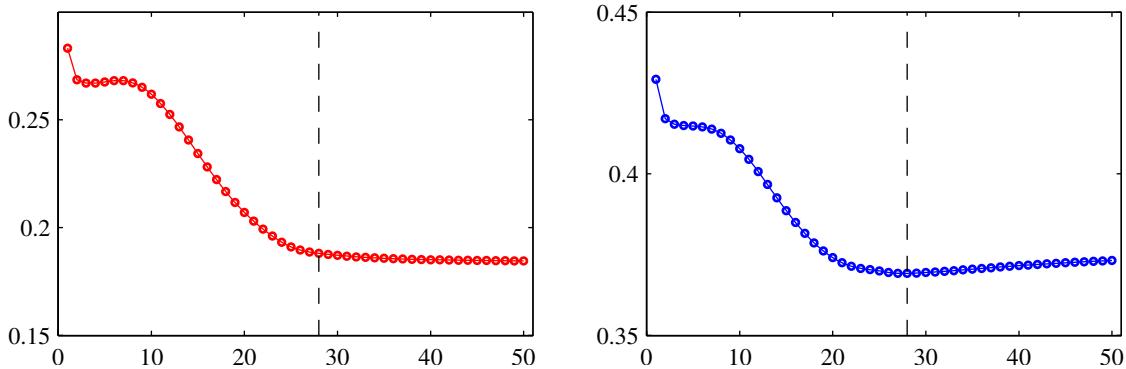


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

parameter λ . The effective number of parameters in the network therefore grows during the course of training.

5.5.3 Invariances

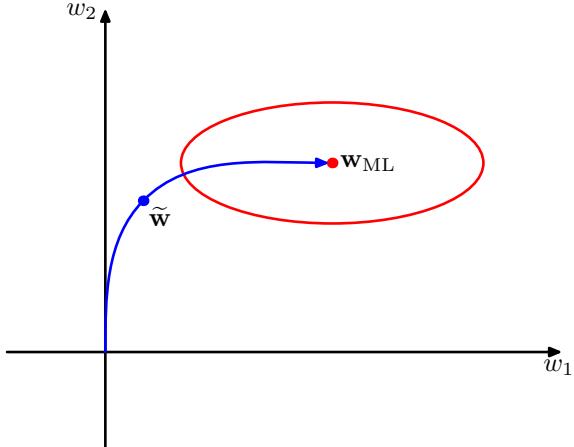
In many applications of pattern recognition, it is known that predictions should be unchanged, or *invariant*, under one or more transformations of the input variables. For example, in the classification of objects in two-dimensional images, such as handwritten digits, a particular object should be assigned the same classification irrespective of its position within the image (*translation invariance*) or of its size (*scale invariance*). Such transformations produce significant changes in the raw data, expressed in terms of the intensities at each of the pixels in the image, and yet should give rise to the same output from the classification system. Similarly in speech recognition, small levels of nonlinear warping along the time axis, which preserve temporal ordering, should not change the interpretation of the signal.

If sufficiently large numbers of training patterns are available, then an adaptive model such as a neural network can learn the invariance, at least approximately. This involves including within the training set a sufficiently large number of examples of the effects of the various transformations. Thus, for translation invariance in an image, the training set should include examples of objects at many different positions.

This approach may be impractical, however, if the number of training examples is limited, or if there are several invariants (because the number of combinations of transformations grows exponentially with the number of such transformations). We therefore seek alternative approaches for encouraging an adaptive model to exhibit the required invariances. These can broadly be divided into four categories:

1. The training set is augmented using replicas of the training patterns, transformed according to the desired invariances. For instance, in our digit recognition example, we could make multiple copies of each example in which the

Figure 5.13 A schematic illustration of why early stopping can give similar results to weight decay in the case of a quadratic error function. The ellipse shows a contour of constant error, and w_{ML} denotes the minimum of the error function. If the weight vector starts at the origin and moves according to the local negative gradient direction, then it will follow the path shown by the curve. By stopping training early, a weight vector \tilde{w} is found that is qualitatively similar to that obtained with a simple weight-decay regularizer and training to the minimum of the regularized error, as can be seen by comparing with Figure 3.15.



digit is shifted to a different position in each image.

2. A regularization term is added to the error function that penalizes changes in the model output when the input is transformed. This leads to the technique of *tangent propagation*, discussed in Section 5.5.4.
3. Invariance is built into the pre-processing by extracting features that are invariant under the required transformations. Any subsequent regression or classification system that uses such features as inputs will necessarily also respect these invariances.
4. The final option is to build the invariance properties into the structure of a neural network (or into the definition of a kernel function in the case of techniques such as the relevance vector machine). One way to achieve this is through the use of local receptive fields and shared weights, as discussed in the context of convolutional neural networks in Section 5.5.6.

Approach 1 is often relatively easy to implement and can be used to encourage complex invariances such as those illustrated in Figure 5.14. For sequential training algorithms, this can be done by transforming each input pattern before it is presented to the model so that, if the patterns are being recycled, a different transformation (drawn from an appropriate distribution) is added each time. For batch methods, a similar effect can be achieved by replicating each data point a number of times and transforming each copy independently. The use of such augmented data can lead to significant improvements in generalization (Simard *et al.*, 2003), although it can also be computationally costly.

Approach 2 leaves the data set unchanged but modifies the error function through the addition of a regularizer. In Section 5.5.5, we shall show that this approach is closely related to approach 2.

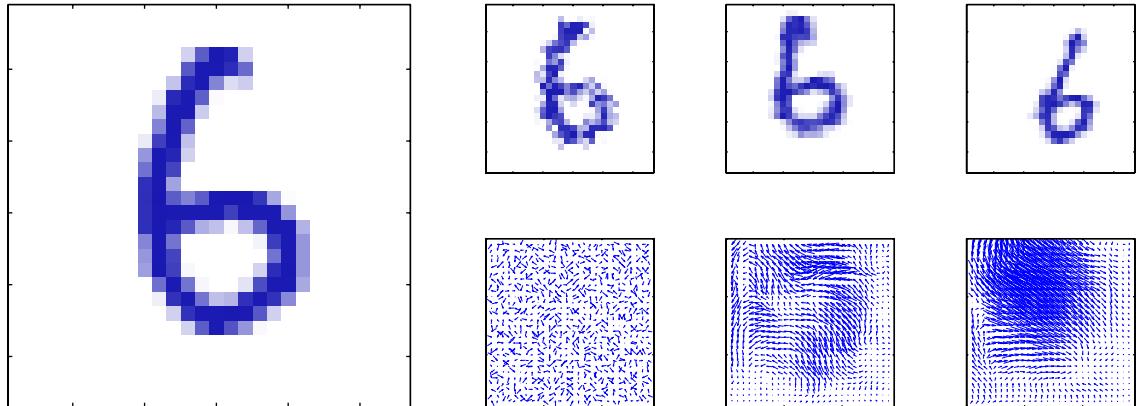


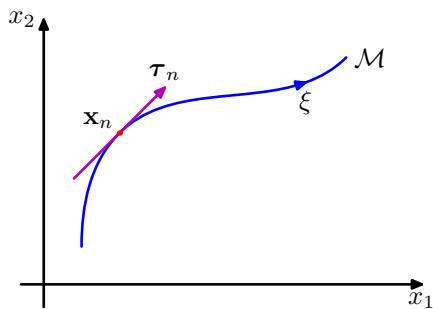
Figure 5.14 Illustration of the synthetic warping of a handwritten digit. The original image is shown on the left. On the right, the top row shows three examples of warped digits, with the corresponding displacement fields shown on the bottom row. These displacement fields are generated by sampling random displacements $\Delta x, \Delta y \in (0, 1)$ at each pixel and then smoothing by convolution with Gaussians of width 0.01, 30 and 60 respectively.

One advantage of approach 3 is that it can correctly extrapolate well beyond the range of transformations included in the training set. However, it can be difficult to find hand-crafted features with the required invariances that do not also discard information that can be useful for discrimination.

5.5.4 Tangent propagation

We can use regularization to encourage models to be invariant to transformations of the input through the technique of *tangent propagation* (Simard *et al.*, 1992). Consider the effect of a transformation on a particular input vector x_n . Provided the transformation is continuous (such as translation or rotation, but not mirror reflection for instance), then the transformed pattern will sweep out a manifold \mathcal{M} within the D -dimensional input space. This is illustrated in Figure 5.15, for the case of $D = 2$ for simplicity. Suppose the transformation is governed by a single parameter ξ (which might be rotation angle for instance). Then the subspace \mathcal{M} swept out by x_n

Figure 5.15 Illustration of a two-dimensional input space showing the effect of a continuous transformation on a particular input vector x_n . A one-dimensional transformation, parameterized by the continuous variable ξ , applied to x_n causes it to sweep out a one-dimensional manifold \mathcal{M} . Locally, the effect of the transformation can be approximated by the tangent vector τ_n .



will be one-dimensional, and will be parameterized by ξ . Let the vector that results from acting on \mathbf{x}_n by this transformation be denoted by $\mathbf{s}(\mathbf{x}_n, \xi)$, which is defined so that $\mathbf{s}(\mathbf{x}, 0) = \mathbf{x}$. Then the tangent to the curve \mathcal{M} is given by the directional derivative $\boldsymbol{\tau} = \partial \mathbf{s} / \partial \xi$, and the tangent vector at the point \mathbf{x}_n is given by

$$\boldsymbol{\tau}_n = \left. \frac{\partial \mathbf{s}(\mathbf{x}_n, \xi)}{\partial \xi} \right|_{\xi=0}. \quad (5.125)$$

Under a transformation of the input vector, the network output vector will, in general, change. The derivative of output k with respect to ξ is given by

$$\left. \frac{\partial y_k}{\partial \xi} \right|_{\xi=0} = \sum_{i=1}^D \left. \frac{\partial y_k}{\partial x_i} \frac{\partial x_i}{\partial \xi} \right|_{\xi=0} = \sum_{i=1}^D J_{ki} \tau_i \quad (5.126)$$

where J_{ki} is the (k, i) element of the Jacobian matrix \mathbf{J} , as discussed in Section 5.3.4. The result (5.126) can be used to modify the standard error function, so as to encourage local invariance in the neighbourhood of the data points, by the addition to the original error function E of a regularization function Ω to give a total error function of the form

$$\tilde{E} = E + \lambda \Omega \quad (5.127)$$

where λ is a regularization coefficient and

$$\Omega = \frac{1}{2} \sum_n \sum_k \left(\left. \frac{\partial y_{nk}}{\partial \xi} \right|_{\xi=0} \right)^2 = \frac{1}{2} \sum_n \sum_k \left(\sum_{i=1}^D J_{nki} \tau_{ni} \right)^2. \quad (5.128)$$

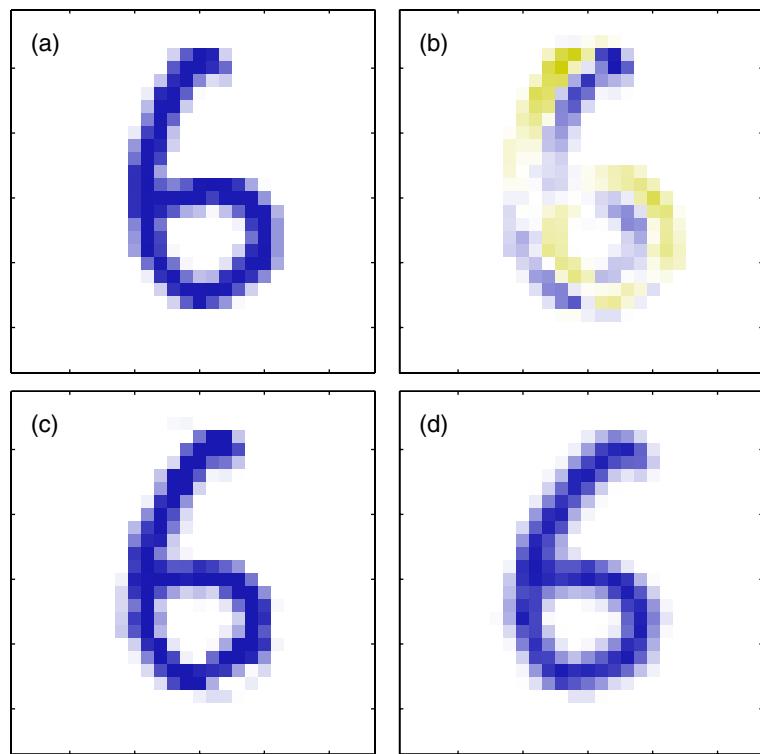
The regularization function will be zero when the network mapping function is invariant under the transformation in the neighbourhood of each pattern vector, and the value of the parameter λ determines the balance between fitting the training data and learning the invariance property.

In a practical implementation, the tangent vector $\boldsymbol{\tau}_n$ can be approximated using finite differences, by subtracting the original vector \mathbf{x}_n from the corresponding vector after transformation using a small value of ξ , and then dividing by ξ . This is illustrated in Figure 5.16.

The regularization function depends on the network weights through the Jacobian \mathbf{J} . A backpropagation formalism for computing the derivatives of the regularizer with respect to the network weights is easily obtained by extension of the techniques introduced in Section 5.3.

If the transformation is governed by L parameters (e.g., $L = 3$ for the case of translations combined with in-plane rotations in a two-dimensional image), then the manifold \mathcal{M} will have dimensionality L , and the corresponding regularizer is given by the sum of terms of the form (5.128), one for each transformation. If several transformations are considered at the same time, and the network mapping is made invariant to each separately, then it will be (locally) invariant to combinations of the transformations (Simard *et al.*, 1992).

Figure 5.16 Illustration showing (a) the original image x of a handwritten digit, (b) the tangent vector τ corresponding to an infinitesimal clockwise rotation, (c) the result of adding a small contribution from the tangent vector to the original image giving $x + \epsilon\tau$ with $\epsilon = 15$ degrees, and (d) the true image rotated for comparison.



A related technique, called *tangent distance*, can be used to build invariance properties into distance-based methods such as nearest-neighbour classifiers (Simard *et al.*, 1993).

5.5.5 Training with transformed data

We have seen that one way to encourage invariance of a model to a set of transformations is to expand the training set using transformed versions of the original input patterns. Here we show that this approach is closely related to the technique of tangent propagation (Bishop, 1995b; Leen, 1995).

As in Section 5.5.4, we shall consider a transformation governed by a single parameter ξ and described by the function $s(x, \xi)$, with $s(x, 0) = x$. We shall also consider a sum-of-squares error function. The error function for untransformed inputs can be written (in the infinite data set limit) in the form

$$E = \frac{1}{2} \iint \{y(x) - t\}^2 p(t|x)p(x) dx dt \quad (5.129)$$

as discussed in Section 1.5.5. Here we have considered a network having a single output, in order to keep the notation uncluttered. If we now consider an infinite number of copies of each data point, each of which is perturbed by the transformation

in which the parameter ξ is drawn from a distribution $p(\xi)$, then the error function defined over this expanded data set can be written as

$$\tilde{E} = \frac{1}{2} \iiint \{y(\mathbf{s}(\mathbf{x}, \xi)) - t\}^2 p(t|\mathbf{x}) p(\mathbf{x}) p(\xi) d\mathbf{x} dt d\xi. \quad (5.130)$$

We now assume that the distribution $p(\xi)$ has zero mean with small variance, so that we are only considering small transformations of the original input vectors. We can then expand the transformation function as a Taylor series in powers of ξ to give

$$\begin{aligned} \mathbf{s}(\mathbf{x}, \xi) &= \mathbf{s}(\mathbf{x}, 0) + \xi \frac{\partial}{\partial \xi} \mathbf{s}(\mathbf{x}, \xi) \Big|_{\xi=0} + \frac{\xi^2}{2} \frac{\partial^2}{\partial \xi^2} \mathbf{s}(\mathbf{x}, \xi) \Big|_{\xi=0} + O(\xi^3) \\ &= \mathbf{x} + \xi \boldsymbol{\tau} + \frac{1}{2} \xi^2 \boldsymbol{\tau}' + O(\xi^3) \end{aligned}$$

where $\boldsymbol{\tau}'$ denotes the second derivative of $\mathbf{s}(\mathbf{x}, \xi)$ with respect to ξ evaluated at $\xi = 0$. This allows us to expand the model function to give

$$y(\mathbf{s}(\mathbf{x}, \xi)) = y(\mathbf{x}) + \xi \boldsymbol{\tau}^T \nabla y(\mathbf{x}) + \frac{\xi^2}{2} \left[(\boldsymbol{\tau}')^T \nabla y(\mathbf{x}) + \boldsymbol{\tau}^T \nabla \nabla y(\mathbf{x}) \boldsymbol{\tau} \right] + O(\xi^3).$$

Substituting into the mean error function (5.130) and expanding, we then have

$$\begin{aligned} \tilde{E} &= \frac{1}{2} \iint \{y(\mathbf{x}) - t\}^2 p(t|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} dt \\ &+ \mathbb{E}[\xi] \iint \{y(\mathbf{x}) - t\} \boldsymbol{\tau}^T \nabla y(\mathbf{x}) p(t|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} dt \\ &+ \mathbb{E}[\xi^2] \iint \left[\{y(\mathbf{x}) - t\} \frac{1}{2} \left\{ (\boldsymbol{\tau}')^T \nabla y(\mathbf{x}) + \boldsymbol{\tau}^T \nabla \nabla y(\mathbf{x}) \boldsymbol{\tau} \right\} \right. \\ &\quad \left. + (\boldsymbol{\tau}^T \nabla y(\mathbf{x}))^2 \right] p(t|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} dt + O(\xi^3). \end{aligned}$$

Because the distribution of transformations has zero mean we have $\mathbb{E}[\xi] = 0$. Also, we shall denote $\mathbb{E}[\xi^2]$ by λ . Omitting terms of $O(\xi^3)$, the average error function then becomes

$$\tilde{E} = E + \lambda \Omega \quad (5.131)$$

where E is the original sum-of-squares error, and the regularization term Ω takes the form

$$\begin{aligned} \Omega &= \int \left[\{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\} \frac{1}{2} \left\{ (\boldsymbol{\tau}')^T \nabla y(\mathbf{x}) + \boldsymbol{\tau}^T \nabla \nabla y(\mathbf{x}) \boldsymbol{\tau} \right\} \right. \\ &\quad \left. + (\boldsymbol{\tau}^T \nabla y(\mathbf{x}))^2 \right] p(\mathbf{x}) d\mathbf{x} \end{aligned} \quad (5.132)$$

in which we have performed the integration over t .

We can further simplify this regularization term as follows. In Section 1.5.5 we saw that the function that minimizes the sum-of-squares error is given by the conditional average $\mathbb{E}[t|\mathbf{x}]$ of the target values t . From (5.131) we see that the regularized error will equal the unregularized sum-of-squares plus terms which are $O(\xi)$, and so the network function that minimizes the total error will have the form

$$y(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] + O(\xi). \quad (5.133)$$

Thus, to leading order in ξ , the first term in the regularizer vanishes and we are left with

$$\Omega = \frac{1}{2} \int (\boldsymbol{\tau}^T \nabla y(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x} \quad (5.134)$$

which is equivalent to the tangent propagation regularizer (5.128).

If we consider the special case in which the transformation of the inputs simply consists of the addition of random noise, so that $\mathbf{x} \rightarrow \mathbf{x} + \boldsymbol{\xi}$, then the regularizer takes the form

$$\Omega = \frac{1}{2} \int \|\nabla y(\mathbf{x})\|^2 p(\mathbf{x}) d\mathbf{x} \quad (5.135)$$

which is known as *Tikhonov regularization* (Tikhonov and Arsenin, 1977; Bishop, 1995b). Derivatives of this regularizer with respect to the network weights can be found using an extended backpropagation algorithm (Bishop, 1993). We see that, for small noise amplitudes, Tikhonov regularization is related to the addition of random noise to the inputs, which has been shown to improve generalization in appropriate circumstances (Sietsma and Dow, 1991).

5.5.6 Convolutional networks

Another approach to creating models that are invariant to certain transformations of the inputs is to build the invariance properties into the structure of a neural network. This is the basis for the *convolutional neural network* (Le Cun *et al.*, 1989; LeCun *et al.*, 1998), which has been widely applied to image data.

Consider the specific task of recognizing handwritten digits. Each input image comprises a set of pixel intensity values, and the desired output is a posterior probability distribution over the ten digit classes. We know that the identity of the digit is invariant under translations and scaling as well as (small) rotations. Furthermore, the network must also exhibit invariance to more subtle transformations such as elastic deformations of the kind illustrated in Figure 5.14. One simple approach would be to treat the image as the input to a fully connected network, such as the kind shown in Figure 5.1. Given a sufficiently large training set, such a network could in principle yield a good solution to this problem and would learn the appropriate invariances by example.

However, this approach ignores a key property of images, which is that nearby pixels are more strongly correlated than more distant pixels. Many of the modern approaches to computer vision exploit this property by extracting *local* features that depend only on small subregions of the image. Information from such features can then be merged in later stages of processing in order to detect higher-order features

Exercise 5.27

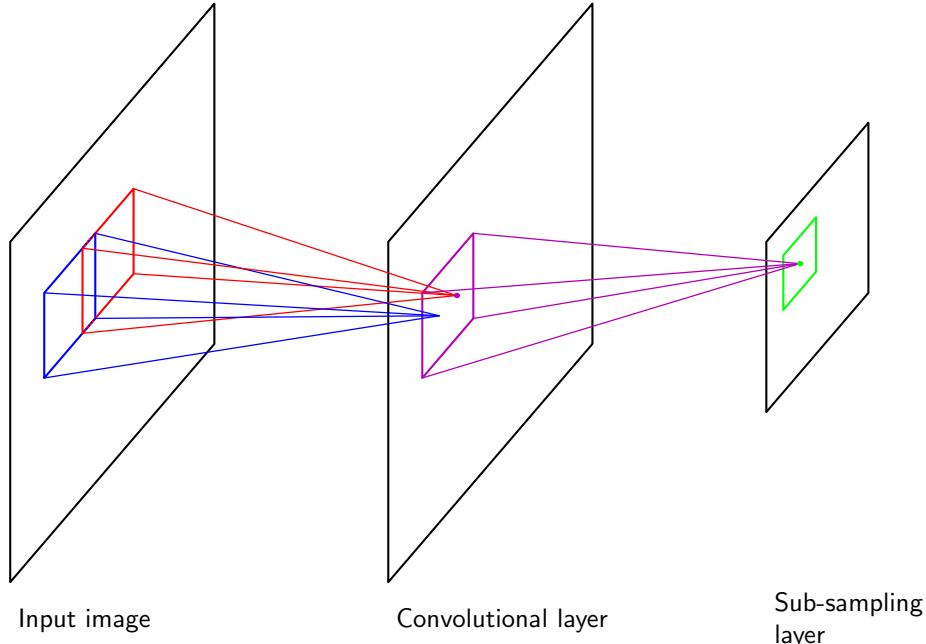


Figure 5.17 Diagram illustrating part of a convolutional neural network, showing a layer of convolutional units followed by a layer of subsampling units. Several successive pairs of such layers may be used.

and ultimately to yield information about the image as whole. Also, local features that are useful in one region of the image are likely to be useful in other regions of the image, for instance if the object of interest is translated.

These notions are incorporated into convolutional neural networks through three mechanisms: (i) local receptive fields, (ii) weight sharing, and (iii) subsampling. The structure of a convolutional network is illustrated in Figure 5.17. In the convolutional layer the units are organized into planes, each of which is called a *feature map*. Units in a feature map each take inputs only from a small subregion of the image, and all of the units in a feature map are constrained to share the same weight values. For instance, a feature map might consist of 100 units arranged in a 10×10 grid, with each unit taking inputs from a 5×5 pixel patch of the image. The whole feature map therefore has 25 adjustable weight parameters plus one adjustable bias parameter. Input values from a patch are linearly combined using the weights and the bias, and the result transformed by a sigmoidal nonlinearity using (5.1). If we think of the units as feature detectors, then all of the units in a feature map detect the same pattern but at different locations in the input image. Due to the weight sharing, the evaluation of the activations of these units is equivalent to a convolution of the image pixel intensities with a ‘kernel’ comprising the weight parameters. If the input image is shifted, the activations of the feature map will be shifted by the same amount but will otherwise be unchanged. This provides the basis for the (approximate) invariance of

the network outputs to translations and distortions of the input image. Because we will typically need to detect multiple features in order to build an effective model, there will generally be multiple feature maps in the convolutional layer, each having its own set of weight and bias parameters.

The outputs of the convolutional units form the inputs to the subsampling layer of the network. For each feature map in the convolutional layer, there is a plane of units in the subsampling layer and each unit takes inputs from a small receptive field in the corresponding feature map of the convolutional layer. These units perform subsampling. For instance, each subsampling unit might take inputs from a 2×2 unit region in the corresponding feature map and would compute the average of those inputs, multiplied by an adaptive weight with the addition of an adaptive bias parameter, and then transformed using a sigmoidal nonlinear activation function. The receptive fields are chosen to be contiguous and nonoverlapping so that there are half the number of rows and columns in the subsampling layer compared with the convolutional layer. In this way, the response of a unit in the subsampling layer will be relatively insensitive to small shifts of the image in the corresponding regions of the input space.

In a practical architecture, there may be several pairs of convolutional and subsampling layers. At each stage there is a larger degree of invariance to input transformations compared to the previous layer. There may be several feature maps in a given convolutional layer for each plane of units in the previous subsampling layer, so that the gradual reduction in spatial resolution is then compensated by an increasing number of features. The final layer of the network would typically be a fully connected, fully adaptive layer, with a softmax output nonlinearity in the case of multiclass classification.

The whole network can be trained by error minimization using backpropagation to evaluate the gradient of the error function. This involves a slight modification of the usual backpropagation algorithm to ensure that the shared-weight constraints are satisfied. Due to the use of local receptive fields, the number of weights in the network is smaller than if the network were fully connected. Furthermore, the number of independent parameters to be learned from the data is much smaller still, due to the substantial numbers of constraints on the weights.

5.5.7 Soft weight sharing

One way to reduce the effective complexity of a network with a large number of weights is to constrain weights within certain groups to be equal. This is the technique of weight sharing that was discussed in Section 5.5.6 as a way of building translation invariance into networks used for image interpretation. It is only applicable, however, to particular problems in which the form of the constraints can be specified in advance. Here we consider a form of *soft weight sharing* (Nowlan and Hinton, 1992) in which the hard constraint of equal weights is replaced by a form of regularization in which groups of weights are encouraged to have similar values. Furthermore, the division of weights into groups, the mean weight value for each group, and the spread of values within the groups are all determined as part of the learning process.

Exercise 5.28

Section 2.3.9

Recall that the simple weight decay regularizer, given in (5.112), can be viewed as the negative log of a Gaussian prior distribution over the weights. We can encourage the weight values to form several groups, rather than just one group, by considering instead a probability distribution that is a *mixture* of Gaussians. The centres and variances of the Gaussian components, as well as the mixing coefficients, will be considered as adjustable parameters to be determined as part of the learning process. Thus, we have a probability density of the form

$$p(\mathbf{w}) = \prod_i p(w_i) \quad (5.136)$$

where

$$p(w_i) = \sum_{j=1}^M \pi_j \mathcal{N}(w_i | \mu_j, \sigma_j^2) \quad (5.137)$$

and π_j are the mixing coefficients. Taking the negative logarithm then leads to a regularization function of the form

$$\Omega(\mathbf{w}) = - \sum_i \ln \left(\sum_{j=1}^M \pi_j \mathcal{N}(w_i | \mu_j, \sigma_j^2) \right). \quad (5.138)$$

The total error function is then given by

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda \Omega(\mathbf{w}) \quad (5.139)$$

where λ is the regularization coefficient. This error is minimized both with respect to the weights w_i and with respect to the parameters $\{\pi_j, \mu_j, \sigma_j\}$ of the mixture model. If the weights were constant, then the parameters of the mixture model could be determined by using the EM algorithm discussed in Chapter 9. However, the distribution of weights is itself evolving during the learning process, and so to avoid numerical instability, a joint optimization is performed simultaneously over the weights and the mixture-model parameters. This can be done using a standard optimization algorithm such as conjugate gradients or quasi-Newton methods.

In order to minimize the total error function, it is necessary to be able to evaluate its derivatives with respect to the various adjustable parameters. To do this it is convenient to regard the $\{\pi_j\}$ as *prior* probabilities and to introduce the corresponding posterior probabilities which, following (2.192), are given by Bayes' theorem in the form

$$\gamma_j(w) = \frac{\pi_j \mathcal{N}(w | \mu_j, \sigma_j^2)}{\sum_k \pi_k \mathcal{N}(w | \mu_k, \sigma_k^2)}. \quad (5.140)$$

The derivatives of the total error function with respect to the weights are then given by

$$\frac{\partial \tilde{E}}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda \sum_j \gamma_j(w_i) \frac{(w_i - \mu_j)}{\sigma_j^2}. \quad (5.141)$$

Exercise 5.29

The effect of the regularization term is therefore to pull each weight towards the centre of the j^{th} Gaussian, with a force proportional to the posterior probability of that Gaussian for the given weight. This is precisely the kind of effect that we are seeking.

Derivatives of the error with respect to the centres of the Gaussians are also easily computed to give

$$\frac{\partial \tilde{E}}{\partial \mu_j} = \lambda \sum_i \gamma_j(w_i) \frac{(\mu_i - w_j)}{\sigma_j^2} \quad (5.142)$$

which has a simple intuitive interpretation, because it pushes μ_j towards an average of the weight values, weighted by the posterior probabilities that the respective weight parameters were generated by component j . Similarly, the derivatives with respect to the variances are given by

$$\frac{\partial \tilde{E}}{\partial \sigma_j} = \lambda \sum_i \gamma_j(w_i) \left(\frac{1}{\sigma_j} - \frac{(w_i - \mu_j)^2}{\sigma_j^3} \right) \quad (5.143)$$

which drives σ_j towards the weighted average of the squared deviations of the weights around the corresponding centre μ_j , where the weighting coefficients are again given by the posterior probability that each weight is generated by component j . Note that in a practical implementation, new variables η_j defined by

$$\sigma_j^2 = \exp(\eta_j) \quad (5.144)$$

are introduced, and the minimization is performed with respect to the η_j . This ensures that the parameters σ_j remain positive. It also has the effect of discouraging pathological solutions in which one or more of the σ_j goes to zero, corresponding to a Gaussian component collapsing onto one of the weight parameter values. Such solutions are discussed in more detail in the context of Gaussian mixture models in Section 9.2.1.

For the derivatives with respect to the mixing coefficients π_j , we need to take account of the constraints

$$\sum_j \pi_j = 1, \quad 0 \leq \pi_i \leq 1 \quad (5.145)$$

which follow from the interpretation of the π_j as prior probabilities. This can be done by expressing the mixing coefficients in terms of a set of auxiliary variables $\{\eta_j\}$ using the *softmax* function given by

$$\pi_j = \frac{\exp(\eta_j)}{\sum_{k=1}^M \exp(\eta_k)}. \quad (5.146)$$

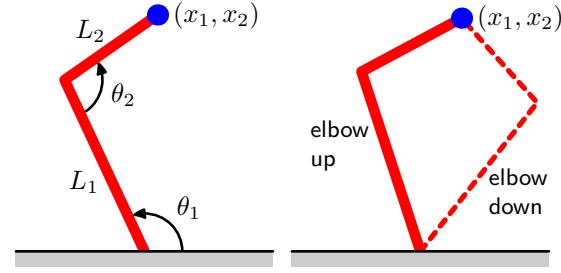
The derivatives of the regularized error function with respect to the $\{\eta_j\}$ then take the form

Exercise 5.30

Exercise 5.31

Exercise 5.32

Figure 5.18 The left figure shows a two-link robot arm, in which the Cartesian coordinates (x_1, x_2) of the end effector are determined uniquely by the two joint angles θ_1 and θ_2 and the (fixed) lengths L_1 and L_2 of the arms. This is known as the *forward kinematics* of the arm. In practice, we have to find the joint angles that will give rise to a desired end effector position and, as shown in the right figure, this *inverse kinematics* has two solutions corresponding to ‘elbow up’ and ‘elbow down’.



$$\frac{\partial \tilde{E}}{\partial \eta_j} = \sum_i \{\pi_j - \gamma_j(w_i)\}. \quad (5.147)$$

We see that π_j is therefore driven towards the average posterior probability for component j .

5.6. Mixture Density Networks

The goal of supervised learning is to model a conditional distribution $p(\mathbf{t}|\mathbf{x})$, which for many simple regression problems is chosen to be Gaussian. However, practical machine learning problems can often have significantly non-Gaussian distributions. These can arise, for example, with *inverse problems* in which the distribution can be multimodal, in which case the Gaussian assumption can lead to very poor predictions.

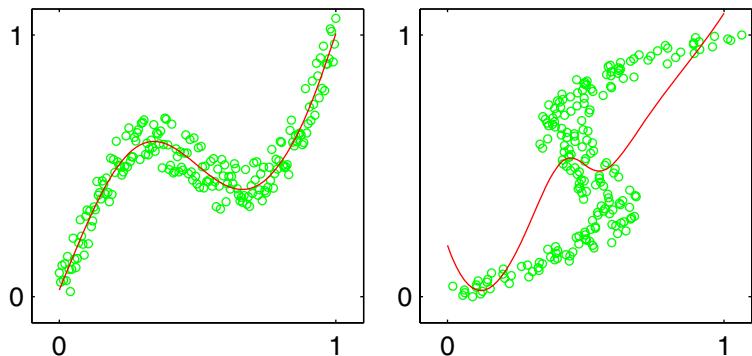
As a simple example of an inverse problem, consider the kinematics of a robot arm, as illustrated in Figure 5.18. The *forward problem* involves finding the end effector position given the joint angles and has a unique solution. However, in practice we wish to move the end effector of the robot to a specific position, and to do this we must set appropriate joint angles. We therefore need to solve the inverse problem, which has two solutions as seen in Figure 5.18.

Forward problems often correspond to causality in a physical system and generally have a unique solution. For instance, a specific pattern of symptoms in the human body may be caused by the presence of a particular disease. In pattern recognition, however, we typically have to solve an inverse problem, such as trying to predict the presence of a disease given a set of symptoms. If the forward problem involves a many-to-one mapping, then the inverse problem will have multiple solutions. For instance, several different diseases may result in the same symptoms.

In the robotics example, the kinematics is defined by geometrical equations, and the multimodality is readily apparent. However, in many machine learning problems the presence of multimodality, particularly in problems involving spaces of high dimensionality, can be less obvious. For tutorial purposes, however, we shall consider a simple toy problem for which we can easily visualize the multimodality. Data for this problem is generated by sampling a variable x uniformly over the interval $(0, 1)$, to give a set of values $\{x_n\}$, and the corresponding target values t_n are obtained

Exercise 5.33

Figure 5.19 On the left is the data set for a simple ‘forward problem’ in which the red curve shows the result of fitting a two-layer neural network by minimizing the sum-of-squares error function. The corresponding inverse problem, shown on the right, is obtained by exchanging the roles of x and t . Here the same network trained again by minimizing the sum-of-squares error function gives a very poor fit to the data due to the multimodality of the data set.



by computing the function $x_n + 0.3 \sin(2\pi x_n)$ and then adding uniform noise over the interval $(-0.1, 0.1)$. The inverse problem is then obtained by keeping the same data points but exchanging the roles of x and t . Figure 5.19 shows the data sets for the forward and inverse problems, along with the results of fitting two-layer neural networks having 6 hidden units and a single linear output unit by minimizing a sum-of-squares error function. Least squares corresponds to maximum likelihood under a Gaussian assumption. We see that this leads to a very poor model for the highly non-Gaussian inverse problem.

We therefore seek a general framework for modelling conditional probability distributions. This can be achieved by using a mixture model for $p(t|x)$ in which both the mixing coefficients as well as the component densities are flexible functions of the input vector x , giving rise to the *mixture density network*. For any given value of x , the mixture model provides a general formalism for modelling an arbitrary conditional density function $p(t|x)$. Provided we consider a sufficiently flexible network, we then have a framework for approximating arbitrary conditional distributions.

Here we shall develop the model explicitly for Gaussian components, so that

$$p(t|x) = \sum_{k=1}^K \pi_k(x) \mathcal{N}(t|\mu_k(x), \sigma_k^2(x)). \quad (5.148)$$

This is an example of a *heteroscedastic* model since the noise variance on the data is a function of the input vector x . Instead of Gaussians, we can use other distributions for the components, such as Bernoulli distributions if the target variables are binary rather than continuous. We have also specialized to the case of isotropic covariances for the components, although the mixture density network can readily be extended to allow for general covariance matrices by representing the covariances using a Cholesky factorization (Williams, 1996). Even with isotropic components, the conditional distribution $p(t|x)$ does not assume factorization with respect to the components of t (in contrast to the standard sum-of-squares regression model) as a consequence of the mixture distribution.

We now take the various parameters of the mixture model, namely the mixing coefficients $\pi_k(x)$, the means $\mu_k(x)$, and the variances $\sigma_k^2(x)$, to be governed by

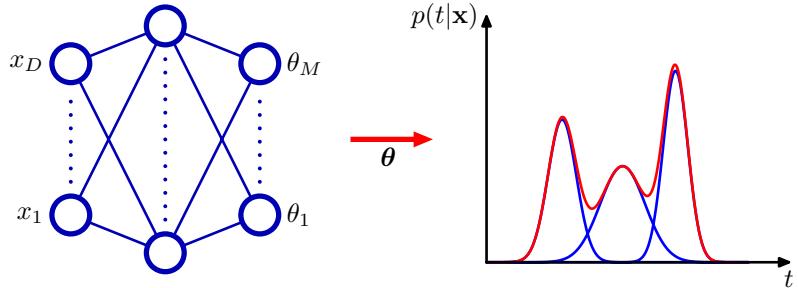


Figure 5.20 The *mixture density network* can represent general conditional probability densities $p(t|x)$ by considering a parametric mixture model for the distribution of t whose parameters are determined by the outputs of a neural network that takes x as its input vector.

the outputs of a conventional neural network that takes \mathbf{x} as its input. The structure of this mixture density network is illustrated in Figure 5.20. The mixture density network is closely related to the mixture of experts discussed in Section 14.5.3. The principle difference is that in the mixture density network the same function is used to predict the parameters of all of the component densities as well as the mixing coefficients, and so the nonlinear hidden units are shared amongst the input-dependent functions.

The neural network in Figure 5.20 can, for example, be a two-layer network having sigmoidal ('tanh') hidden units. If there are L components in the mixture model (5.148), and if \mathbf{t} has K components, then the network will have L output unit activations denoted by a_k^π that determine the mixing coefficients $\pi_k(\mathbf{x})$, K outputs denoted by a_k^σ that determine the kernel widths $\sigma_k(\mathbf{x})$, and $L \times K$ outputs denoted by a_{kj}^μ that determine the components $\mu_{kj}(\mathbf{x})$ of the kernel centres $\mu_k(\mathbf{x})$. The total number of network outputs is given by $(K + 2)L$, as compared with the usual K outputs for a network, which simply predicts the conditional means of the target variables.

The mixing coefficients must satisfy the constraints

$$\sum_{k=1}^K \pi_k(\mathbf{x}) = 1, \quad 0 \leq \pi_k(\mathbf{x}) \leq 1 \quad (5.149)$$

which can be achieved using a set of softmax outputs

$$\pi_k(\mathbf{x}) = \frac{\exp(a_k^\pi)}{\sum_{l=1}^K \exp(a_l^\pi)}. \quad (5.150)$$

Similarly, the variances must satisfy $\sigma_k^2(\mathbf{x}) \geq 0$ and so can be represented in terms of the exponentials of the corresponding network activations using

$$\sigma_k(\mathbf{x}) = \exp(a_k^\sigma). \quad (5.151)$$

Finally, because the means $\mu_k(\mathbf{x})$ have real components, they can be represented

directly by the network output activations

$$\mu_{kj}(\mathbf{x}) = a_{kj}^\mu. \quad (5.152)$$

The adaptive parameters of the mixture density network comprise the vector \mathbf{w} of weights and biases in the neural network, that can be set by maximum likelihood, or equivalently by minimizing an error function defined to be the negative logarithm of the likelihood. For independent data, this error function takes the form

$$E(\mathbf{w}) = - \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k(\mathbf{x}_n, \mathbf{w}) \mathcal{N}(\mathbf{t}_n | \boldsymbol{\mu}_k(\mathbf{x}_n, \mathbf{w}), \sigma_k^2(\mathbf{x}_n, \mathbf{w})) \right\} \quad (5.153)$$

where we have made the dependencies on \mathbf{w} explicit.

In order to minimize the error function, we need to calculate the derivatives of the error $E(\mathbf{w})$ with respect to the components of \mathbf{w} . These can be evaluated by using the standard backpropagation procedure, provided we obtain suitable expressions for the derivatives of the error with respect to the output-unit activations. These represent error signals δ for each pattern and for each output unit, and can be back-propagated to the hidden units and the error function derivatives evaluated in the usual way. Because the error function (5.153) is composed of a sum of terms, one for each training data point, we can consider the derivatives for a particular pattern n and then find the derivatives of E by summing over all patterns.

Because we are dealing with mixture distributions, it is convenient to view the mixing coefficients $\pi_k(\mathbf{x})$ as \mathbf{x} -dependent prior probabilities and to introduce the corresponding posterior probabilities given by

$$\gamma_k(\mathbf{t}|\mathbf{x}) = \frac{\pi_k \mathcal{N}_{nk}}{\sum_{l=1}^K \pi_l \mathcal{N}_{nl}} \quad (5.154)$$

where \mathcal{N}_{nk} denotes $\mathcal{N}(\mathbf{t}_n | \boldsymbol{\mu}_k(\mathbf{x}_n), \sigma_k^2(\mathbf{x}_n))$.

The derivatives with respect to the network output activations governing the mixing coefficients are given by

$$\frac{\partial E_n}{\partial a_k^\pi} = \pi_k - \gamma_k. \quad (5.155)$$

Similarly, the derivatives with respect to the output activations controlling the component means are given by

$$\frac{\partial E_n}{\partial a_{kl}^\mu} = \gamma_k \left\{ \frac{\mu_{kl} - t_l}{\sigma_k^2} \right\}. \quad (5.156)$$

Finally, the derivatives with respect to the output activations controlling the component variances are given by

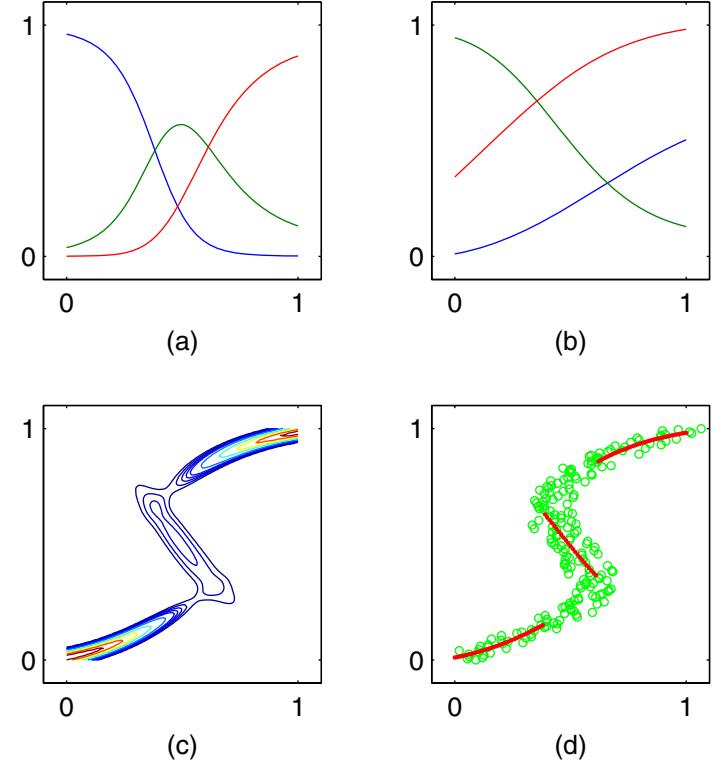
$$\frac{\partial E_n}{\partial a_k^\sigma} = -\gamma_k \left\{ \frac{\|\mathbf{t} - \boldsymbol{\mu}_k\|^2}{\sigma_k^3} - \frac{1}{\sigma_k} \right\}. \quad (5.157)$$

Exercise 5.34

Exercise 5.35

Exercise 5.36

Figure 5.21 (a) Plot of the mixing coefficients $\pi_k(x)$ as a function of x for the three kernel functions in a mixture density network trained on the data shown in Figure 5.19. The model has three Gaussian components, and uses a two-layer multi-layer perceptron with five ‘tanh’ sigmoidal units in the hidden layer, and nine outputs (corresponding to the 3 means and 3 variances of the Gaussian components and the 3 mixing coefficients). At both small and large values of x , where the conditional probability density of the target data is unimodal, only one of the kernels has a high value for its prior probability, while at intermediate values of x , where the conditional density is trimodal, the three mixing coefficients have comparable values. (b) Plots of the means $\mu_k(x)$ using the same colour coding as for the mixing coefficients. (c) Plot of the contours of the corresponding conditional probability density of the target data for the same mixture density network. (d) Plot of the approximate conditional mode, shown by the red points, of the conditional density.



We illustrate the use of a mixture density network by returning to the toy example of an inverse problem shown in Figure 5.19. Plots of the mixing coefficients $\pi_k(x)$, the means $\mu_k(x)$, and the conditional density contours corresponding to $p(t|x)$, are shown in Figure 5.21. The outputs of the neural network, and hence the parameters in the mixture model, are necessarily continuous single-valued functions of the input variables. However, we see from Figure 5.21(c) that the model is able to produce a conditional density that is unimodal for some values of x and trimodal for other values by modulating the amplitudes of the mixing components $\pi_k(x)$.

Once a mixture density network has been trained, it can predict the conditional density function of the target data for any given value of the input vector. This conditional density represents a complete description of the generator of the data, so far as the problem of predicting the value of the output vector is concerned. From this density function we can calculate more specific quantities that may be of interest in different applications. One of the simplest of these is the mean, corresponding to the conditional average of the target data, and is given by

$$\mathbb{E}[t|x] = \int tp(t|x) dt = \sum_{k=1}^K \pi_k(x) \mu_k(x) \quad (5.158)$$

where we have used (5.148). Because a standard network trained by least squares is approximating the conditional mean, we see that a mixture density network can reproduce the conventional least-squares result as a special case. Of course, as we have already noted, for a multimodal distribution the conditional mean is of limited value.

We can similarly evaluate the variance of the density function about the conditional average, to give

$$s^2(\mathbf{x}) = \mathbb{E} [\|\mathbf{t} - \mathbb{E}[\mathbf{t}|\mathbf{x}]\|^2 | \mathbf{x}] \quad (5.159)$$

$$= \sum_{k=1}^K \pi_k(\mathbf{x}) \left\{ \sigma_k^2(\mathbf{x}) + \left\| \boldsymbol{\mu}_k(\mathbf{x}) - \sum_{l=1}^K \pi_l(\mathbf{x}) \boldsymbol{\mu}_l(\mathbf{x}) \right\|^2 \right\} \quad (5.160)$$

where we have used (5.148) and (5.158). This is more general than the corresponding least-squares result because the variance is a function of \mathbf{x} .

We have seen that for multimodal distributions, the conditional mean can give a poor representation of the data. For instance, in controlling the simple robot arm shown in Figure 5.18, we need to pick one of the two possible joint angle settings in order to achieve the desired end-effector location, whereas the average of the two solutions is not itself a solution. In such cases, the conditional mode may be of more value. Because the conditional mode for the mixture density network does not have a simple analytical solution, this would require numerical iteration. A simple alternative is to take the mean of the most probable component (i.e., the one with the largest mixing coefficient) at each value of \mathbf{x} . This is shown for the toy data set in Figure 5.21(d).

5.7. Bayesian Neural Networks

So far, our discussion of neural networks has focussed on the use of maximum likelihood to determine the network parameters (weights and biases). Regularized maximum likelihood can be interpreted as a MAP (maximum posterior) approach in which the regularizer can be viewed as the logarithm of a prior parameter distribution. However, in a Bayesian treatment we need to marginalize over the distribution of parameters in order to make predictions.

In Section 3.3, we developed a Bayesian solution for a simple linear regression model under the assumption of Gaussian noise. We saw that the posterior distribution, which is Gaussian, could be evaluated exactly and that the predictive distribution could also be found in closed form. In the case of a multilayered network, the highly nonlinear dependence of the network function on the parameter values means that an exact Bayesian treatment can no longer be found. In fact, the log of the posterior distribution will be nonconvex, corresponding to the multiple local minima in the error function.

The technique of variational inference, to be discussed in Chapter 10, has been applied to Bayesian neural networks using a factorized Gaussian approximation

Exercise 5.37

to the posterior distribution (Hinton and van Camp, 1993) and also using a full-covariance Gaussian (Barber and Bishop, 1998a; Barber and Bishop, 1998b). The most complete treatment, however, has been based on the Laplace approximation (MacKay, 1992c; MacKay, 1992b) and forms the basis for the discussion given here. We will approximate the posterior distribution by a Gaussian, centred at a mode of the true posterior. Furthermore, we shall assume that the covariance of this Gaussian is small so that the network function is approximately linear with respect to the parameters over the region of parameter space for which the posterior probability is significantly nonzero. With these two approximations, we will obtain models that are analogous to the linear regression and classification models discussed in earlier chapters and so we can exploit the results obtained there. We can then make use of the evidence framework to provide point estimates for the hyperparameters and to compare alternative models (for example, networks having different numbers of hidden units). To start with, we shall discuss the regression case and then later consider the modifications needed for solving classification tasks.

5.7.1 Posterior parameter distribution

Consider the problem of predicting a single continuous target variable t from a vector \mathbf{x} of inputs (the extension to multiple targets is straightforward). We shall suppose that the conditional distribution $p(t|\mathbf{x})$ is Gaussian, with an \mathbf{x} -dependent mean given by the output of a neural network model $y(\mathbf{x}, \mathbf{w})$, and with precision (inverse variance) β

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}). \quad (5.161)$$

Similarly, we shall choose a prior distribution over the weights \mathbf{w} that is Gaussian of the form

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}). \quad (5.162)$$

For an i.i.d. data set of N observations $\mathbf{x}_1, \dots, \mathbf{x}_N$, with a corresponding set of target values $\mathcal{D} = \{t_1, \dots, t_N\}$, the likelihood function is given by

$$p(\mathcal{D}|\mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}) \quad (5.163)$$

and so the resulting posterior distribution is then

$$p(\mathbf{w}|\mathcal{D}, \alpha, \beta) \propto p(\mathbf{w}|\alpha)p(\mathcal{D}|\mathbf{w}, \beta). \quad (5.164)$$

which, as a consequence of the nonlinear dependence of $y(\mathbf{x}, \mathbf{w})$ on \mathbf{w} , will be non-Gaussian.

We can find a Gaussian approximation to the posterior distribution by using the Laplace approximation. To do this, we must first find a (local) maximum of the posterior, and this must be done using iterative numerical optimization. As usual, it is convenient to maximize the logarithm of the posterior, which can be written in the

form

$$\ln p(\mathbf{w}|\mathcal{D}) = -\frac{\alpha}{2}\mathbf{w}^T\mathbf{w} - \frac{\beta}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \text{const} \quad (5.165)$$

which corresponds to a regularized sum-of-squares error function. Assuming for the moment that α and β are fixed, we can find a maximum of the posterior, which we denote \mathbf{w}_{MAP} , by standard nonlinear optimization algorithms such as conjugate gradients, using error backpropagation to evaluate the required derivatives.

Having found a mode \mathbf{w}_{MAP} , we can then build a local Gaussian approximation by evaluating the matrix of second derivatives of the negative log posterior distribution. From (5.165), this is given by

$$\mathbf{A} = -\nabla\nabla \ln p(\mathbf{w}|\mathcal{D}, \alpha, \beta) = \alpha\mathbf{I} + \beta\mathbf{H} \quad (5.166)$$

where \mathbf{H} is the Hessian matrix comprising the second derivatives of the sum-of-squares error function with respect to the components of \mathbf{w} . Algorithms for computing and approximating the Hessian were discussed in Section 5.4. The corresponding Gaussian approximation to the posterior is then given from (4.134) by

$$q(\mathbf{w}|\mathcal{D}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\text{MAP}}, \mathbf{A}^{-1}). \quad (5.167)$$

Similarly, the predictive distribution is obtained by marginalizing with respect to this posterior distribution

$$p(t|\mathbf{x}, \mathcal{D}) = \int p(t|\mathbf{x}, \mathbf{w})q(\mathbf{w}|\mathcal{D}) d\mathbf{w}. \quad (5.168)$$

However, even with the Gaussian approximation to the posterior, this integration is still analytically intractable due to the nonlinearity of the network function $y(\mathbf{x}, \mathbf{w})$ as a function of \mathbf{w} . To make progress, we now assume that the posterior distribution has small variance compared with the characteristic scales of \mathbf{w} over which $y(\mathbf{x}, \mathbf{w})$ is varying. This allows us to make a Taylor series expansion of the network function around \mathbf{w}_{MAP} and retain only the linear terms

$$y(\mathbf{x}, \mathbf{w}) \simeq y(\mathbf{x}, \mathbf{w}_{\text{MAP}}) + \mathbf{g}^T(\mathbf{w} - \mathbf{w}_{\text{MAP}}) \quad (5.169)$$

where we have defined

$$\mathbf{g} = \nabla_{\mathbf{w}} y(\mathbf{x}, \mathbf{w})|_{\mathbf{w}=\mathbf{w}_{\text{MAP}}}. \quad (5.170)$$

With this approximation, we now have a linear-Gaussian model with a Gaussian distribution for $p(\mathbf{w})$ and a Gaussian for $p(t|\mathbf{w})$ whose mean is a linear function of \mathbf{w} of the form

$$p(t|\mathbf{x}, \mathbf{w}, \beta) \simeq \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{\text{MAP}}) + \mathbf{g}^T(\mathbf{w} - \mathbf{w}_{\text{MAP}}), \beta^{-1}). \quad (5.171)$$

We can therefore make use of the general result (2.115) for the marginal $p(t)$ to give

$$p(t|\mathbf{x}, \mathcal{D}, \alpha, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{\text{MAP}}), \sigma^2(\mathbf{x})) \quad (5.172)$$

where the input-dependent variance is given by

$$\sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}^T \mathbf{A}^{-1} \mathbf{g}. \quad (5.173)$$

We see that the predictive distribution $p(t|\mathbf{x}, \mathcal{D})$ is a Gaussian whose mean is given by the network function $y(\mathbf{x}, \mathbf{w}_{\text{MAP}})$ with the parameter set to their MAP value. The variance has two terms, the first of which arises from the intrinsic noise on the target variable, whereas the second is an \mathbf{x} -dependent term that expresses the uncertainty in the interpolant due to the uncertainty in the model parameters \mathbf{w} . This should be compared with the corresponding predictive distribution for the linear regression model, given by (3.58) and (3.59).

5.7.2 Hyperparameter optimization

So far, we have assumed that the hyperparameters α and β are fixed and known. We can make use of the evidence framework, discussed in Section 3.5, together with the Gaussian approximation to the posterior obtained using the Laplace approximation, to obtain a practical procedure for choosing the values of such hyperparameters.

The marginal likelihood, or evidence, for the hyperparameters is obtained by integrating over the network weights

$$p(\mathcal{D}|\alpha, \beta) = \int p(\mathcal{D}|\mathbf{w}, \beta) p(\mathbf{w}|\alpha) d\mathbf{w}. \quad (5.174)$$

Exercise 5.39

This is easily evaluated by making use of the Laplace approximation result (4.135). Taking logarithms then gives

$$\ln p(\mathcal{D}|\alpha, \beta) \simeq -E(\mathbf{w}_{\text{MAP}}) - \frac{1}{2} \ln |\mathbf{A}| + \frac{W}{2} \ln \alpha + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) \quad (5.175)$$

where W is the total number of parameters in \mathbf{w} , and the regularized error function is defined by

$$E(\mathbf{w}_{\text{MAP}}) = \frac{\beta}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}_{\text{MAP}}) - t_n\}^2 + \frac{\alpha}{2} \mathbf{w}_{\text{MAP}}^T \mathbf{w}_{\text{MAP}}. \quad (5.176)$$

We see that this takes the same form as the corresponding result (3.86) for the linear regression model.

In the evidence framework, we make point estimates for α and β by maximizing $\ln p(\mathcal{D}|\alpha, \beta)$. Consider first the maximization with respect to α , which can be done by analogy with the linear regression case discussed in Section 3.5.2. We first define the eigenvalue equation

$$\beta \mathbf{H} \mathbf{u}_i = \lambda_i \mathbf{u}_i \quad (5.177)$$

where \mathbf{H} is the Hessian matrix comprising the second derivatives of the sum-of-squares error function, evaluated at $\mathbf{w} = \mathbf{w}_{\text{MAP}}$. By analogy with (3.92), we obtain

$$\alpha = \frac{\gamma}{\mathbf{w}_{\text{MAP}}^T \mathbf{w}_{\text{MAP}}} \quad (5.178)$$

Section 3.5.3

where γ represents the effective number of parameters and is defined by

$$\gamma = \sum_{i=1}^W \frac{\lambda_i}{\alpha + \lambda_i}. \quad (5.179)$$

Note that this result was exact for the linear regression case. For the nonlinear neural network, however, it ignores the fact that changes in α will cause changes in the Hessian \mathbf{H} , which in turn will change the eigenvalues. We have therefore implicitly ignored terms involving the derivatives of λ_i with respect to α .

Similarly, from (3.95) we see that maximizing the evidence with respect to β gives the re-estimation formula

$$\frac{1}{\beta} = \frac{1}{N - \gamma} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}_{MAP}) - t_n\}^2. \quad (5.180)$$

As with the linear model, we need to alternate between re-estimation of the hyperparameters α and β and updating of the posterior distribution. The situation with a neural network model is more complex, however, due to the multimodality of the posterior distribution. As a consequence, the solution for \mathbf{w}_{MAP} found by maximizing the log posterior will depend on the initialization of \mathbf{w} . Solutions that differ only as a consequence of the interchange and sign reversal symmetries in the hidden units are identical so far as predictions are concerned, and it is irrelevant which of the equivalent solutions is found. However, there may be inequivalent solutions as well, and these will generally yield different values for the optimized hyperparameters.

In order to compare different models, for example neural networks having different numbers of hidden units, we need to evaluate the model evidence $p(\mathcal{D})$. This can be approximated by taking (5.175) and substituting the values of α and β obtained from the iterative optimization of these hyperparameters. A more careful evaluation is obtained by marginalizing over α and β , again by making a Gaussian approximation (MacKay, 1992c; Bishop, 1995a). In either case, it is necessary to evaluate the determinant $|\mathbf{A}|$ of the Hessian matrix. This can be problematic in practice because the determinant, unlike the trace, is sensitive to the small eigenvalues that are often difficult to determine accurately.

The Laplace approximation is based on a local quadratic expansion around a mode of the posterior distribution over weights. We have seen in Section 5.1.1 that any given mode in a two-layer network is a member of a set of $M!2^M$ equivalent modes that differ by interchange and sign-change symmetries, where M is the number of hidden units. When comparing networks having different numbers of hidden units, this can be taken into account by multiplying the evidence by a factor of $M!2^M$.

5.7.3 Bayesian neural networks for classification

So far, we have used the Laplace approximation to develop a Bayesian treatment of neural network regression models. We now discuss the modifications to

Section 5.1.1

this framework that arise when it is applied to classification. Here we shall consider a network having a single logistic sigmoid output corresponding to a two-class classification problem. The extension to networks with multiclass softmax outputs is straightforward. We shall build extensively on the analogous results for linear classification models discussed in Section 4.5, and so we encourage the reader to familiarize themselves with that material before studying this section.

The log likelihood function for this model is given by

$$\ln p(\mathcal{D}|\mathbf{w}) = \sum_n = 1^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \quad (5.181)$$

where $t_n \in \{0, 1\}$ are the target values, and $y_n \equiv y(\mathbf{x}_n, \mathbf{w})$. Note that there is no hyperparameter β , because the data points are assumed to be correctly labelled. As before, the prior is taken to be an isotropic Gaussian of the form (5.162).

The first stage in applying the Laplace framework to this model is to initialize the hyperparameter α , and then to determine the parameter vector \mathbf{w} by maximizing the log posterior distribution. This is equivalent to minimizing the regularized error function

$$E(\mathbf{w}) = -\ln p(\mathcal{D}|\mathbf{w}) + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} \quad (5.182)$$

and can be achieved using error backpropagation combined with standard optimization algorithms, as discussed in Section 5.3.

Having found a solution \mathbf{w}_{MAP} for the weight vector, the next step is to evaluate the Hessian matrix \mathbf{H} comprising the second derivatives of the negative log likelihood function. This can be done, for instance, using the exact method of Section 5.4.5, or using the outer product approximation given by (5.85). The second derivatives of the negative log posterior can again be written in the form (5.166), and the Gaussian approximation to the posterior is then given by (5.167).

To optimize the hyperparameter α , we again maximize the marginal likelihood, which is easily shown to take the form

$$\ln p(\mathcal{D}|\alpha) \simeq -E(\mathbf{w}_{MAP}) - \frac{1}{2} \ln |\mathbf{A}| + \frac{W}{2} \ln \alpha + \text{const} \quad (5.183)$$

where the regularized error function is defined by

$$E(\mathbf{w}_{MAP}) = -\sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} + \frac{\alpha}{2} \mathbf{w}_{MAP}^T \mathbf{w}_{MAP} \quad (5.184)$$

in which $y_n \equiv y(\mathbf{x}_n, \mathbf{w}_{MAP})$. Maximizing this evidence function with respect to α again leads to the re-estimation equation given by (5.178).

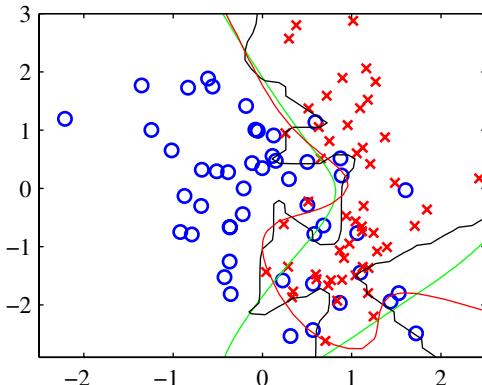
The use of the evidence procedure to determine α is illustrated in Figure 5.22 for the synthetic two-dimensional data discussed in Appendix A.

Finally, we need the predictive distribution, which is defined by (5.168). Again, this integration is intractable due to the nonlinearity of the network function. The

Exercise 5.40

Exercise 5.41

Figure 5.22 Illustration of the evidence framework applied to a synthetic two-class data set. The green curve shows the optimal decision boundary, the black curve shows the result of fitting a two-layer network with 8 hidden units by maximum likelihood, and the red curve shows the result of including a regularizer in which α is optimized using the evidence procedure, starting from the initial value $\alpha = 0$. Note that the evidence procedure greatly reduces the over-fitting of the network.



simplest approximation is to assume that the posterior distribution is very narrow and hence make the approximation

$$p(t|\mathbf{x}, \mathcal{D}) \simeq p(t|\mathbf{x}, \mathbf{w}_{\text{MAP}}). \quad (5.185)$$

We can improve on this, however, by taking account of the variance of the posterior distribution. In this case, a linear approximation for the network outputs, as was used in the case of regression, would be inappropriate due to the logistic sigmoid output-unit activation function that constrains the output to lie in the range $(0, 1)$. Instead, we make a linear approximation for the output unit activation in the form

$$a(\mathbf{x}, \mathbf{w}) \simeq a_{\text{MAP}}(\mathbf{x}) + \mathbf{b}^T(\mathbf{w} - \mathbf{w}_{\text{MAP}}) \quad (5.186)$$

where $a_{\text{MAP}}(\mathbf{x}) = a(\mathbf{x}, \mathbf{w}_{\text{MAP}})$, and the vector $\mathbf{b} \equiv \nabla a(\mathbf{x}, \mathbf{w}_{\text{MAP}})$ can be found by backpropagation.

Because we now have a Gaussian approximation for the posterior distribution over \mathbf{w} , and a model for a that is a linear function of \mathbf{w} , we can now appeal to the results of Section 4.5.2. The distribution of output unit activation values, induced by the distribution over network weights, is given by

$$p(a|\mathbf{x}, \mathcal{D}) = \int \delta(a - a_{\text{MAP}}(\mathbf{x}) - \mathbf{b}^T(\mathbf{x})(\mathbf{w} - \mathbf{w}_{\text{MAP}})) q(\mathbf{w}|\mathcal{D}) d\mathbf{w} \quad (5.187)$$

where $q(\mathbf{w}|\mathcal{D})$ is the Gaussian approximation to the posterior distribution given by (5.167). From Section 4.5.2, we see that this distribution is Gaussian with mean $a_{\text{MAP}} \equiv a(\mathbf{x}, \mathbf{w}_{\text{MAP}})$, and variance

$$\sigma_a^2(\mathbf{x}) = \mathbf{b}^T(\mathbf{x}) \mathbf{A}^{-1} \mathbf{b}(\mathbf{x}). \quad (5.188)$$

Finally, to obtain the predictive distribution, we must marginalize over a using

$$p(t = 1|\mathbf{x}, \mathcal{D}) = \int \sigma(a)p(a|\mathbf{x}, \mathcal{D}) da. \quad (5.189)$$

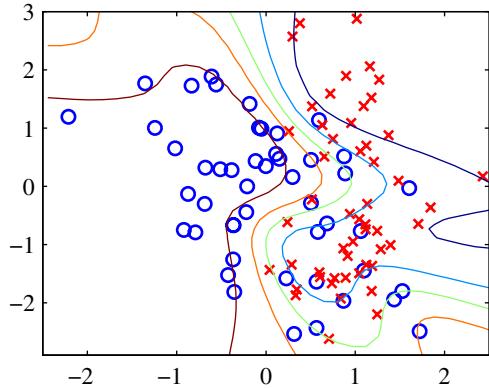
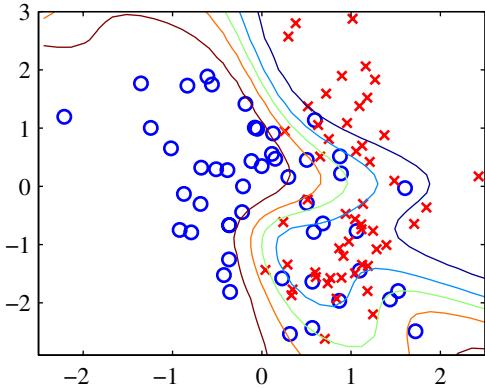


Figure 5.23 An illustration of the Laplace approximation for a Bayesian neural network having 8 hidden units with ‘tanh’ activation functions and a single logistic-sigmoid output unit. The weight parameters were found using scaled conjugate gradients, and the hyperparameter α was optimized using the evidence framework. On the left is the result of using the simple approximation (5.185) based on a point estimate \mathbf{w}_{MAP} of the parameters, in which the green curve shows the $y = 0.5$ decision boundary, and the other contours correspond to output probabilities of $y = 0.1, 0.3, 0.7$, and 0.9 . On the right is the corresponding result obtained using (5.190). Note that the effect of marginalization is to spread out the contours and to make the predictions less confident, so that at each input point \mathbf{x} , the posterior probabilities are shifted towards 0.5, while the $y = 0.5$ contour itself is unaffected.

The convolution of a Gaussian with a logistic sigmoid is intractable. We therefore apply the approximation (4.153) to (5.189) giving

$$p(t = 1 | \mathbf{x}, \mathcal{D}) = \sigma(\kappa(\sigma_a^2) \mathbf{b}^T \mathbf{w}_{\text{MAP}}) \quad (5.190)$$

where $\kappa(\cdot)$ is defined by (4.154). Recall that both σ_a^2 and \mathbf{b} are functions of \mathbf{x} .

Figure 5.23 shows an example of this framework applied to the synthetic classification data set described in Appendix A.

Exercises

- 5.1 (**) Consider a two-layer network function of the form (5.7) in which the hidden-unit nonlinear activation functions $g(\cdot)$ are given by logistic sigmoid functions of the form

$$\sigma(a) = \{1 + \exp(-a)\}^{-1}. \quad (5.191)$$

Show that there exists an equivalent network, which computes exactly the same function, but with hidden unit activation functions given by $\tanh(a)$ where the \tanh function is defined by (5.59). Hint: first find the relation between $\sigma(a)$ and $\tanh(a)$, and then show that the parameters of the two networks differ by linear transformations.

- 5.2 (*) **www** Show that maximizing the likelihood function under the conditional distribution (5.16) for a multioutput neural network is equivalent to minimizing the sum-of-squares error function (5.11).

- 5.3** (**) Consider a regression problem involving multiple target variables in which it is assumed that the distribution of the targets, conditioned on the input vector \mathbf{x} , is a Gaussian of the form

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x}, \mathbf{w}), \boldsymbol{\Sigma}) \quad (5.192)$$

where $\mathbf{y}(\mathbf{x}, \mathbf{w})$ is the output of a neural network with input vector \mathbf{x} and weight vector \mathbf{w} , and $\boldsymbol{\Sigma}$ is the covariance of the assumed Gaussian noise on the targets. Given a set of independent observations of \mathbf{x} and \mathbf{t} , write down the error function that must be minimized in order to find the maximum likelihood solution for \mathbf{w} , if we assume that $\boldsymbol{\Sigma}$ is fixed and known. Now assume that $\boldsymbol{\Sigma}$ is also to be determined from the data, and write down an expression for the maximum likelihood solution for $\boldsymbol{\Sigma}$. Note that the optimizations of \mathbf{w} and $\boldsymbol{\Sigma}$ are now coupled, in contrast to the case of independent target variables discussed in Section 5.2.

- 5.4** (**) Consider a binary classification problem in which the target values are $t \in \{0, 1\}$, with a network output $y(\mathbf{x}, \mathbf{w})$ that represents $p(t = 1|\mathbf{x})$, and suppose that there is a probability ϵ that the class label on a training data point has been incorrectly set. Assuming independent and identically distributed data, write down the error function corresponding to the negative log likelihood. Verify that the error function (5.21) is obtained when $\epsilon = 0$. Note that this error function makes the model robust to incorrectly labelled data, in contrast to the usual error function.

- 5.5** (*) **www** Show that maximizing likelihood for a multiclass neural network model in which the network outputs have the interpretation $y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1|\mathbf{x})$ is equivalent to the minimization of the cross-entropy error function (5.24).

- 5.6** (*) **www** Show the derivative of the error function (5.21) with respect to the activation a_k for an output unit having a logistic sigmoid activation function satisfies (5.18).

- 5.7** (*) Show the derivative of the error function (5.24) with respect to the activation a_k for output units having a softmax activation function satisfies (5.18).

- 5.8** (*) We saw in (4.88) that the derivative of the logistic sigmoid activation function can be expressed in terms of the function value itself. Derive the corresponding result for the ‘tanh’ activation function defined by (5.59).

- 5.9** (*) **www** The error function (5.21) for binary classification problems was derived for a network having a logistic-sigmoid output activation function, so that $0 \leq y(\mathbf{x}, \mathbf{w}) \leq 1$, and data having target values $t \in \{0, 1\}$. Derive the corresponding error function if we consider a network having an output $-1 \leq y(\mathbf{x}, \mathbf{w}) \leq 1$ and target values $t = 1$ for class C_1 and $t = -1$ for class C_2 . What would be the appropriate choice of output unit activation function?

- 5.10** (*) **www** Consider a Hessian matrix \mathbf{H} with eigenvector equation (5.33). By setting the vector \mathbf{v} in (5.39) equal to each of the eigenvectors \mathbf{u}_i in turn, show that \mathbf{H} is positive definite if, and only if, all of its eigenvalues are positive.

- 5.11** (**) **www** Consider a quadratic error function defined by (5.32), in which the Hessian matrix \mathbf{H} has an eigenvalue equation given by (5.33). Show that the contours of constant error are ellipses whose axes are aligned with the eigenvectors \mathbf{u}_i , with lengths that are inversely proportional to the square root of the corresponding eigenvalues λ_i .
- 5.12** (**) **www** By considering the local Taylor expansion (5.32) of an error function about a stationary point \mathbf{w}^* , show that the necessary and sufficient condition for the stationary point to be a local minimum of the error function is that the Hessian matrix \mathbf{H} , defined by (5.30) with $\hat{\mathbf{w}} = \mathbf{w}^*$, be positive definite.
- 5.13** (*) Show that as a consequence of the symmetry of the Hessian matrix \mathbf{H} , the number of independent elements in the quadratic error function (5.28) is given by $W(W + 3)/2$.
- 5.14** (*) By making a Taylor expansion, verify that the terms that are $O(\epsilon)$ cancel on the right-hand side of (5.69).
- 5.15** (**) In Section 5.3.4, we derived a procedure for evaluating the Jacobian matrix of a neural network using a backpropagation procedure. Derive an alternative formalism for finding the Jacobian based on *forward propagation* equations.
- 5.16** (*) The outer product approximation to the Hessian matrix for a neural network using a sum-of-squares error function is given by (5.84). Extend this result to the case of multiple outputs.
- 5.17** (*) Consider a squared loss function of the form
- $$E = \frac{1}{2} \iint \{y(\mathbf{x}, \mathbf{w}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt \quad (5.193)$$
- where $y(\mathbf{x}, \mathbf{w})$ is a parametric function such as a neural network. The result (1.89) shows that the function $y(\mathbf{x}, \mathbf{w})$ that minimizes this error is given by the conditional expectation of t given \mathbf{x} . Use this result to show that the second derivative of E with respect to two elements w_r and w_s of the vector \mathbf{w} , is given by
- $$\frac{\partial^2 E}{\partial w_r \partial w_s} = \int \frac{\partial y}{\partial w_r} \frac{\partial y}{\partial w_s} p(\mathbf{x}) d\mathbf{x}. \quad (5.194)$$
- Note that, for a finite sample from $p(\mathbf{x})$, we obtain (5.84).
- 5.18** (*) Consider a two-layer network of the form shown in Figure 5.1 with the addition of extra parameters corresponding to skip-layer connections that go directly from the inputs to the outputs. By extending the discussion of Section 5.3.2, write down the equations for the derivatives of the error function with respect to these additional parameters.
- 5.19** (*) **www** Derive the expression (5.85) for the outer product approximation to the Hessian matrix for a network having a single output with a logistic sigmoid output-unit activation function and a cross-entropy error function, corresponding to the result (5.84) for the sum-of-squares error function.

- 5.20** (*) Derive an expression for the outer product approximation to the Hessian matrix for a network having K outputs with a softmax output-unit activation function and a cross-entropy error function, corresponding to the result (5.84) for the sum-of-squares error function.
- 5.21** (***) Extend the expression (5.86) for the outer product approximation of the Hessian matrix to the case of $K > 1$ output units. Hence, derive a recursive expression analogous to (5.87) for incrementing the number N of patterns and a similar expression for incrementing the number K of outputs. Use these results, together with the identity (5.88), to find sequential update expressions analogous to (5.89) for finding the inverse of the Hessian by incrementally including both extra patterns and extra outputs.
- 5.22** (**) Derive the results (5.93), (5.94), and (5.95) for the elements of the Hessian matrix of a two-layer feed-forward network by application of the chain rule of calculus.
- 5.23** (**) Extend the results of Section 5.4.5 for the exact Hessian of a two-layer network to include skip-layer connections that go directly from inputs to outputs.
- 5.24** (*) Verify that the network function defined by (5.113) and (5.114) is invariant under the transformation (5.115) applied to the inputs, provided the weights and biases are simultaneously transformed using (5.116) and (5.117). Similarly, show that the network outputs can be transformed according (5.118) by applying the transformation (5.119) and (5.120) to the second-layer weights and biases.
- 5.25** (***) **www** Consider a quadratic error function of the form

$$E = E_0 + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H} (\mathbf{w} - \mathbf{w}^*) \quad (5.195)$$

where \mathbf{w}^* represents the minimum, and the Hessian matrix \mathbf{H} is positive definite and constant. Suppose the initial weight vector $\mathbf{w}^{(0)}$ is chosen to be at the origin and is updated using simple gradient descent

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \rho \nabla E \quad (5.196)$$

where τ denotes the step number, and ρ is the learning rate (which is assumed to be small). Show that, after τ steps, the components of the weight vector parallel to the eigenvectors of \mathbf{H} can be written

$$w_j^{(\tau)} = \{1 - (1 - \rho \eta_j)^\tau\} w_j^* \quad (5.197)$$

where $w_j = \mathbf{w}^T \mathbf{u}_j$, and \mathbf{u}_j and η_j are the eigenvectors and eigenvalues, respectively, of \mathbf{H} so that

$$\mathbf{H} \mathbf{u}_j = \eta_j \mathbf{u}_j. \quad (5.198)$$

Show that as $\tau \rightarrow \infty$, this gives $\mathbf{w}^{(\tau)} \rightarrow \mathbf{w}^*$ as expected, provided $|1 - \rho \eta_j| < 1$. Now suppose that training is halted after a finite number τ of steps. Show that the

components of the weight vector parallel to the eigenvectors of the Hessian satisfy

$$w_j^{(\tau)} \simeq w_j^* \quad \text{when} \quad \eta_j \gg (\rho\tau)^{-1} \quad (5.199)$$

$$|w_j^{(\tau)}| \ll |w_j^*| \quad \text{when} \quad \eta_j \ll (\rho\tau)^{-1}. \quad (5.200)$$

Compare this result with the discussion in Section 3.5.3 of regularization with simple weight decay, and hence show that $(\rho\tau)^{-1}$ is analogous to the regularization parameter λ . The above results also show that the effective number of parameters in the network, as defined by (3.91), grows as the training progresses.

- 5.26** (**) Consider a multilayer perceptron with arbitrary feed-forward topology, which is to be trained by minimizing the *tangent propagation* error function (5.127) in which the regularizing function is given by (5.128). Show that the regularization term Ω can be written as a sum over patterns of terms of the form

$$\Omega_n = \frac{1}{2} \sum_k (\mathcal{G}y_k)^2 \quad (5.201)$$

where \mathcal{G} is a differential operator defined by

$$\mathcal{G} \equiv \sum_i \tau_i \frac{\partial}{\partial x_i}. \quad (5.202)$$

By acting on the forward propagation equations

$$z_j = h(a_j), \quad a_j = \sum_i w_{ji} z_i \quad (5.203)$$

with the operator \mathcal{G} , show that Ω_n can be evaluated by forward propagation using the following equations:

$$\alpha_j = h'(a_j) \beta_j, \quad \beta_j = \sum_i w_{ji} \alpha_i. \quad (5.204)$$

where we have defined the new variables

$$\alpha_j \equiv \mathcal{G}z_j, \quad \beta_j \equiv \mathcal{G}a_j. \quad (5.205)$$

Now show that the derivatives of Ω_n with respect to a weight w_{rs} in the network can be written in the form

$$\frac{\partial \Omega_n}{\partial w_{rs}} = \sum_k \alpha_k \{ \phi_{kr} z_s + \delta_{kr} \alpha_s \} \quad (5.206)$$

where we have defined

$$\delta_{kr} \equiv \frac{\partial y_k}{\partial a_r}, \quad \phi_{kr} \equiv \mathcal{G}\delta_{kr}. \quad (5.207)$$

Write down the backpropagation equations for δ_{kr} , and hence derive a set of back-propagation equations for the evaluation of the ϕ_{kr} .

5.27 (**) **www** Consider the framework for training with transformed data in the special case in which the transformation consists simply of the addition of random noise $\mathbf{x} \rightarrow \mathbf{x} + \boldsymbol{\xi}$ where $\boldsymbol{\xi}$ has a Gaussian distribution with zero mean and unit covariance. By following an argument analogous to that of Section 5.5.5, show that the resulting regularizer reduces to the Tikhonov form (5.135).

5.28 (*) **www** Consider a neural network, such as the convolutional network discussed in Section 5.5.6, in which multiple weights are constrained to have the same value. Discuss how the standard backpropagation algorithm must be modified in order to ensure that such constraints are satisfied when evaluating the derivatives of an error function with respect to the adjustable parameters in the network.

5.29 (*) **www** Verify the result (5.141).

5.30 (*) Verify the result (5.142).

5.31 (*) Verify the result (5.143).

5.32 (**) Show that the derivatives of the mixing coefficients $\{\pi_k\}$, defined by (5.146), with respect to the auxiliary parameters $\{\eta_j\}$ are given by

$$\frac{\partial \pi_k}{\partial \eta_j} = \delta_{jk} \pi_j - \pi_j \pi_k. \quad (5.208)$$

Hence, by making use of the constraint $\sum_k \pi_k = 1$, derive the result (5.147).

5.33 (*) Write down a pair of equations that express the Cartesian coordinates (x_1, x_2) for the robot arm shown in Figure 5.18 in terms of the joint angles θ_1 and θ_2 and the lengths L_1 and L_2 of the links. Assume the origin of the coordinate system is given by the attachment point of the lower arm. These equations define the ‘forward kinematics’ of the robot arm.

5.34 (*) **www** Derive the result (5.155) for the derivative of the error function with respect to the network output activations controlling the mixing coefficients in the mixture density network.

5.35 (*) Derive the result (5.156) for the derivative of the error function with respect to the network output activations controlling the component means in the mixture density network.

5.36 (*) Derive the result (5.157) for the derivative of the error function with respect to the network output activations controlling the component variances in the mixture density network.

5.37 (*) Verify the results (5.158) and (5.160) for the conditional mean and variance of the mixture density network model.

5.38 (*) Using the general result (2.115), derive the predictive distribution (5.172) for the Laplace approximation to the Bayesian neural network model.

- 5.39** (*) **www** Make use of the Laplace approximation result (4.135) to show that the evidence function for the hyperparameters α and β in the Bayesian neural network model can be approximated by (5.175).
- 5.40** (*) **www** Outline the modifications needed to the framework for Bayesian neural networks, discussed in Section 5.7.3, to handle multiclass problems using networks having softmax output-unit activation functions.
- 5.41** (**) By following analogous steps to those given in Section 5.7.1 for regression networks, derive the result (5.183) for the marginal likelihood in the case of a network having a cross-entropy error function and logistic-sigmoid output-unit activation function.

6

Kernel Methods

In Chapters 3 and 4, we considered linear parametric models for regression and classification in which the form of the mapping $y(\mathbf{x}, \mathbf{w})$ from input \mathbf{x} to output y is governed by a vector \mathbf{w} of adaptive parameters. During the learning phase, a set of training data is used either to obtain a point estimate of the parameter vector or to determine a posterior distribution over this vector. The training data is then discarded, and predictions for new inputs are based purely on the learned parameter vector \mathbf{w} . This approach is also used in nonlinear parametric models such as neural networks.

However, there is a class of pattern recognition techniques, in which the training data points, or a subset of them, are kept and used also during the prediction phase. For instance, the Parzen probability density model comprised a linear combination of ‘kernel’ functions each one centred on one of the training data points. Similarly, in Section 2.5.2 we introduced a simple technique for classification called nearest neighbours, which involved assigning to each new test vector the same label as the

closest example from the training set. These are examples of *memory-based* methods that involve storing the entire training set in order to make predictions for future data points. They typically require a metric to be defined that measures the similarity of any two vectors in input space, and are generally fast to ‘train’ but slow at making predictions for test data points.

Many linear parametric models can be re-cast into an equivalent ‘dual representation’ in which the predictions are also based on linear combinations of a *kernel function* evaluated at the training data points. As we shall see, for models which are based on a fixed nonlinear *feature space* mapping $\phi(\mathbf{x})$, the kernel function is given by the relation

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}'). \quad (6.1)$$

From this definition, we see that the kernel is a symmetric function of its arguments so that $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$. The kernel concept was introduced into the field of pattern recognition by Aizerman *et al.* (1964) in the context of the method of potential functions, so-called because of an analogy with electrostatics. Although neglected for many years, it was re-introduced into machine learning in the context of large-margin classifiers by Boser *et al.* (1992) giving rise to the technique of *support vector machines*. Since then, there has been considerable interest in this topic, both in terms of theory and applications. One of the most significant developments has been the extension of kernels to handle symbolic objects, thereby greatly expanding the range of problems that can be addressed.

The simplest example of a kernel function is obtained by considering the identity mapping for the feature space in (6.1) so that $\phi(\mathbf{x}) = \mathbf{x}$, in which case $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$. We shall refer to this as the linear kernel.

The concept of a kernel formulated as an inner product in a feature space allows us to build interesting extensions of many well-known algorithms by making use of the *kernel trick*, also known as *kernel substitution*. The general idea is that, if we have an algorithm formulated in such a way that the input vector \mathbf{x} enters only in the form of scalar products, then we can replace that scalar product with some other choice of kernel. For instance, the technique of kernel substitution can be applied to principal component analysis in order to develop a nonlinear variant of PCA (Schölkopf *et al.*, 1998). Other examples of kernel substitution include nearest-neighbour classifiers and the kernel Fisher discriminant (Mika *et al.*, 1999; Roth and Steinhage, 2000; Baudat and Anouar, 2000).

There are numerous forms of kernel functions in common use, and we shall encounter several examples in this chapter. Many have the property of being a function only of the difference between the arguments, so that $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$, which are known as *stationary* kernels because they are invariant to translations in input space. A further specialization involves *homogeneous* kernels, also known as *radial basis functions*, which depend only on the magnitude of the distance (typically Euclidean) between the arguments so that $k(\mathbf{x}, \mathbf{x}') = k(\|\mathbf{x} - \mathbf{x}'\|)$.

For recent textbooks on kernel methods, see Schölkopf and Smola (2002), Herbrich (2002), and Shawe-Taylor and Cristianini (2004).

Chapter 7

Section 12.3

Section 6.3

6.1. Dual Representations

Many linear models for regression and classification can be reformulated in terms of a dual representation in which the kernel function arises naturally. This concept will play an important role when we consider support vector machines in the next chapter. Here we consider a linear regression model whose parameters are determined by minimizing a regularized sum-of-squares error function given by

$$J(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{\mathbf{w}^T \phi(\mathbf{x}_n) - t_n\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \quad (6.2)$$

where $\lambda \geq 0$. If we set the gradient of $J(\mathbf{w})$ with respect to \mathbf{w} equal to zero, we see that the solution for \mathbf{w} takes the form of a linear combination of the vectors $\phi(\mathbf{x}_n)$, with coefficients that are functions of \mathbf{w} , of the form

$$\mathbf{w} = -\frac{1}{\lambda} \sum_{n=1}^N \{\mathbf{w}^T \phi(\mathbf{x}_n) - t_n\} \phi(\mathbf{x}_n) = \sum_{n=1}^N a_n \phi(\mathbf{x}_n) = \Phi^T \mathbf{a} \quad (6.3)$$

where Φ is the design matrix, whose n^{th} row is given by $\phi(\mathbf{x}_n)^T$. Here the vector $\mathbf{a} = (a_1, \dots, a_N)^T$, and we have defined

$$a_n = -\frac{1}{\lambda} \{\mathbf{w}^T \phi(\mathbf{x}_n) - t_n\}. \quad (6.4)$$

Instead of working with the parameter vector \mathbf{w} , we can now reformulate the least-squares algorithm in terms of the parameter vector \mathbf{a} , giving rise to a *dual representation*. If we substitute $\mathbf{w} = \Phi^T \mathbf{a}$ into $J(\mathbf{w})$, we obtain

$$J(\mathbf{a}) = \frac{1}{2} \mathbf{a}^T \Phi \Phi^T \Phi \Phi^T \mathbf{a} - \mathbf{a}^T \Phi \Phi^T \mathbf{t} + \frac{1}{2} \mathbf{t}^T \mathbf{t} + \frac{\lambda}{2} \mathbf{a}^T \Phi \Phi^T \mathbf{a} \quad (6.5)$$

where $\mathbf{t} = (t_1, \dots, t_N)^T$. We now define the *Gram matrix* $\mathbf{K} = \Phi \Phi^T$, which is an $N \times N$ symmetric matrix with elements

$$K_{nm} = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) \quad (6.6)$$

where we have introduced the *kernel function* $k(\mathbf{x}, \mathbf{x}')$ defined by (6.1). In terms of the Gram matrix, the sum-of-squares error function can be written as

$$J(\mathbf{a}) = \frac{1}{2} \mathbf{a}^T \mathbf{K} \mathbf{K} \mathbf{a} - \mathbf{a}^T \mathbf{K} \mathbf{t} + \frac{1}{2} \mathbf{t}^T \mathbf{t} + \frac{\lambda}{2} \mathbf{a}^T \mathbf{K} \mathbf{a}. \quad (6.7)$$

Setting the gradient of $J(\mathbf{a})$ with respect to \mathbf{a} to zero, we obtain the following solution

$$\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{t}. \quad (6.8)$$

If we substitute this back into the linear regression model, we obtain the following prediction for a new input \mathbf{x}

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) = \mathbf{a}^T \Phi \phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{t} \quad (6.9)$$

where we have defined the vector $\mathbf{k}(\mathbf{x})$ with elements $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$. Thus we see that the dual formulation allows the solution to the least-squares problem to be expressed entirely in terms of the kernel function $k(\mathbf{x}, \mathbf{x}')$. This is known as a dual formulation because, by noting that the solution for \mathbf{a} can be expressed as a linear combination of the elements of $\phi(\mathbf{x})$, we recover the original formulation in terms of the parameter vector \mathbf{w} . Note that the prediction at \mathbf{x} is given by a linear combination of the target values from the training set. In fact, we have already obtained this result, using a slightly different notation, in Section 3.3.3.

In the dual formulation, we determine the parameter vector \mathbf{a} by inverting an $N \times N$ matrix, whereas in the original parameter space formulation we had to invert an $M \times M$ matrix in order to determine \mathbf{w} . Because N is typically much larger than M , the dual formulation does not seem to be particularly useful. However, the advantage of the dual formulation, as we shall see, is that it is expressed entirely in terms of the kernel function $k(\mathbf{x}, \mathbf{x}')$. We can therefore work directly in terms of kernels and avoid the explicit introduction of the feature vector $\phi(\mathbf{x})$, which allows us implicitly to use feature spaces of high, even infinite, dimensionality.

Exercise 6.1

The existence of a dual representation based on the Gram matrix is a property of many linear models, including the perceptron. In Section 6.4, we will develop a duality between probabilistic linear models for regression and the technique of Gaussian processes. Duality will also play an important role when we discuss support vector machines in Chapter 7.

6.2. Constructing Kernels

In order to exploit kernel substitution, we need to be able to construct valid kernel functions. One approach is to choose a feature space mapping $\phi(\mathbf{x})$ and then use this to find the corresponding kernel, as is illustrated in Figure 6.1. Here the kernel function is defined for a one-dimensional input space by

$$k(x, x') = \phi(x)^T \phi(x') = \sum_{i=1}^M \phi_i(x) \phi_i(x') \quad (6.10)$$

where $\phi_i(x)$ are the basis functions.

An alternative approach is to construct kernel functions directly. In this case, we must ensure that the function we choose is a valid kernel, in other words that it corresponds to a scalar product in some (perhaps infinite dimensional) feature space. As a simple example, consider a kernel function given by

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2. \quad (6.11)$$

Exercise 6.2

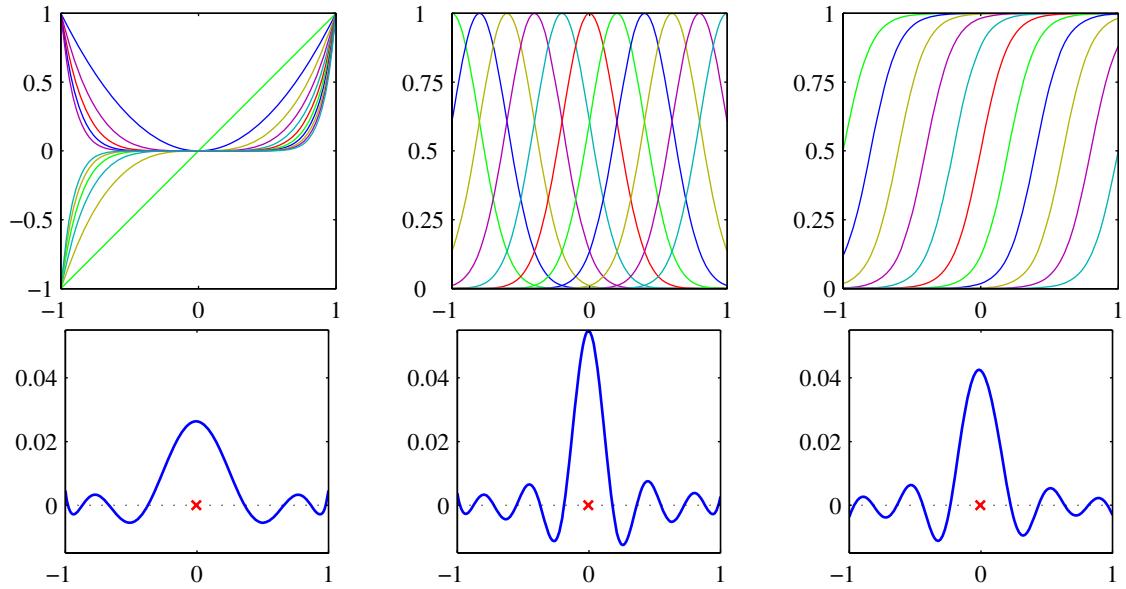


Figure 6.1 Illustration of the construction of kernel functions starting from a corresponding set of basis functions. In each column the lower plot shows the kernel function $k(x, x')$ defined by (6.10) plotted as a function of x for $x' = 0$, while the upper plot shows the corresponding basis functions given by polynomials (left column), ‘Gaussians’ (centre column), and logistic sigmoids (right column).

If we take the particular case of a two-dimensional input space $\mathbf{x} = (x_1, x_2)$ we can expand out the terms and thereby identify the corresponding nonlinear feature mapping

$$\begin{aligned}
 k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z})^2 = (x_1 z_1 + x_2 z_2)^2 \\
 &= x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\
 &= (x_1^2, \sqrt{2}x_1 x_2, x_2^2)(z_1^2, \sqrt{2}z_1 z_2, z_2^2)^T \\
 &= \phi(\mathbf{x})^T \phi(\mathbf{z}).
 \end{aligned} \tag{6.12}$$

We see that the feature mapping takes the form $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)^T$ and therefore comprises all possible second order terms, with a specific weighting between them.

More generally, however, we need a simple way to test whether a function constitutes a valid kernel without having to construct the function $\phi(\mathbf{x})$ explicitly. A necessary and sufficient condition for a function $k(\mathbf{x}, \mathbf{x}')$ to be a valid kernel (Shawe-Taylor and Cristianini, 2004) is that the Gram matrix \mathbf{K} , whose elements are given by $k(\mathbf{x}_n, \mathbf{x}_m)$, should be positive semidefinite for all possible choices of the set $\{\mathbf{x}_n\}$. Note that a positive semidefinite matrix is not the same thing as a matrix whose elements are nonnegative.

One powerful technique for constructing new kernels is to build them out of simpler kernels as building blocks. This can be done using the following properties:

Techniques for Constructing New Kernels.

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') \quad (6.13)$$

$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \quad (6.14)$$

$$k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}')) \quad (6.15)$$

$$k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}')) \quad (6.16)$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \quad (6.17)$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \quad (6.18)$$

$$k(\mathbf{x}, \mathbf{x}') = k_3(\phi(\mathbf{x}), \phi(\mathbf{x}')) \quad (6.19)$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{A} \mathbf{x}' \quad (6.20)$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \quad (6.21)$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b) \quad (6.22)$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\phi(\mathbf{x})$ is a function from \mathbf{x} to \mathbb{R}^M , $k_3(\cdot, \cdot)$ is a valid kernel in \mathbb{R}^M , \mathbf{A} is a symmetric positive semidefinite matrix, \mathbf{x}_a and \mathbf{x}_b are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

Equipped with these properties, we can now embark on the construction of more complex kernels appropriate to specific applications. We require that the kernel $k(\mathbf{x}, \mathbf{x}')$ be symmetric and positive semidefinite and that it expresses the appropriate form of similarity between \mathbf{x} and \mathbf{x}' according to the intended application. Here we consider a few common examples of kernel functions. For a more extensive discussion of ‘kernel engineering’, see Shawe-Taylor and Cristianini (2004).

We saw that the simple polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^2$ contains only terms of degree two. If we consider the slightly generalized kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^2$ with $c > 0$, then the corresponding feature mapping $\phi(\mathbf{x})$ contains constant and linear terms as well as terms of order two. Similarly, $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^M$ contains all monomials of order M . For instance, if \mathbf{x} and \mathbf{x}' are two images, then the kernel represents a particular weighted sum of all possible products of M pixels in the first image with M pixels in the second image. This can similarly be generalized to include all terms up to degree M by considering $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^M$ with $c > 0$. Using the results (6.17) and (6.18) for combining kernels we see that these will all be valid kernel functions.

Another commonly used kernel takes the form

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2) \quad (6.23)$$

and is often called a ‘Gaussian’ kernel. Note, however, that in this context it is not interpreted as a probability density, and hence the normalization coefficient is

omitted. We can see that this is a valid kernel by expanding the square

$$\|\mathbf{x} - \mathbf{x}'\|^2 = \mathbf{x}^T \mathbf{x} + (\mathbf{x}')^T \mathbf{x}' - 2\mathbf{x}^T \mathbf{x}' \quad (6.24)$$

to give

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\mathbf{x}^T \mathbf{x}/2\sigma^2) \exp(\mathbf{x}^T \mathbf{x}'/\sigma^2) \exp(-(\mathbf{x}')^T \mathbf{x}'/2\sigma^2) \quad (6.25)$$

and then making use of (6.14) and (6.16), together with the validity of the linear kernel $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$. Note that the feature vector that corresponds to the Gaussian kernel has infinite dimensionality.

The Gaussian kernel is not restricted to the use of Euclidean distance. If we use kernel substitution in (6.24) to replace $\mathbf{x}^T \mathbf{x}'$ with a nonlinear kernel $\kappa(\mathbf{x}, \mathbf{x}')$, we obtain

$$k(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{1}{2\sigma^2} (\kappa(\mathbf{x}, \mathbf{x}) + \kappa(\mathbf{x}', \mathbf{x}') - 2\kappa(\mathbf{x}, \mathbf{x}'))\right\}. \quad (6.26)$$

An important contribution to arise from the kernel viewpoint has been the extension to inputs that are symbolic, rather than simply vectors of real numbers. Kernel functions can be defined over objects as diverse as graphs, sets, strings, and text documents. Consider, for instance, a fixed set and define a nonvectorial space consisting of all possible subsets of this set. If A_1 and A_2 are two such subsets then one simple choice of kernel would be

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|} \quad (6.27)$$

where $A_1 \cap A_2$ denotes the intersection of sets A_1 and A_2 , and $|A|$ denotes the number of subsets in A . This is a valid kernel function because it can be shown to correspond to an inner product in a feature space.

One powerful approach to the construction of kernels starts from a probabilistic generative model (Haussler, 1999), which allows us to apply generative models in a discriminative setting. Generative models can deal naturally with missing data and in the case of hidden Markov models can handle sequences of varying length. By contrast, discriminative models generally give better performance on discriminative tasks than generative models. It is therefore of some interest to combine these two approaches (Lasserre *et al.*, 2006). One way to combine them is to use a generative model to define a kernel, and then use this kernel in a discriminative approach.

Given a generative model $p(\mathbf{x})$ we can define a kernel by

$$k(\mathbf{x}, \mathbf{x}') = p(\mathbf{x})p(\mathbf{x}'). \quad (6.28)$$

This is clearly a valid kernel function because we can interpret it as an inner product in the one-dimensional feature space defined by the mapping $p(\mathbf{x})$. It says that two inputs \mathbf{x} and \mathbf{x}' are similar if they both have high probabilities. We can use (6.13) and (6.17) to extend this class of kernels by considering sums over products of different probability distributions, with positive weighting coefficients $p(i)$, of the form

$$k(\mathbf{x}, \mathbf{x}') = \sum_i p(\mathbf{x}|i)p(\mathbf{x}'|i)p(i). \quad (6.29)$$

Exercise 6.11

Exercise 6.12

Section 9.2

This is equivalent, up to an overall multiplicative constant, to a mixture distribution in which the components factorize, with the index i playing the role of a ‘latent’ variable. Two inputs \mathbf{x} and \mathbf{x}' will give a large value for the kernel function, and hence appear similar, if they have significant probability under a range of different components. Taking the limit of an infinite sum, we can also consider kernels of the form

$$k(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{x}'|\mathbf{z})p(\mathbf{z}) d\mathbf{z} \quad (6.30)$$

where \mathbf{z} is a continuous latent variable.

Now suppose that our data consists of ordered sequences of length L so that an observation is given by $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_L\}$. A popular generative model for sequences is the hidden Markov model, which expresses the distribution $p(\mathbf{X})$ as a marginalization over a corresponding sequence of hidden states $\mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_L\}$. We can use this approach to define a kernel function measuring the similarity of two sequences \mathbf{X} and \mathbf{X}' by extending the mixture representation (6.29) to give

$$k(\mathbf{X}, \mathbf{X}') = \sum_{\mathbf{Z}} p(\mathbf{X}|\mathbf{Z})p(\mathbf{X}'|\mathbf{Z})p(\mathbf{Z}) \quad (6.31)$$

so that both observed sequences are generated by the same hidden sequence \mathbf{Z} . This model can easily be extended to allow sequences of differing length to be compared.

An alternative technique for using generative models to define kernel functions is known as the *Fisher kernel* (Jaakkola and Haussler, 1999). Consider a parametric generative model $p(\mathbf{x}|\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ denotes the vector of parameters. The goal is to find a kernel that measures the similarity of two input vectors \mathbf{x} and \mathbf{x}' induced by the generative model. Jaakkola and Haussler (1999) consider the gradient with respect to $\boldsymbol{\theta}$, which defines a vector in a ‘feature’ space having the same dimensionality as $\boldsymbol{\theta}$. In particular, they consider the *Fisher score*

$$\mathbf{g}(\boldsymbol{\theta}, \mathbf{x}) = \nabla_{\boldsymbol{\theta}} \ln p(\mathbf{x}|\boldsymbol{\theta}) \quad (6.32)$$

from which the Fisher kernel is defined by

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^T \mathbf{F}^{-1} \mathbf{g}(\boldsymbol{\theta}, \mathbf{x}'). \quad (6.33)$$

Here \mathbf{F} is the *Fisher information matrix*, given by

$$\mathbf{F} = \mathbb{E}_{\mathbf{x}} [\mathbf{g}(\boldsymbol{\theta}, \mathbf{x}) \mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^T] \quad (6.34)$$

where the expectation is with respect to \mathbf{x} under the distribution $p(\mathbf{x}|\boldsymbol{\theta})$. This can be motivated from the perspective of *information geometry* (Amari, 1998), which considers the differential geometry of the space of model parameters. Here we simply note that the presence of the Fisher information matrix causes this kernel to be invariant under a nonlinear re-parameterization of the density model $\boldsymbol{\theta} \rightarrow \psi(\boldsymbol{\theta})$.

In practice, it is often infeasible to evaluate the Fisher information matrix. One approach is simply to replace the expectation in the definition of the Fisher information with the sample average, giving

$$\mathbf{F} \simeq \frac{1}{N} \sum_{n=1}^N \mathbf{g}(\boldsymbol{\theta}, \mathbf{x}_n) \mathbf{g}(\boldsymbol{\theta}, \mathbf{x}_n)^T. \quad (6.35)$$

Exercise 6.13

Section 12.1.3

This is the covariance matrix of the Fisher scores, and so the Fisher kernel corresponds to a whitening of these scores. More simply, we can just omit the Fisher information matrix altogether and use the noninvariant kernel

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^T \mathbf{g}(\boldsymbol{\theta}, \mathbf{x}'). \quad (6.36)$$

An application of Fisher kernels to document retrieval is given by Hofmann (2000).

A final example of a kernel function is the sigmoidal kernel given by

$$k(\mathbf{x}, \mathbf{x}') = \tanh(a \mathbf{x}^T \mathbf{x}' + b) \quad (6.37)$$

whose Gram matrix in general is not positive semidefinite. This form of kernel has, however, been used in practice (Vapnik, 1995), possibly because it gives kernel expansions such as the support vector machine a superficial resemblance to neural network models. As we shall see, in the limit of an infinite number of basis functions, a Bayesian neural network with an appropriate prior reduces to a Gaussian process, thereby providing a deeper link between neural networks and kernel methods.

Section 6.4.7

6.3. Radial Basis Function Networks

In Chapter 3, we discussed regression models based on linear combinations of fixed basis functions, although we did not discuss in detail what form those basis functions might take. One choice that has been widely used is that of *radial basis functions*, which have the property that each basis function depends only on the radial distance (typically Euclidean) from a centre $\boldsymbol{\mu}_j$, so that $\phi_j(\mathbf{x}) = h(\|\mathbf{x} - \boldsymbol{\mu}_j\|)$.

Historically, radial basis functions were introduced for the purpose of exact function interpolation (Powell, 1987). Given a set of input vectors $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ along with corresponding target values $\{t_1, \dots, t_N\}$, the goal is to find a smooth function $f(\mathbf{x})$ that fits every target value exactly, so that $f(\mathbf{x}_n) = t_n$ for $n = 1, \dots, N$. This is achieved by expressing $f(\mathbf{x})$ as a linear combination of radial basis functions, one centred on every data point

$$f(\mathbf{x}) = \sum_{n=1}^N w_n h(\|\mathbf{x} - \mathbf{x}_n\|). \quad (6.38)$$

The values of the coefficients $\{w_n\}$ are found by least squares, and because there are the same number of coefficients as there are constraints, the result is a function that fits every target value exactly. In pattern recognition applications, however, the target values are generally noisy, and exact interpolation is undesirable because this corresponds to an over-fitted solution.

Expansions in radial basis functions also arise from regularization theory (Poggio and Girosi, 1990; Bishop, 1995a). For a sum-of-squares error function with a regularizer defined in terms of a differential operator, the optimal solution is given by an expansion in the *Green's functions* of the operator (which are analogous to the eigenvectors of a discrete matrix), again with one basis function centred on each data

point. If the differential operator is isotropic then the Green's functions depend only on the radial distance from the corresponding data point. Due to the presence of the regularizer, the solution no longer interpolates the training data exactly.

Another motivation for radial basis functions comes from a consideration of the interpolation problem when the input (rather than the target) variables are noisy (Webb, 1994; Bishop, 1995a). If the noise on the input variable \mathbf{x} is described by a variable ξ having a distribution $\nu(\xi)$, then the sum-of-squares error function becomes

$$E = \frac{1}{2} \sum_{n=1}^N \int \{y(\mathbf{x}_n + \xi) - t_n\}^2 \nu(\xi) d\xi. \quad (6.39)$$

Appendix D
Exercise 6.17

Using the calculus of variations, we can optimize with respect to the function $f(\mathbf{x})$ to give

$$y(\mathbf{x}_n) = \sum_{n=1}^N t_n h(\mathbf{x} - \mathbf{x}_n) \quad (6.40)$$

where the basis functions are given by

$$h(\mathbf{x} - \mathbf{x}_n) = \frac{\nu(\mathbf{x} - \mathbf{x}_n)}{\sum_{n=1}^N \nu(\mathbf{x} - \mathbf{x}_n)}. \quad (6.41)$$

We see that there is one basis function centred on every data point. This is known as the *Nadaraya-Watson* model and will be derived again from a different perspective in Section 6.3.1. If the noise distribution $\nu(\xi)$ is isotropic, so that it is a function only of $\|\xi\|$, then the basis functions will be radial.

Note that the basis functions (6.41) are normalized, so that $\sum_n h(\mathbf{x} - \mathbf{x}_n) = 1$ for any value of \mathbf{x} . The effect of such normalization is shown in Figure 6.2. Normalization is sometimes used in practice as it avoids having regions of input space where all of the basis functions take small values, which would necessarily lead to predictions in such regions that are either small or controlled purely by the bias parameter.

Another situation in which expansions in normalized radial basis functions arise is in the application of kernel density estimation to the problem of regression, as we shall discuss in Section 6.3.1.

Because there is one basis function associated with every data point, the corresponding model can be computationally costly to evaluate when making predictions for new data points. Models have therefore been proposed (Broomhead and Lowe, 1988; Moody and Darken, 1989; Poggio and Girosi, 1990), which retain the expansion in radial basis functions but where the number M of basis functions is smaller than the number N of data points. Typically, the number of basis functions, and the locations μ_i of their centres, are determined based on the input data $\{\mathbf{x}_n\}$ alone. The basis functions are then kept fixed and the coefficients $\{w_i\}$ are determined by least squares by solving the usual set of linear equations, as discussed in Section 3.1.1.

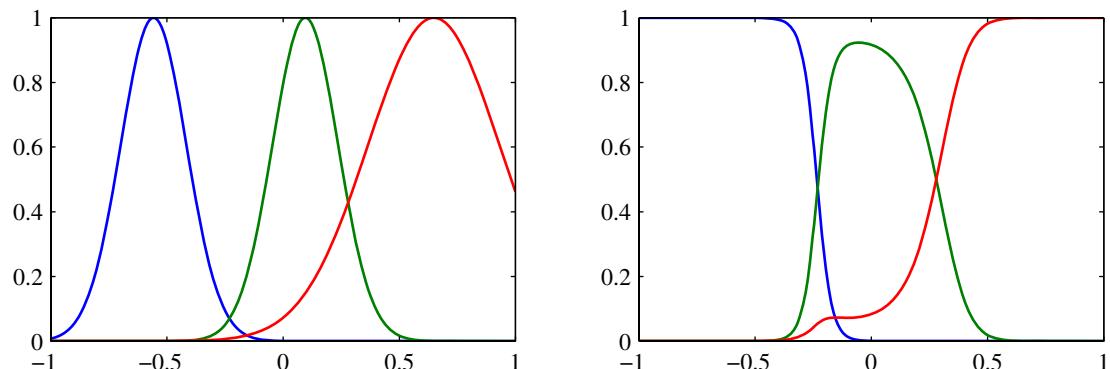


Figure 6.2 Plot of a set of Gaussian basis functions on the left, together with the corresponding normalized basis functions on the right.

One of the simplest ways of choosing basis function centres is to use a randomly chosen subset of the data points. A more systematic approach is called *orthogonal least squares* (Chen *et al.*, 1991). This is a sequential selection process in which at each step the next data point to be chosen as a basis function centre corresponds to the one that gives the greatest reduction in the sum-of-squares error. Values for the expansion coefficients are determined as part of the algorithm. Clustering algorithms such as K -means have also been used, which give a set of basis function centres that no longer coincide with training data points.

Section 9.1

6.3.1 Nadaraya-Watson model

In Section 3.3.3, we saw that the prediction of a linear regression model for a new input \mathbf{x} takes the form of a linear combination of the training set target values with coefficients given by the ‘equivalent kernel’ (3.62) where the equivalent kernel satisfies the summation constraint (3.64).

We can motivate the kernel regression model (3.61) from a different perspective, starting with kernel density estimation. Suppose we have a training set $\{\mathbf{x}_n, t_n\}$ and we use a Parzen density estimator to model the joint distribution $p(\mathbf{x}, t)$, so that

$$p(\mathbf{x}, t) = \frac{1}{N} \sum_{n=1}^N f(\mathbf{x} - \mathbf{x}_n, t - t_n) \quad (6.42)$$

Section 2.5.1

where $f(\mathbf{x}, t)$ is the component density function, and there is one such component centred on each data point. We now find an expression for the regression function $y(\mathbf{x})$, corresponding to the conditional average of the target variable conditioned on

the input variable, which is given by

$$\begin{aligned}
 y(\mathbf{x}) &= \mathbb{E}[t|\mathbf{x}] = \int_{-\infty}^{\infty} tp(t|\mathbf{x}) dt \\
 &= \frac{\int tp(\mathbf{x}, t) dt}{\int p(\mathbf{x}, t) dt} \\
 &= \frac{\sum_n \int tf(\mathbf{x} - \mathbf{x}_n, t - t_n) dt}{\sum_m \int f(\mathbf{x} - \mathbf{x}_m, t - t_m) dt}. \tag{6.43}
 \end{aligned}$$

We now assume for simplicity that the component density functions have zero mean so that

$$\int_{-\infty}^{\infty} f(\mathbf{x}, t)t dt = 0 \tag{6.44}$$

for all values of \mathbf{x} . Using a simple change of variable, we then obtain

$$\begin{aligned}
 y(\mathbf{x}) &= \frac{\sum_n g(\mathbf{x} - \mathbf{x}_n)t_n}{\sum_m g(\mathbf{x} - \mathbf{x}_m)} \\
 &= \sum_n k(\mathbf{x}, \mathbf{x}_n)t_n \tag{6.45}
 \end{aligned}$$

where $n, m = 1, \dots, N$ and the kernel function $k(\mathbf{x}, \mathbf{x}_n)$ is given by

$$k(\mathbf{x}, \mathbf{x}_n) = \frac{g(\mathbf{x} - \mathbf{x}_n)}{\sum_m g(\mathbf{x} - \mathbf{x}_m)} \tag{6.46}$$

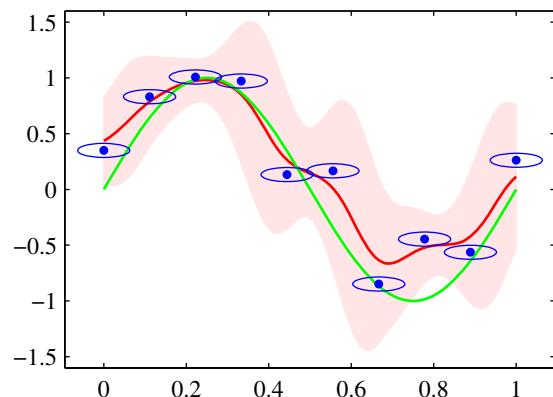
and we have defined

$$g(\mathbf{x}) = \int_{-\infty}^{\infty} f(\mathbf{x}, t) dt. \tag{6.47}$$

The result (6.45) is known as the *Nadaraya-Watson* model, or *kernel regression* (Nadaraya, 1964; Watson, 1964). For a localized kernel function, it has the property of giving more weight to the data points \mathbf{x}_n that are close to \mathbf{x} . Note that the kernel (6.46) satisfies the summation constraint

$$\sum_{n=1}^N k(\mathbf{x}, \mathbf{x}_n) = 1.$$

Figure 6.3 Illustration of the Nadaraya-Watson kernel regression model using isotropic Gaussian kernels, for the sinusoidal data set. The original sine function is shown by the green curve, the data points are shown in blue, and each is the centre of an isotropic Gaussian kernel. The resulting regression function, given by the conditional mean, is shown by the red line, along with the two-standard-deviation region for the conditional distribution $p(t|x)$ shown by the red shading. The blue ellipses around each data point show one standard deviation contour for the corresponding kernel. These appear noncircular due to the different scales on the horizontal and vertical axes.



In fact, this model defines not only a conditional expectation but also a full conditional distribution given by

$$p(t|\mathbf{x}) = \frac{p(t, \mathbf{x})}{\int p(t, \mathbf{x}) dt} = \frac{\sum_n f(\mathbf{x} - \mathbf{x}_n, t - t_n)}{\sum_m \int f(\mathbf{x} - \mathbf{x}_m, t - t_m) dt} \quad (6.48)$$

from which other expectations can be evaluated.

As an illustration we consider the case of a single input variable x in which $f(x, t)$ is given by a zero-mean isotropic Gaussian over the variable $\mathbf{z} = (x, t)$ with variance σ^2 . The corresponding conditional distribution (6.48) is given by a Gaussian mixture, and is shown, together with the conditional mean, for the sinusoidal synthetic data set in Figure 6.3.

An obvious extension of this model is to allow for more flexible forms of Gaussian components, for instance having different variance parameters for the input and target variables. More generally, we could model the joint distribution $p(t, \mathbf{x})$ using a Gaussian mixture model, trained using techniques discussed in Chapter 9 (Ghahramani and Jordan, 1994), and then find the corresponding conditional distribution $p(t|\mathbf{x})$. In this latter case we no longer have a representation in terms of kernel functions evaluated at the training set data points. However, the number of components in the mixture model can be smaller than the number of training set points, resulting in a model that is faster to evaluate for test data points. We have thereby accepted an increased computational cost during the training phase in order to have a model that is faster at making predictions.

6.4. Gaussian Processes

In Section 6.1, we introduced kernels by applying the concept of duality to a non-probabilistic model for regression. Here we extend the role of kernels to probabilis-

tic discriminative models, leading to the framework of Gaussian processes. We shall thereby see how kernels arise naturally in a Bayesian setting.

In Chapter 3, we considered linear regression models of the form $y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x})$ in which \mathbf{w} is a vector of parameters and $\phi(\mathbf{x})$ is a vector of fixed nonlinear basis functions that depend on the input vector \mathbf{x} . We showed that a prior distribution over \mathbf{w} induced a corresponding prior distribution over functions $y(\mathbf{x}, \mathbf{w})$. Given a training data set, we then evaluated the posterior distribution over \mathbf{w} and thereby obtained the corresponding posterior distribution over regression functions, which in turn (with the addition of noise) implies a predictive distribution $p(t|\mathbf{x})$ for new input vectors \mathbf{x} .

In the Gaussian process viewpoint, we dispense with the parametric model and instead define a prior probability distribution over functions directly. At first sight, it might seem difficult to work with a distribution over the uncountably infinite space of functions. However, as we shall see, for a finite training set we only need to consider the values of the function at the discrete set of input values \mathbf{x}_n corresponding to the training set and test set data points, and so in practice we can work in a finite space.

Models equivalent to Gaussian processes have been widely studied in many different fields. For instance, in the geostatistics literature Gaussian process regression is known as *kriging* (Cressie, 1993). Similarly, ARMA (autoregressive moving average) models, Kalman filters, and radial basis function networks can all be viewed as forms of Gaussian process models. Reviews of Gaussian processes from a machine learning perspective can be found in MacKay (1998), Williams (1999), and MacKay (2003), and a comparison of Gaussian process models with alternative approaches is given in Rasmussen (1996). See also Rasmussen and Williams (2006) for a recent textbook on Gaussian processes.

6.4.1 Linear regression revisited

In order to motivate the Gaussian process viewpoint, let us return to the linear regression example and re-derive the predictive distribution by working in terms of distributions over functions $y(\mathbf{x}, \mathbf{w})$. This will provide a specific example of a Gaussian process.

Consider a model defined in terms of a linear combination of M fixed basis functions given by the elements of the vector $\phi(\mathbf{x})$ so that

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) \quad (6.49)$$

where \mathbf{x} is the input vector and \mathbf{w} is the M -dimensional weight vector. Now consider a prior distribution over \mathbf{w} given by an isotropic Gaussian of the form

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \alpha^{-1} \mathbf{I}) \quad (6.50)$$

governed by the hyperparameter α , which represents the precision (inverse variance) of the distribution. For any given value of \mathbf{w} , the definition (6.49) defines a particular function of \mathbf{x} . The probability distribution over \mathbf{w} defined by (6.50) therefore induces a probability distribution over functions $y(\mathbf{x})$. In practice, we wish to evaluate this function at specific values of \mathbf{x} , for example at the training data points

$\mathbf{x}_1, \dots, \mathbf{x}_N$. We are therefore interested in the joint distribution of the function values $y(\mathbf{x}_1), \dots, y(\mathbf{x}_N)$, which we denote by the vector \mathbf{y} with elements $y_n = y(\mathbf{x}_n)$ for $n = 1, \dots, N$. From (6.49), this vector is given by

$$\mathbf{y} = \Phi \mathbf{w} \quad (6.51)$$

where Φ is the design matrix with elements $\Phi_{nk} = \phi_k(\mathbf{x}_n)$. We can find the probability distribution of \mathbf{y} as follows. First of all we note that \mathbf{y} is a linear combination of Gaussian distributed variables given by the elements of \mathbf{w} and hence is itself Gaussian. We therefore need only to find its mean and covariance, which are given from (6.50) by

$$\mathbb{E}[\mathbf{y}] = \Phi \mathbb{E}[\mathbf{w}] = \mathbf{0} \quad (6.52)$$

$$\text{cov}[\mathbf{y}] = \mathbb{E}[\mathbf{y}\mathbf{y}^T] = \Phi \mathbb{E}[\mathbf{w}\mathbf{w}^T] \Phi^T = \frac{1}{\alpha} \Phi \Phi^T = \mathbf{K} \quad (6.53)$$

where \mathbf{K} is the Gram matrix with elements

$$K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \frac{1}{\alpha} \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) \quad (6.54)$$

and $k(\mathbf{x}, \mathbf{x}')$ is the kernel function.

This model provides us with a particular example of a Gaussian process. In general, a Gaussian process is defined as a probability distribution over functions $y(\mathbf{x})$ such that the set of values of $y(\mathbf{x})$ evaluated at an arbitrary set of points $\mathbf{x}_1, \dots, \mathbf{x}_N$ jointly have a Gaussian distribution. In cases where the input vector \mathbf{x} is two dimensional, this may also be known as a *Gaussian random field*. More generally, a *stochastic process* $y(\mathbf{x})$ is specified by giving the joint probability distribution for any finite set of values $y(\mathbf{x}_1), \dots, y(\mathbf{x}_N)$ in a consistent manner.

A key point about Gaussian stochastic processes is that the joint distribution over N variables y_1, \dots, y_N is specified completely by the second-order statistics, namely the mean and the covariance. In most applications, we will not have any prior knowledge about the mean of $y(\mathbf{x})$ and so by symmetry we take it to be zero. This is equivalent to choosing the mean of the prior over weight values $p(\mathbf{w}|\alpha)$ to be zero in the basis function viewpoint. The specification of the Gaussian process is then completed by giving the covariance of $y(\mathbf{x})$ evaluated at any two values of \mathbf{x} , which is given by the kernel function

$$\mathbb{E}[y(\mathbf{x}_n)y(\mathbf{x}_m)] = k(\mathbf{x}_n, \mathbf{x}_m). \quad (6.55)$$

For the specific case of a Gaussian process defined by the linear regression model (6.49) with a weight prior (6.50), the kernel function is given by (6.54).

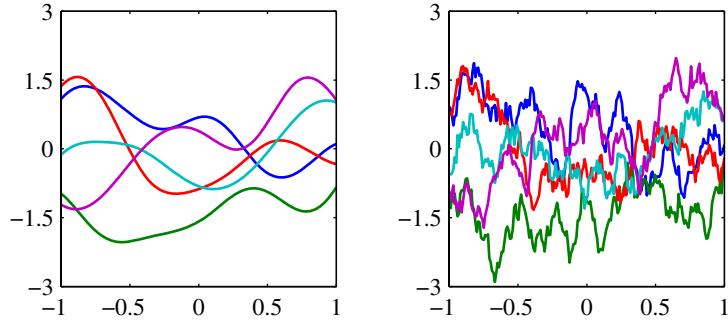
We can also define the kernel function directly, rather than indirectly through a choice of basis function. Figure 6.4 shows samples of functions drawn from Gaussian processes for two different choices of kernel function. The first of these is a ‘Gaussian’ kernel of the form (6.23), and the second is the exponential kernel given by

$$k(x, x') = \exp(-\theta |x - x'|) \quad (6.56)$$

which corresponds to the *Ornstein-Uhlenbeck process* originally introduced by Uhlenbeck and Ornstein (1930) to describe Brownian motion.

Exercise 2.31

Figure 6.4 Samples from Gaussian processes for a ‘Gaussian’ kernel (left) and an exponential kernel (right).



6.4.2 Gaussian processes for regression

In order to apply Gaussian process models to the problem of regression, we need to take account of the noise on the observed target values, which are given by

$$t_n = y_n + \epsilon_n \quad (6.57)$$

where $y_n = y(\mathbf{x}_n)$, and ϵ_n is a random noise variable whose value is chosen independently for each observation n . Here we shall consider noise processes that have a Gaussian distribution, so that

$$p(t_n|y_n) = \mathcal{N}(t_n|y_n, \beta^{-1}) \quad (6.58)$$

where β is a hyperparameter representing the precision of the noise. Because the noise is independent for each data point, the joint distribution of the target values $\mathbf{t} = (t_1, \dots, t_N)^T$ conditioned on the values of $\mathbf{y} = (y_1, \dots, y_N)^T$ is given by an isotropic Gaussian of the form

$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}(\mathbf{t}|\mathbf{y}, \beta^{-1} \mathbf{I}_N) \quad (6.59)$$

where \mathbf{I}_N denotes the $N \times N$ unit matrix. From the definition of a Gaussian process, the marginal distribution $p(\mathbf{y})$ is given by a Gaussian whose mean is zero and whose covariance is defined by a Gram matrix \mathbf{K} so that

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}). \quad (6.60)$$

The kernel function that determines \mathbf{K} is typically chosen to express the property that, for points \mathbf{x}_n and \mathbf{x}_m that are similar, the corresponding values $y(\mathbf{x}_n)$ and $y(\mathbf{x}_m)$ will be more strongly correlated than for dissimilar points. Here the notion of similarity will depend on the application.

In order to find the marginal distribution $p(\mathbf{t})$, conditioned on the input values $\mathbf{x}_1, \dots, \mathbf{x}_N$, we need to integrate over \mathbf{y} . This can be done by making use of the results from Section 2.3.3 for the linear-Gaussian model. Using (2.115), we see that the marginal distribution of \mathbf{t} is given by

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y}) d\mathbf{y} = \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C}) \quad (6.61)$$

where the covariance matrix \mathbf{C} has elements

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1} \delta_{nm}. \quad (6.62)$$

This result reflects the fact that the two Gaussian sources of randomness, namely that associated with $y(\mathbf{x})$ and that associated with ϵ , are independent and so their covariances simply add.

One widely used kernel function for Gaussian process regression is given by the exponential of a quadratic form, with the addition of constant and linear terms to give

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp \left\{ -\frac{\theta_1}{2} \|\mathbf{x}_n - \mathbf{x}_m\|^2 \right\} + \theta_2 + \theta_3 \mathbf{x}_n^T \mathbf{x}_m. \quad (6.63)$$

Note that the term involving θ_3 corresponds to a parametric model that is a linear function of the input variables. Samples from this prior are plotted for various values of the parameters $\theta_0, \dots, \theta_3$ in Figure 6.5, and Figure 6.6 shows a set of points sampled from the joint distribution (6.60) along with the corresponding values defined by (6.61).

So far, we have used the Gaussian process viewpoint to build a model of the joint distribution over sets of data points. Our goal in regression, however, is to make predictions of the target variables for new inputs, given a set of training data. Let us suppose that $\mathbf{t}_N = (t_1, \dots, t_N)^T$, corresponding to input values $\mathbf{x}_1, \dots, \mathbf{x}_N$, comprise the observed training set, and our goal is to predict the target variable t_{N+1} for a new input vector \mathbf{x}_{N+1} . This requires that we evaluate the predictive distribution $p(t_{N+1} | \mathbf{t}_N)$. Note that this distribution is conditioned also on the variables $\mathbf{x}_1, \dots, \mathbf{x}_N$ and \mathbf{x}_{N+1} . However, to keep the notation simple we will not show these conditioning variables explicitly.

To find the conditional distribution $p(t_{N+1} | \mathbf{t})$, we begin by writing down the joint distribution $p(\mathbf{t}_{N+1})$, where \mathbf{t}_{N+1} denotes the vector $(t_1, \dots, t_N, t_{N+1})^T$. We then apply the results from Section 2.3.1 to obtain the required conditional distribution, as illustrated in Figure 6.7.

From (6.61), the joint distribution over t_1, \dots, t_{N+1} will be given by

$$p(\mathbf{t}_{N+1}) = \mathcal{N}(\mathbf{t}_{N+1} | \mathbf{0}, \mathbf{C}_{N+1}) \quad (6.64)$$

where \mathbf{C}_{N+1} is an $(N + 1) \times (N + 1)$ covariance matrix with elements given by (6.62). Because this joint distribution is Gaussian, we can apply the results from Section 2.3.1 to find the conditional Gaussian distribution. To do this, we partition the covariance matrix as follows

$$\mathbf{C}_{N+1} = \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{pmatrix} \quad (6.65)$$

where \mathbf{C}_N is the $N \times N$ covariance matrix with elements given by (6.62) for $n, m = 1, \dots, N$, the vector \mathbf{k} has elements $k(\mathbf{x}_n, \mathbf{x}_{N+1})$ for $n = 1, \dots, N$, and the scalar

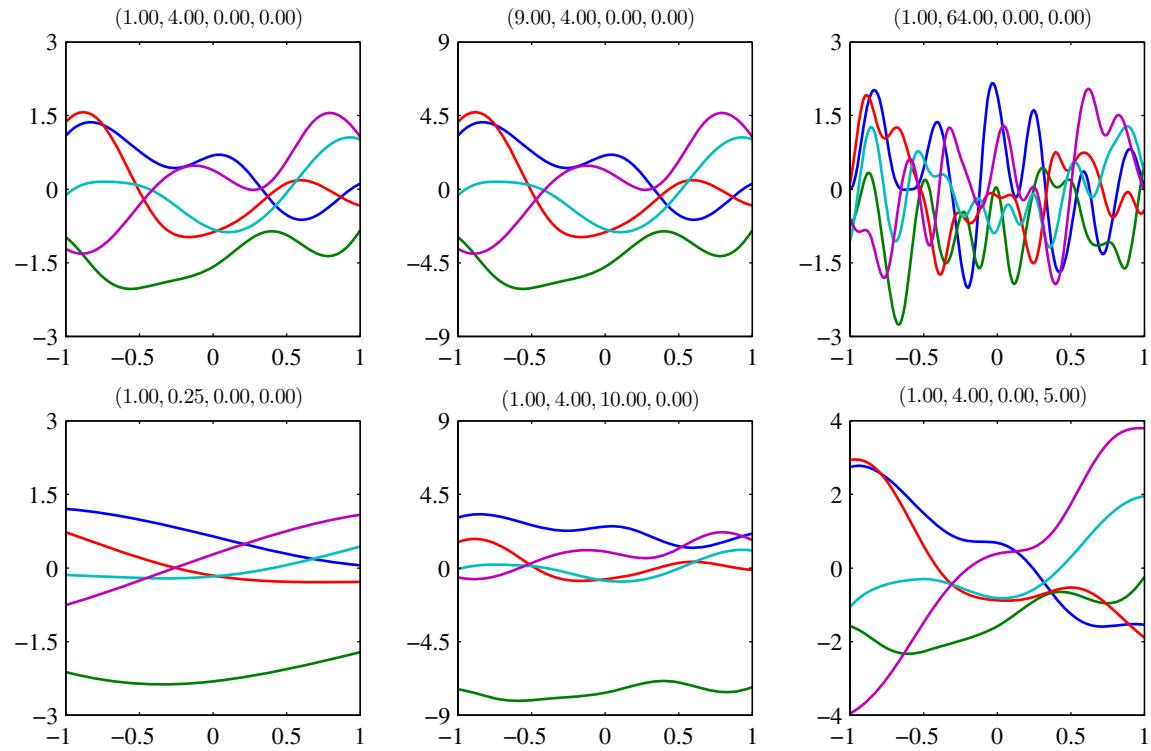


Figure 6.5 Samples from a Gaussian process prior defined by the covariance function (6.63). The title above each plot denotes $(\theta_0, \theta_1, \theta_2, \theta_3)$.

$c = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \beta^{-1}$. Using the results (2.81) and (2.82), we see that the conditional distribution $p(t_{N+1}|\mathbf{t})$ is a Gaussian distribution with mean and covariance given by

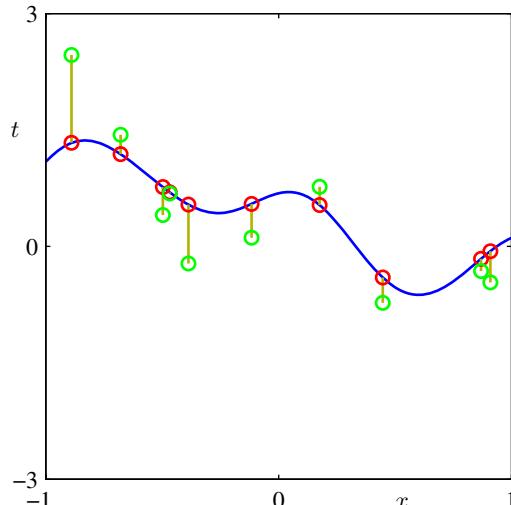
$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t} \quad (6.66)$$

$$\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k}. \quad (6.67)$$

These are the key results that define Gaussian process regression. Because the vector \mathbf{k} is a function of the test point input value \mathbf{x}_{N+1} , we see that the predictive distribution is a Gaussian whose mean and variance both depend on \mathbf{x}_{N+1} . An example of Gaussian process regression is shown in Figure 6.8.

The only restriction on the kernel function is that the covariance matrix given by (6.62) must be positive definite. If λ_i is an eigenvalue of \mathbf{K} , then the corresponding eigenvalue of \mathbf{C} will be $\lambda_i + \beta^{-1}$. It is therefore sufficient that the kernel matrix $k(\mathbf{x}_n, \mathbf{x}_m)$ be positive semidefinite for any pair of points \mathbf{x}_n and \mathbf{x}_m , so that $\lambda_i \geq 0$, because any eigenvalue λ_i that is zero will still give rise to a positive eigenvalue for \mathbf{C} because $\beta > 0$. This is the same restriction on the kernel function discussed earlier, and so we can again exploit all of the techniques in Section 6.2 to construct

Figure 6.6 Illustration of the sampling of data points $\{t_n\}$ from a Gaussian process. The blue curve shows a sample function from the Gaussian process prior over functions, and the red points show the values of y_n obtained by evaluating the function at a set of input values $\{x_n\}$. The corresponding values of $\{t_n\}$, shown in green, are obtained by adding independent Gaussian noise to each of the $\{y_n\}$.



suitable kernels.

Note that the mean (6.66) of the predictive distribution can be written, as a function of \mathbf{x}_{N+1} , in the form

$$m(\mathbf{x}_{N+1}) = \sum_{n=1}^N a_n k(\mathbf{x}_n, \mathbf{x}_{N+1}) \quad (6.68)$$

where a_n is the n^{th} component of $\mathbf{C}_N^{-1}\mathbf{t}$. Thus, if the kernel function $k(\mathbf{x}_n, \mathbf{x}_m)$ depends only on the distance $\|\mathbf{x}_n - \mathbf{x}_m\|$, then we obtain an expansion in radial basis functions.

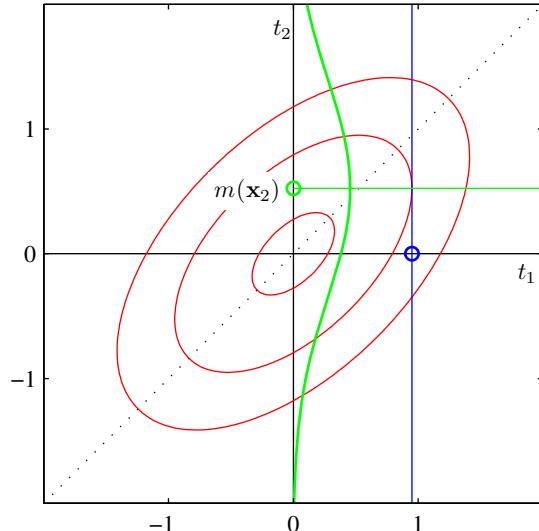
The results (6.66) and (6.67) define the predictive distribution for Gaussian process regression with an arbitrary kernel function $k(\mathbf{x}_n, \mathbf{x}_m)$. In the particular case in which the kernel function $k(\mathbf{x}, \mathbf{x}')$ is defined in terms of a finite set of basis functions, we can derive the results obtained previously in Section 3.3.2 for linear regression starting from the Gaussian process viewpoint.

For such models, we can therefore obtain the predictive distribution either by taking a parameter space viewpoint and using the linear regression result or by taking a function space viewpoint and using the Gaussian process result.

The central computational operation in using Gaussian processes will involve the inversion of a matrix of size $N \times N$, for which standard methods require $O(N^3)$ computations. By contrast, in the basis function model we have to invert a matrix \mathbf{S}_N of size $M \times M$, which has $O(M^3)$ computational complexity. Note that for both viewpoints, the matrix inversion must be performed once for the given training set. For each new test point, both methods require a vector-matrix multiply, which has cost $O(N^2)$ in the Gaussian process case and $O(M^2)$ for the linear basis function model. If the number M of basis functions is smaller than the number N of data points, it will be computationally more efficient to work in the basis function

Exercise 6.21

Figure 6.7 Illustration of the mechanism of Gaussian process regression for the case of one training point and one test point, in which the red ellipses show contours of the joint distribution $p(t_1, t_2)$. Here t_1 is the training data point, and conditioning on the value of t_1 , corresponding to the vertical blue line, we obtain $p(t_2|t_1)$ shown as a function of t_2 by the green curve.



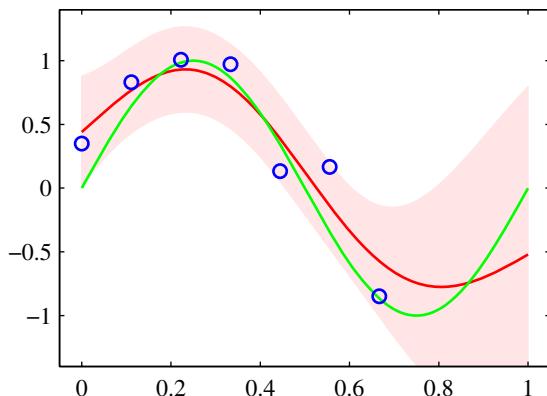
framework. However, an advantage of a Gaussian processes viewpoint is that we can consider covariance functions that can only be expressed in terms of an infinite number of basis functions.

For large training data sets, however, the direct application of Gaussian process methods can become infeasible, and so a range of approximation schemes have been developed that have better scaling with training set size than the exact approach (Gibbs, 1997; Tresp, 2001; Smola and Bartlett, 2001; Williams and Seeger, 2001; Csató and Opper, 2002; Seeger *et al.*, 2003). Practical issues in the application of Gaussian processes are discussed in Bishop and Nabney (2008).

We have introduced Gaussian process regression for the case of a single target variable. The extension of this formalism to multiple target variables, known as co-kriging (Cressie, 1993), is straightforward. Various other extensions of Gaus-

Exercise 6.23

Figure 6.8 Illustration of Gaussian process regression applied to the sinusoidal data set in Figure A.6 in which the three right-most data points have been omitted. The green curve shows the sinusoidal function from which the data points, shown in blue, are obtained by sampling and addition of Gaussian noise. The red line shows the mean of the Gaussian process predictive distribution, and the shaded region corresponds to plus and minus two standard deviations. Notice how the uncertainty increases in the region to the right of the data points.



sian process regression have also been considered, for purposes such as modelling the distribution over low-dimensional manifolds for unsupervised learning (Bishop *et al.*, 1998a) and the solution of stochastic differential equations (Graepel, 2003).

6.4.3 Learning the hyperparameters

The predictions of a Gaussian process model will depend, in part, on the choice of covariance function. In practice, rather than fixing the covariance function, we may prefer to use a parametric family of functions and then infer the parameter values from the data. These parameters govern such things as the length scale of the correlations and the precision of the noise and correspond to the hyperparameters in a standard parametric model.

Techniques for learning the hyperparameters are based on the evaluation of the likelihood function $p(\mathbf{t}|\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ denotes the hyperparameters of the Gaussian process model. The simplest approach is to make a point estimate of $\boldsymbol{\theta}$ by maximizing the log likelihood function. Because $\boldsymbol{\theta}$ represents a set of hyperparameters for the regression problem, this can be viewed as analogous to the type 2 maximum likelihood procedure for linear regression models. Maximization of the log likelihood can be done using efficient gradient-based optimization algorithms such as conjugate gradients (Fletcher, 1987; Nocedal and Wright, 1999; Bishop and Nabney, 2008).

The log likelihood function for a Gaussian process regression model is easily evaluated using the standard form for a multivariate Gaussian distribution, giving

$$\ln p(\mathbf{t}|\boldsymbol{\theta}) = -\frac{1}{2} \ln |\mathbf{C}_N| - \frac{1}{2} \mathbf{t}^T \mathbf{C}_N^{-1} \mathbf{t} - \frac{N}{2} \ln(2\pi). \quad (6.69)$$

For nonlinear optimization, we also need the gradient of the log likelihood function with respect to the parameter vector $\boldsymbol{\theta}$. We shall assume that evaluation of the derivatives of \mathbf{C}_N is straightforward, as would be the case for the covariance functions considered in this chapter. Making use of the result (C.21) for the derivative of \mathbf{C}_N^{-1} , together with the result (C.22) for the derivative of $\ln |\mathbf{C}_N|$, we obtain

$$\frac{\partial}{\partial \theta_i} \ln p(\mathbf{t}|\boldsymbol{\theta}) = -\frac{1}{2} \text{Tr} \left(\mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta_i} \right) + \frac{1}{2} \mathbf{t}^T \mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta_i} \mathbf{C}_N^{-1} \mathbf{t}. \quad (6.70)$$

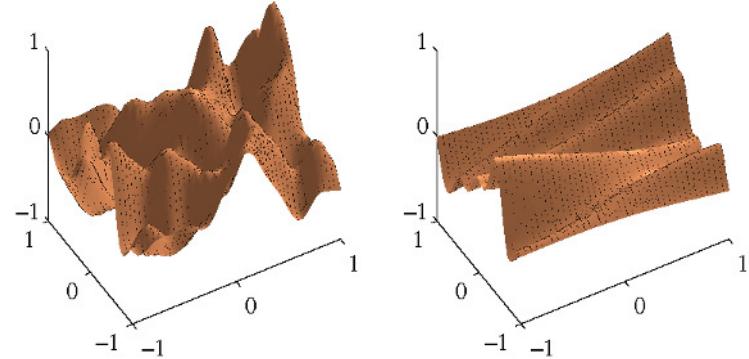
Because $\ln p(\mathbf{t}|\boldsymbol{\theta})$ will in general be a nonconvex function, it can have multiple maxima.

It is straightforward to introduce a prior over $\boldsymbol{\theta}$ and to maximize the log posterior using gradient-based methods. In a fully Bayesian treatment, we need to evaluate marginals over $\boldsymbol{\theta}$ weighted by the product of the prior $p(\boldsymbol{\theta})$ and the likelihood function $p(\mathbf{t}|\boldsymbol{\theta})$. In general, however, exact marginalization will be intractable, and we must resort to approximations.

The Gaussian process regression model gives a predictive distribution whose mean and variance are functions of the input vector \mathbf{x} . However, we have assumed that the contribution to the predictive variance arising from the additive noise, governed by the parameter β , is a constant. For some problems, known as *heteroscedastic*, the noise variance itself will also depend on \mathbf{x} . To model this, we can extend the

Section 3.5

Figure 6.9 Samples from the ARD prior for Gaussian processes, in which the kernel function is given by (6.71). The left plot corresponds to $\eta_1 = \eta_2 = 1$, and the right plot corresponds to $\eta_1 = 1, \eta_2 = 0.01$.



Gaussian process framework by introducing a second Gaussian process to represent the dependence of β on the input \mathbf{x} (Goldberg *et al.*, 1998). Because β is a variance, and hence nonnegative, we use the Gaussian process to model $\ln \beta(\mathbf{x})$.

6.4.4 Automatic relevance determination

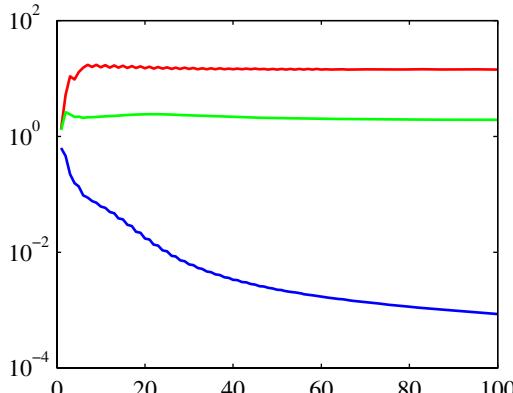
In the previous section, we saw how maximum likelihood could be used to determine a value for the correlation length-scale parameter in a Gaussian process. This technique can usefully be extended by incorporating a separate parameter for each input variable (Rasmussen and Williams, 2006). The result, as we shall see, is that the optimization of these parameters by maximum likelihood allows the relative importance of different inputs to be inferred from the data. This represents an example in the Gaussian process context of *automatic relevance determination*, or *ARD*, which was originally formulated in the framework of neural networks (MacKay, 1994; Neal, 1996). The mechanism by which appropriate inputs are preferred is discussed in Section 7.2.2.

Consider a Gaussian process with a two-dimensional input space $\mathbf{x} = (x_1, x_2)$, having a kernel function of the form

$$k(\mathbf{x}, \mathbf{x}') = \theta_0 \exp \left\{ -\frac{1}{2} \sum_{i=1}^2 \eta_i (x_i - x'_i)^2 \right\}. \quad (6.71)$$

Samples from the resulting prior over functions $y(\mathbf{x})$ are shown for two different settings of the precision parameters η_i in Figure 6.9. We see that, as a particular parameter η_i becomes small, the function becomes relatively insensitive to the corresponding input variable x_i . By adapting these parameters to a data set using maximum likelihood, it becomes possible to detect input variables that have little effect on the predictive distribution, because the corresponding values of η_i will be small. This can be useful in practice because it allows such inputs to be discarded. ARD is illustrated using a simple synthetic data set having three inputs x_1, x_2 and x_3 (Nabney, 2002) in Figure 6.10. The target variable t , is generated by sampling 100 values of x_1 from a Gaussian, evaluating the function $\sin(2\pi x_1)$, and then adding

Figure 6.10 Illustration of automatic relevance determination in a Gaussian process for a synthetic problem having three inputs x_1 , x_2 , and x_3 , for which the curves show the corresponding values of the hyperparameters η_1 (red), η_2 (green), and η_3 (blue) as a function of the number of iterations when optimizing the marginal likelihood. Details are given in the text. Note the logarithmic scale on the vertical axis.



Gaussian noise. Values of x_2 are given by copying the corresponding values of x_1 and adding noise, and values of x_3 are sampled from an independent Gaussian distribution. Thus x_1 is a good predictor of t , x_2 is a more noisy predictor of t , and x_3 has only chance correlations with t . The marginal likelihood for a Gaussian process with ARD parameters η_1, η_2, η_3 is optimized using the scaled conjugate gradients algorithm. We see from Figure 6.10 that η_1 converges to a relatively large value, η_2 converges to a much smaller value, and η_3 becomes very small indicating that x_3 is irrelevant for predicting t .

The ARD framework is easily incorporated into the exponential-quadratic kernel (6.63) to give the following form of kernel function, which has been found useful for applications of Gaussian processes to a range of regression problems

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp \left\{ -\frac{1}{2} \sum_{i=1}^D \eta_i (x_{ni} - x_{mi})^2 \right\} + \theta_2 + \theta_3 \sum_{i=1}^D x_{ni} x_{mi} \quad (6.72)$$

where D is the dimensionality of the input space.

6.4.5 Gaussian processes for classification

In a probabilistic approach to classification, our goal is to model the posterior probabilities of the target variable for a new input vector, given a set of training data. These probabilities must lie in the interval $(0, 1)$, whereas a Gaussian process model makes predictions that lie on the entire real axis. However, we can easily adapt Gaussian processes to classification problems by transforming the output of the Gaussian process using an appropriate nonlinear activation function.

Consider first the two-class problem with a target variable $t \in \{0, 1\}$. If we define a Gaussian process over a function $a(\mathbf{x})$ and then transform the function using a logistic sigmoid $y = \sigma(a)$, given by (4.59), then we will obtain a non-Gaussian stochastic process over functions $y(\mathbf{x})$ where $y \in (0, 1)$. This is illustrated for the case of a one-dimensional input space in Figure 6.11 in which the probability distri-

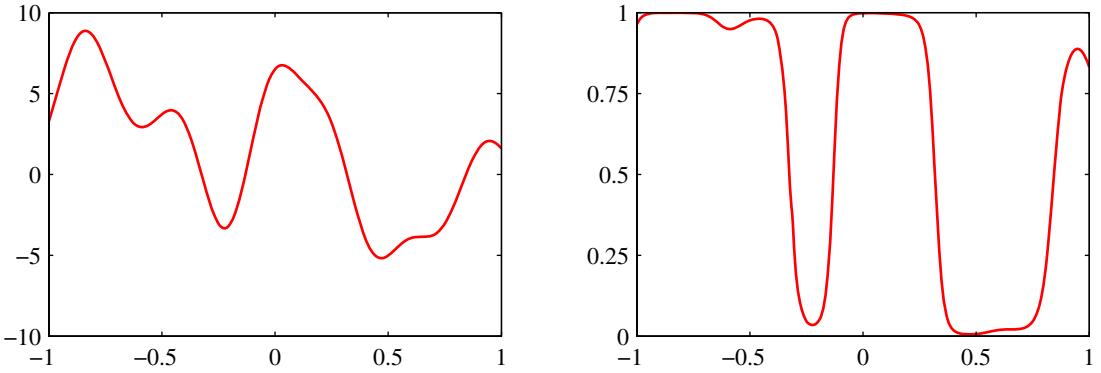


Figure 6.11 The left plot shows a sample from a Gaussian process prior over functions $a(\mathbf{x})$, and the right plot shows the result of transforming this sample using a logistic sigmoid function.

bution over the target variable t is then given by the Bernoulli distribution

$$p(t|a) = \sigma(a)^t(1 - \sigma(a))^{1-t}. \quad (6.73)$$

As usual, we denote the training set inputs by $\mathbf{x}_1, \dots, \mathbf{x}_N$ with corresponding observed target variables $\mathbf{t} = (t_1, \dots, t_N)^T$. We also consider a single test point \mathbf{x}_{N+1} with target value t_{N+1} . Our goal is to determine the predictive distribution $p(t_{N+1}|\mathbf{t})$, where we have left the conditioning on the input variables implicit. To do this we introduce a Gaussian process prior over the vector \mathbf{a}_{N+1} , which has components $a(\mathbf{x}_1), \dots, a(\mathbf{x}_{N+1})$. This in turn defines a non-Gaussian process over \mathbf{t}_{N+1} , and by conditioning on the training data \mathbf{t}_N we obtain the required predictive distribution. The Gaussian process prior for \mathbf{a}_{N+1} takes the form

$$p(\mathbf{a}_{N+1}) = \mathcal{N}(\mathbf{a}_{N+1} | \mathbf{0}, \mathbf{C}_{N+1}). \quad (6.74)$$

Unlike the regression case, the covariance matrix no longer includes a noise term because we assume that all of the training data points are correctly labelled. However, for numerical reasons it is convenient to introduce a noise-like term governed by a parameter ν that ensures that the covariance matrix is positive definite. Thus the covariance matrix \mathbf{C}_{N+1} has elements given by

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \nu \delta_{nm} \quad (6.75)$$

where $k(\mathbf{x}_n, \mathbf{x}_m)$ is any positive semidefinite kernel function of the kind considered in Section 6.2, and the value of ν is typically fixed in advance. We shall assume that the kernel function $k(\mathbf{x}, \mathbf{x}')$ is governed by a vector θ of parameters, and we shall later discuss how θ may be learned from the training data.

For two-class problems, it is sufficient to predict $p(t_{N+1} = 1|\mathbf{t}_N)$ because the value of $p(t_{N+1} = 0|\mathbf{t}_N)$ is then given by $1 - p(t_{N+1} = 1|\mathbf{t}_N)$. The required

predictive distribution is given by

$$p(t_{N+1} = 1 | \mathbf{t}_N) = \int p(t_{N+1} = 1 | a_{N+1}) p(a_{N+1} | \mathbf{t}_N) da_{N+1} \quad (6.76)$$

where $p(t_{N+1} = 1 | a_{N+1}) = \sigma(a_{N+1})$.

This integral is analytically intractable, and so may be approximated using sampling methods (Neal, 1997). Alternatively, we can consider techniques based on an analytical approximation. In Section 4.5.2, we derived the approximate formula (4.153) for the convolution of a logistic sigmoid with a Gaussian distribution. We can use this result to evaluate the integral in (6.76) provided we have a Gaussian approximation to the posterior distribution $p(a_{N+1} | \mathbf{t}_N)$. The usual justification for a Gaussian approximation to a posterior distribution is that the true posterior will tend to a Gaussian as the number of data points increases as a consequence of the central limit theorem. In the case of Gaussian processes, the number of variables grows with the number of data points, and so this argument does not apply directly. However, if we consider increasing the number of data points falling in a fixed region of \mathbf{x} space, then the corresponding uncertainty in the function $a(\mathbf{x})$ will decrease, again leading asymptotically to a Gaussian (Williams and Barber, 1998).

Section 2.3

Section 10.1

Section 10.7

Section 4.4

Three different approaches to obtaining a Gaussian approximation have been considered. One technique is based on *variational inference* (Gibbs and MacKay, 2000) and makes use of the local variational bound (10.144) on the logistic sigmoid. This allows the product of sigmoid functions to be approximated by a product of Gaussians thereby allowing the marginalization over \mathbf{a}_N to be performed analytically. The approach also yields a lower bound on the likelihood function $p(\mathbf{t}_N | \theta)$. The variational framework for Gaussian process classification can also be extended to multiclass ($K > 2$) problems by using a Gaussian approximation to the softmax function (Gibbs, 1997).

A second approach uses *expectation propagation* (Opper and Winther, 2000b; Minka, 2001b; Seeger, 2003). Because the true posterior distribution is unimodal, as we shall see shortly, the expectation propagation approach can give good results.

6.4.6 Laplace approximation

The third approach to Gaussian process classification is based on the Laplace approximation, which we now consider in detail. In order to evaluate the predictive distribution (6.76), we seek a Gaussian approximation to the posterior distribution over a_{N+1} , which, using Bayes' theorem, is given by

$$\begin{aligned} p(a_{N+1} | \mathbf{t}_N) &= \int p(a_{N+1}, \mathbf{a}_N | \mathbf{t}_N) d\mathbf{a}_N \\ &= \frac{1}{p(\mathbf{t}_N)} \int p(a_{N+1}, \mathbf{a}_N) p(\mathbf{t}_N | a_{N+1}, \mathbf{a}_N) d\mathbf{a}_N \\ &= \frac{1}{p(\mathbf{t}_N)} \int p(a_{N+1} | \mathbf{a}_N) p(\mathbf{a}_N) p(\mathbf{t}_N | \mathbf{a}_N) d\mathbf{a}_N \\ &= \int p(a_{N+1} | \mathbf{a}_N) p(\mathbf{a}_N | \mathbf{t}_N) d\mathbf{a}_N \end{aligned} \quad (6.77)$$

where we have used $p(\mathbf{t}_N | a_{N+1}, \mathbf{a}_N) = p(\mathbf{t}_N | \mathbf{a}_N)$. The conditional distribution $p(a_{N+1} | \mathbf{a}_N)$ is obtained by invoking the results (6.66) and (6.67) for Gaussian process regression, to give

$$p(a_{N+1} | \mathbf{a}_N) = \mathcal{N}(a_{N+1} | \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{a}_N, c - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k}). \quad (6.78)$$

We can therefore evaluate the integral in (6.77) by finding a Laplace approximation for the posterior distribution $p(\mathbf{a}_N | \mathbf{t}_N)$, and then using the standard result for the convolution of two Gaussian distributions.

The prior $p(\mathbf{a}_N)$ is given by a zero-mean Gaussian process with covariance matrix \mathbf{C}_N , and the data term (assuming independence of the data points) is given by

$$p(\mathbf{t}_N | \mathbf{a}_N) = \prod_{n=1}^N \sigma(a_n)^{t_n} (1 - \sigma(a_n))^{1-t_n} = \prod_{n=1}^N e^{a_n t_n} \sigma(-a_n). \quad (6.79)$$

We then obtain the Laplace approximation by Taylor expanding the logarithm of $p(\mathbf{a}_N | \mathbf{t}_N)$, which up to an additive normalization constant is given by the quantity

$$\begin{aligned} \Psi(\mathbf{a}_N) &= \ln p(\mathbf{a}_N) + \ln p(\mathbf{t}_N | \mathbf{a}_N) \\ &= -\frac{1}{2} \mathbf{a}_N^T \mathbf{C}_N^{-1} \mathbf{a}_N - \frac{N}{2} \ln(2\pi) - \frac{1}{2} \ln |\mathbf{C}_N| + \mathbf{t}_N^T \mathbf{a}_N \\ &\quad - \sum_{n=1}^N \ln(1 + e^{a_n}) + \text{const.} \end{aligned} \quad (6.80)$$

First we need to find the mode of the posterior distribution, and this requires that we evaluate the gradient of $\Psi(\mathbf{a}_N)$, which is given by

$$\nabla \Psi(\mathbf{a}_N) = \mathbf{t}_N - \boldsymbol{\sigma}_N - \mathbf{C}_N^{-1} \mathbf{a}_N \quad (6.81)$$

where $\boldsymbol{\sigma}_N$ is a vector with elements $\sigma(a_n)$. We cannot simply find the mode by setting this gradient to zero, because $\boldsymbol{\sigma}_N$ depends nonlinearly on \mathbf{a}_N , and so we resort to an iterative scheme based on the Newton-Raphson method, which gives rise to an iterative reweighted least squares (IRLS) algorithm. This requires the second derivatives of $\Psi(\mathbf{a}_N)$, which we also require for the Laplace approximation anyway, and which are given by

$$\nabla \nabla \Psi(\mathbf{a}_N) = -\mathbf{W}_N - \mathbf{C}_N^{-1} \quad (6.82)$$

where \mathbf{W}_N is a diagonal matrix with elements $\sigma(a_n)(1 - \sigma(a_n))$, and we have used the result (4.88) for the derivative of the logistic sigmoid function. Note that these diagonal elements lie in the range $(0, 1/4)$, and hence \mathbf{W}_N is a positive definite matrix. Because \mathbf{C}_N (and hence its inverse) is positive definite by construction, and because the sum of two positive definite matrices is also positive definite, we see that the Hessian matrix $\mathbf{A} = -\nabla \nabla \Psi(\mathbf{a}_N)$ is positive definite and so the posterior distribution $p(\mathbf{a}_N | \mathbf{t}_N)$ is log convex and therefore has a single mode that is the global

Section 4.3.3

Exercise 6.24

maximum. The posterior distribution is not Gaussian, however, because the Hessian is a function of \mathbf{a}_N .

Using the Newton-Raphson formula (4.92), the iterative update equation for \mathbf{a}_N is given by

$$\mathbf{a}_N^{\text{new}} = \mathbf{C}_N(\mathbf{I} + \mathbf{W}_N \mathbf{C}_N)^{-1} \{ \mathbf{t}_N - \boldsymbol{\sigma}_N + \mathbf{W}_N \mathbf{a}_N \}. \quad (6.83)$$

These equations are iterated until they converge to the mode which we denote by \mathbf{a}_N^* . At the mode, the gradient $\nabla \Psi(\mathbf{a}_N)$ will vanish, and hence \mathbf{a}_N^* will satisfy

$$\mathbf{a}_N^* = \mathbf{C}_N(\mathbf{t}_N - \boldsymbol{\sigma}_N). \quad (6.84)$$

Once we have found the mode \mathbf{a}_N^* of the posterior, we can evaluate the Hessian matrix given by

$$\mathbf{H} = -\nabla \nabla \Psi(\mathbf{a}_N) = \mathbf{W}_N + \mathbf{C}_N^{-1} \quad (6.85)$$

where the elements of \mathbf{W}_N are evaluated using \mathbf{a}_N^* . This defines our Gaussian approximation to the posterior distribution $p(\mathbf{a}_N | \mathbf{t}_N)$ given by

$$q(\mathbf{a}_N) = \mathcal{N}(\mathbf{a}_N | \mathbf{a}_N^*, \mathbf{H}^{-1}). \quad (6.86)$$

We can now combine this with (6.78) and hence evaluate the integral (6.77). Because this corresponds to a linear-Gaussian model, we can use the general result (2.115) to give

$$\mathbb{E}[a_{N+1} | \mathbf{t}_N] = \mathbf{k}^T(\mathbf{t}_N - \boldsymbol{\sigma}_N) \quad (6.87)$$

$$\text{var}[a_{N+1} | \mathbf{t}_N] = c - \mathbf{k}^T(\mathbf{W}_N^{-1} + \mathbf{C}_N)^{-1}\mathbf{k}. \quad (6.88)$$

Now that we have a Gaussian distribution for $p(a_{N+1} | \mathbf{t}_N)$, we can approximate the integral (6.76) using the result (4.153). As with the Bayesian logistic regression model of Section 4.5, if we are only interested in the decision boundary corresponding to $p(t_{N+1} | \mathbf{t}_N) = 0.5$, then we need only consider the mean and we can ignore the effect of the variance.

We also need to determine the parameters $\boldsymbol{\theta}$ of the covariance function. One approach is to maximize the likelihood function given by $p(\mathbf{t}_N | \boldsymbol{\theta})$ for which we need expressions for the log likelihood and its gradient. If desired, suitable regularization terms can also be added, leading to a penalized maximum likelihood solution. The likelihood function is defined by

$$p(\mathbf{t}_N | \boldsymbol{\theta}) = \int p(\mathbf{t}_N | \mathbf{a}_N) p(\mathbf{a}_N | \boldsymbol{\theta}) d\mathbf{a}_N. \quad (6.89)$$

This integral is analytically intractable, so again we make use of the Laplace approximation. Using the result (4.135), we obtain the following approximation for the log of the likelihood function

$$\ln p(\mathbf{t}_N | \boldsymbol{\theta}) = \Psi(\mathbf{a}_N^*) - \frac{1}{2} \ln |\mathbf{W}_N + \mathbf{C}_N^{-1}| + \frac{N}{2} \ln(2\pi) \quad (6.90)$$

Exercise 6.25

Exercise 6.26

where $\Psi(\mathbf{a}_N^*) = \ln p(\mathbf{a}_N^* | \boldsymbol{\theta}) + \ln p(\mathbf{t}_N | \mathbf{a}_N^*)$. We also need to evaluate the gradient of $\ln p(\mathbf{t}_N | \boldsymbol{\theta})$ with respect to the parameter vector $\boldsymbol{\theta}$. Note that changes in $\boldsymbol{\theta}$ will cause changes in \mathbf{a}_N^* , leading to additional terms in the gradient. Thus, when we differentiate (6.90) with respect to $\boldsymbol{\theta}$, we obtain two sets of terms, the first arising from the dependence of the covariance matrix \mathbf{C}_N on $\boldsymbol{\theta}$, and the rest arising from dependence of \mathbf{a}_N^* on $\boldsymbol{\theta}$.

The terms arising from the explicit dependence on $\boldsymbol{\theta}$ can be found by using (6.80) together with the results (C.21) and (C.22), and are given by

$$\begin{aligned} \frac{\partial \ln p(\mathbf{t}_N | \boldsymbol{\theta})}{\partial \theta_j} &= \frac{1}{2} \mathbf{a}_N^{*\top} \mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta_j} \mathbf{C}_N^{-1} \mathbf{a}_N^* \\ &\quad - \frac{1}{2} \text{Tr} \left[(\mathbf{I} + \mathbf{C}_N \mathbf{W}_N)^{-1} \mathbf{W}_N \frac{\partial \mathbf{C}_N}{\partial \theta_j} \right]. \end{aligned} \quad (6.91)$$

To compute the terms arising from the dependence of \mathbf{a}_N^* on $\boldsymbol{\theta}$, we note that the Laplace approximation has been constructed such that $\Psi(\mathbf{a}_N)$ has zero gradient at $\mathbf{a}_N = \mathbf{a}_N^*$, and so $\Psi(\mathbf{a}_N^*)$ gives no contribution to the gradient as a result of its dependence on \mathbf{a}_N^* . This leaves the following contribution to the derivative with respect to a component θ_j of $\boldsymbol{\theta}$

$$\begin{aligned} &- \frac{1}{2} \sum_{n=1}^N \frac{\partial \ln |\mathbf{W}_N + \mathbf{C}_N^{-1}|}{\partial a_n^*} \frac{\partial a_n^*}{\partial \theta_j} \\ &= -\frac{1}{2} \sum_{n=1}^N [(\mathbf{I} + \mathbf{C}_N \mathbf{W}_N)^{-1} \mathbf{C}_N]_{nn} \sigma_n^*(1 - \sigma_n^*)(1 - 2\sigma_n^*) \frac{\partial a_n^*}{\partial \theta_j} \end{aligned} \quad (6.92)$$

where $\sigma_n^* = \sigma(a_n^*)$, and again we have used the result (C.22) together with the definition of \mathbf{W}_N . We can evaluate the derivative of a_N^* with respect to θ_j by differentiating the relation (6.84) with respect to θ_j to give

$$\frac{\partial a_n^*}{\partial \theta_j} = \frac{\partial \mathbf{C}_N}{\partial \theta_j} (\mathbf{t}_N - \boldsymbol{\sigma}_N) - \mathbf{C}_N \mathbf{W}_N \frac{\partial a_n^*}{\partial \theta_j}. \quad (6.93)$$

Rearranging then gives

$$\frac{\partial a_n^*}{\partial \theta_j} = (\mathbf{I} + \mathbf{W}_N \mathbf{C}_N)^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta_j} (\mathbf{t}_N - \boldsymbol{\sigma}_N). \quad (6.94)$$

Combining (6.91), (6.92), and (6.94), we can evaluate the gradient of the log-likelihood function, which can be used with standard nonlinear optimization algorithms in order to determine a value for $\boldsymbol{\theta}$.

We can illustrate the application of the Laplace approximation for Gaussian processes using the synthetic two-class data set shown in Figure 6.12. Extension of the Laplace approximation to Gaussian processes involving $K > 2$ classes, using the softmax activation function, is straightforward (Williams and Barber, 1998).

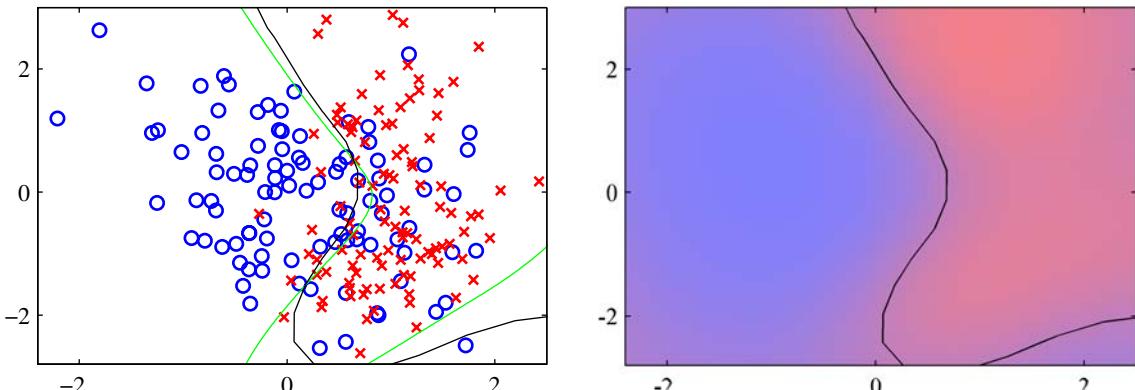


Figure 6.12 Illustration of the use of a Gaussian process for classification, showing the data on the left together with the optimal decision boundary from the true distribution in green, and the decision boundary from the Gaussian process classifier in black. On the right is the predicted posterior probability for the blue and red classes together with the Gaussian process decision boundary.

6.4.7 Connection to neural networks

We have seen that the range of functions which can be represented by a neural network is governed by the number M of hidden units, and that, for sufficiently large M , a two-layer network can approximate any given function with arbitrary accuracy. In the framework of maximum likelihood, the number of hidden units needs to be limited (to a level dependent on the size of the training set) in order to avoid over-fitting. However, from a Bayesian perspective it makes little sense to limit the number of parameters in the network according to the size of the training set.

In a Bayesian neural network, the prior distribution over the parameter vector w , in conjunction with the network function $f(x, w)$, produces a prior distribution over functions from $y(x)$ where y is the vector of network outputs. Neal (1996) has shown that, for a broad class of prior distributions over w , the distribution of functions generated by a neural network will tend to a Gaussian process in the limit $M \rightarrow \infty$. It should be noted, however, that in this limit the output variables of the neural network become independent. One of the great merits of neural networks is that the outputs share the hidden units and so they can ‘borrow statistical strength’ from each other, that is, the weights associated with each hidden unit are influenced by all of the output variables not just by one of them. This property is therefore lost in the Gaussian process limit.

We have seen that a Gaussian process is determined by its covariance (kernel) function. Williams (1998) has given explicit forms for the covariance in the case of two specific choices for the hidden unit activation function (probit and Gaussian). These kernel functions $k(x, x')$ are nonstationary, i.e. they cannot be expressed as a function of the difference $x - x'$, as a consequence of the Gaussian weight prior being centred on zero which breaks translation invariance in weight space.

By working directly with the covariance function we have implicitly marginalized over the distribution of weights. If the weight prior is governed by hyperparameters, then their values will determine the length scales of the distribution over functions, as can be understood by studying the examples in Figure 5.11 for the case of a finite number of hidden units. Note that we cannot marginalize out the hyperparameters analytically, and must instead resort to techniques of the kind discussed in Section 6.4.

Exercises

- 6.1** (**) **www** Consider the dual formulation of the least squares linear regression problem given in Section 6.1. Show that the solution for the components a_n of the vector \mathbf{a} can be expressed as a linear combination of the elements of the vector $\phi(\mathbf{x}_n)$. Denoting these coefficients by the vector \mathbf{w} , show that the dual of the dual formulation is given by the original representation in terms of the parameter vector \mathbf{w} .
- 6.2** (**) In this exercise, we develop a dual formulation of the perceptron learning algorithm. Using the perceptron learning rule (4.55), show that the learned weight vector \mathbf{w} can be written as a linear combination of the vectors $t_n \phi(\mathbf{x}_n)$ where $t_n \in \{-1, +1\}$. Denote the coefficients of this linear combination by α_n and derive a formulation of the perceptron learning algorithm, and the predictive function for the perceptron, in terms of the α_n . Show that the feature vector $\phi(\mathbf{x})$ enters only in the form of the kernel function $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$.
- 6.3** (*) The nearest-neighbour classifier (Section 2.5.2) assigns a new input vector \mathbf{x} to the same class as that of the nearest input vector \mathbf{x}_n from the training set, where in the simplest case, the distance is defined by the Euclidean metric $\|\mathbf{x} - \mathbf{x}_n\|^2$. By expressing this rule in terms of scalar products and then making use of kernel substitution, formulate the nearest-neighbour classifier for a general nonlinear kernel.
- 6.4** (*) In Appendix C, we give an example of a matrix that has positive elements but that has a negative eigenvalue and hence that is not positive definite. Find an example of the converse property, namely a 2×2 matrix with positive eigenvalues yet that has at least one negative element.
- 6.5** (*) **www** Verify the results (6.13) and (6.14) for constructing valid kernels.
- 6.6** (*) Verify the results (6.15) and (6.16) for constructing valid kernels.
- 6.7** (*) **www** Verify the results (6.17) and (6.18) for constructing valid kernels.
- 6.8** (*) Verify the results (6.19) and (6.20) for constructing valid kernels.
- 6.9** (*) Verify the results (6.21) and (6.22) for constructing valid kernels.
- 6.10** (*) Show that an excellent choice of kernel for learning a function $f(\mathbf{x})$ is given by $k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})f(\mathbf{x}')$ by showing that a linear learning machine based on this kernel will always find a solution proportional to $f(\mathbf{x})$.

6.11 (*) By making use of the expansion (6.25), and then expanding the middle factor as a power series, show that the Gaussian kernel (6.23) can be expressed as the inner product of an infinite-dimensional feature vector.

6.12 (**) **www** Consider the space of all possible subsets A of a given fixed set D . Show that the kernel function (6.27) corresponds to an inner product in a feature space of dimensionality $2^{|D|}$ defined by the mapping $\phi(A)$ where A is a subset of D and the element $\phi_U(A)$, indexed by the subset U , is given by

$$\phi_U(A) = \begin{cases} 1, & \text{if } U \subseteq A; \\ 0, & \text{otherwise.} \end{cases} \quad (6.95)$$

Here $U \subseteq A$ denotes that U is either a subset of A or is equal to A .

6.13 (*) Show that the Fisher kernel, defined by (6.33), remains invariant if we make a nonlinear transformation of the parameter vector $\theta \rightarrow \psi(\theta)$, where the function $\psi(\cdot)$ is invertible and differentiable.

6.14 (*) **www** Write down the form of the Fisher kernel, defined by (6.33), for the case of a distribution $p(\mathbf{x}|\boldsymbol{\mu}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{S})$ that is Gaussian with mean $\boldsymbol{\mu}$ and fixed covariance \mathbf{S} .

6.15 (*) By considering the determinant of a 2×2 Gram matrix, show that a positive-definite kernel function $k(x, x')$ satisfies the Cauchy-Schwartz inequality

$$k(x_1, x_2)^2 \leq k(x_1, x_1)k(x_2, x_2). \quad (6.96)$$

6.16 (**) Consider a parametric model governed by the parameter vector \mathbf{w} together with a data set of input values $\mathbf{x}_1, \dots, \mathbf{x}_N$ and a nonlinear feature mapping $\phi(\mathbf{x})$. Suppose that the dependence of the error function on \mathbf{w} takes the form

$$J(\mathbf{w}) = f(\mathbf{w}^T \phi(\mathbf{x}_1), \dots, \mathbf{w}^T \phi(\mathbf{x}_N)) + g(\mathbf{w}^T \mathbf{w}) \quad (6.97)$$

where $g(\cdot)$ is a monotonically increasing function. By writing \mathbf{w} in the form

$$\mathbf{w} = \sum_{n=1}^N \alpha_n \phi(\mathbf{x}_n) + \mathbf{w}_\perp \quad (6.98)$$

show that the value of \mathbf{w} that minimizes $J(\mathbf{w})$ takes the form of a linear combination of the basis functions $\phi(\mathbf{x}_n)$ for $n = 1, \dots, N$.

6.17 (**) **www** Consider the sum-of-squares error function (6.39) for data having noisy inputs, where $\nu(\xi)$ is the distribution of the noise. Use the calculus of variations to minimize this error function with respect to the function $y(\mathbf{x})$, and hence show that the optimal solution is given by an expansion of the form (6.40) in which the basis functions are given by (6.41).

6.18 (*) Consider a Nadaraya-Watson model with one input variable x and one target variable t having Gaussian components with isotropic covariances, so that the covariance matrix is given by $\sigma^2 \mathbf{I}$ where \mathbf{I} is the unit matrix. Write down expressions for the conditional density $p(t|x)$ and for the conditional mean $\mathbb{E}[t|x]$ and variance $\text{var}[t|x]$, in terms of the kernel function $k(x, x_n)$.

6.19 (**) Another viewpoint on kernel regression comes from a consideration of regression problems in which the input variables as well as the target variables are corrupted with additive noise. Suppose each target value t_n is generated as usual by taking a function $y(\mathbf{z}_n)$ evaluated at a point \mathbf{z}_n , and adding Gaussian noise. The value of \mathbf{z}_n is not directly observed, however, but only a noise corrupted version $\mathbf{x}_n = \mathbf{z}_n + \boldsymbol{\xi}_n$ where the random variable $\boldsymbol{\xi}$ is governed by some distribution $g(\boldsymbol{\xi})$. Consider a set of observations $\{\mathbf{x}_n, t_n\}$, where $n = 1, \dots, N$, together with a corresponding sum-of-squares error function defined by averaging over the distribution of input noise to give

$$E = \frac{1}{2} \sum_{n=1}^N \int \{y(\mathbf{x}_n - \boldsymbol{\xi}_n) - t_n\}^2 g(\boldsymbol{\xi}_n) d\boldsymbol{\xi}_n. \quad (6.99)$$

By minimizing E with respect to the function $y(\mathbf{z})$ using the calculus of variations (Appendix D), show that optimal solution for $y(\mathbf{x})$ is given by a Nadaraya-Watson kernel regression solution of the form (6.45) with a kernel of the form (6.46).

6.20 (**) **www** Verify the results (6.66) and (6.67).

6.21 (**) **www** Consider a Gaussian process regression model in which the kernel function is defined in terms of a fixed set of nonlinear basis functions. Show that the predictive distribution is identical to the result (3.58) obtained in Section 3.3.2 for the Bayesian linear regression model. To do this, note that both models have Gaussian predictive distributions, and so it is only necessary to show that the conditional mean and variance are the same. For the mean, make use of the matrix identity (C.6), and for the variance, make use of the matrix identity (C.7).

6.22 (**) Consider a regression problem with N training set input vectors $\mathbf{x}_1, \dots, \mathbf{x}_N$ and L test set input vectors $\mathbf{x}_{N+1}, \dots, \mathbf{x}_{N+L}$, and suppose we define a Gaussian process prior over functions $t(\mathbf{x})$. Derive an expression for the joint predictive distribution for $t(\mathbf{x}_{N+1}), \dots, t(\mathbf{x}_{N+L})$, given the values of $t(\mathbf{x}_1), \dots, t(\mathbf{x}_N)$. Show the marginal of this distribution for one of the test observations t_j where $N+1 \leq j \leq N+L$ is given by the usual Gaussian process regression result (6.66) and (6.67).

6.23 (**) **www** Consider a Gaussian process regression model in which the target variable \mathbf{t} has dimensionality D . Write down the conditional distribution of \mathbf{t}_{N+1} for a test input vector \mathbf{x}_{N+1} , given a training set of input vectors $\mathbf{x}_1, \dots, \mathbf{x}_{N+1}$ and corresponding target observations $\mathbf{t}_1, \dots, \mathbf{t}_{N+1}$.

6.24 (*) Show that a diagonal matrix \mathbf{W} whose elements satisfy $0 < W_{ii} < 1$ is positive definite. Show that the sum of two positive definite matrices is itself positive definite.

- 6.25** (*) **www** Using the Newton-Raphson formula (4.92), derive the iterative update formula (6.83) for finding the mode \mathbf{a}_N^* of the posterior distribution in the Gaussian process classification model.
- 6.26** (*) Using the result (2.115), derive the expressions (6.87) and (6.88) for the mean and variance of the posterior distribution $p(a_{N+1}|\mathbf{t}_N)$ in the Gaussian process classification model.
- 6.27** (***) Derive the result (6.90) for the log likelihood function in the Laplace approximation framework for Gaussian process classification. Similarly, derive the results (6.91), (6.92), and (6.94) for the terms in the gradient of the log likelihood.

7

Sparse Kernel Machines



In the previous chapter, we explored a variety of learning algorithms based on non-linear kernels. One of the significant limitations of many such algorithms is that the kernel function $k(\mathbf{x}_n, \mathbf{x}_m)$ must be evaluated for all possible pairs \mathbf{x}_n and \mathbf{x}_m of training points, which can be computationally infeasible during training and can lead to excessive computation times when making predictions for new data points. In this chapter we shall look at kernel-based algorithms that have *sparse* solutions, so that predictions for new inputs depend only on the kernel function evaluated at a subset of the training data points.

We begin by looking in some detail at the *support vector machine* (SVM), which became popular in some years ago for solving problems in classification, regression, and novelty detection. An important property of support vector machines is that the determination of the model parameters corresponds to a convex optimization problem, and so any local solution is also a global optimum. Because the discussion of support vector machines makes extensive use of Lagrange multipliers, the reader is

encouraged to review the key concepts covered in Appendix E. Additional information on support vector machines can be found in Vapnik (1995), Burges (1998), Cristianini and Shawe-Taylor (2000), Müller *et al.* (2001), Schölkopf and Smola (2002), and Herbrich (2002).

The SVM is a decision machine and so does not provide posterior probabilities. We have already discussed some of the benefits of determining probabilities in Section 1.5.4. An alternative sparse kernel technique, known as the *relevance vector machine* (RVM), is based on a Bayesian formulation and provides posterior probabilistic outputs, as well as having typically much sparser solutions than the SVM.

Section 7.2

7.1. Maximum Margin Classifiers

We begin our discussion of support vector machines by returning to the two-class classification problem using linear models of the form

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \quad (7.1)$$

where $\phi(\mathbf{x})$ denotes a fixed feature-space transformation, and we have made the bias parameter b explicit. Note that we shall shortly introduce a dual representation expressed in terms of kernel functions, which avoids having to work explicitly in feature space. The training data set comprises N input vectors $\mathbf{x}_1, \dots, \mathbf{x}_N$, with corresponding target values t_1, \dots, t_N where $t_n \in \{-1, 1\}$, and new data points \mathbf{x} are classified according to the sign of $y(\mathbf{x})$.

We shall assume for the moment that the training data set is linearly separable in feature space, so that by definition there exists at least one choice of the parameters \mathbf{w} and b such that a function of the form (7.1) satisfies $y(\mathbf{x}_n) > 0$ for points having $t_n = +1$ and $y(\mathbf{x}_n) < 0$ for points having $t_n = -1$, so that $t_n y(\mathbf{x}_n) > 0$ for all training data points.

There may of course exist many such solutions that separate the classes exactly. In Section 4.1.7, we described the perceptron algorithm that is guaranteed to find a solution in a finite number of steps. The solution that it finds, however, will be dependent on the (arbitrary) initial values chosen for \mathbf{w} and b as well as on the order in which the data points are presented. If there are multiple solutions all of which classify the training data set exactly, then we should try to find the one that will give the smallest generalization error. The support vector machine approaches this problem through the concept of the *margin*, which is defined to be the smallest distance between the decision boundary and any of the samples, as illustrated in Figure 7.1.

In support vector machines the decision boundary is chosen to be the one for which the margin is maximized. The maximum margin solution can be motivated using *computational learning theory*, also known as *statistical learning theory*. However, a simple insight into the origins of maximum margin has been given by Tong and Koller (2000) who consider a framework for classification based on a hybrid of generative and discriminative approaches. They first model the distribution over input vectors \mathbf{x} for each class using a Parzen density estimator with Gaussian kernels

Section 7.1.5

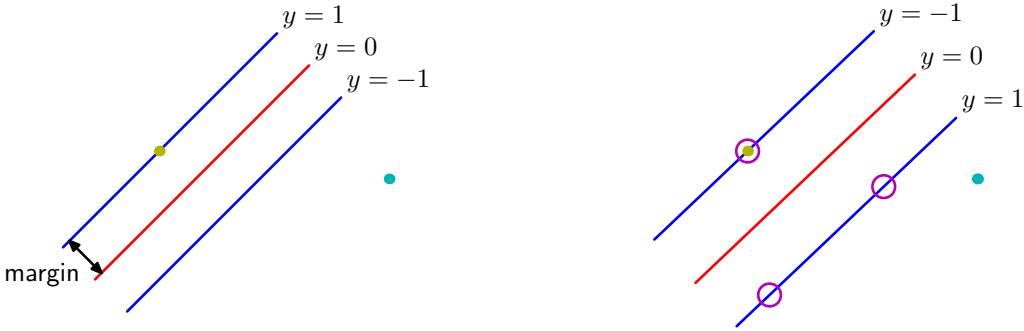


Figure 7.1 The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left figure. Maximizing the margin leads to a particular choice of decision boundary, as shown on the right. The location of this boundary is determined by a subset of the data points, known as support vectors, which are indicated by the circles.

having a common parameter σ^2 . Together with the class priors, this defines an optimal misclassification-rate decision boundary. However, instead of using this optimal boundary, they determine the best hyperplane by minimizing the probability of error relative to the learned density model. In the limit $\sigma^2 \rightarrow 0$, the optimal hyperplane is shown to be the one having maximum margin. The intuition behind this result is that as σ^2 is reduced, the hyperplane is increasingly dominated by nearby data points relative to more distant ones. In the limit, the hyperplane becomes independent of data points that are not support vectors.

We shall see in Figure 10.13 that marginalization with respect to the prior distribution of the parameters in a Bayesian approach for a simple linearly separable data set leads to a decision boundary that lies in the middle of the region separating the data points. The large margin solution has similar behaviour.

Recall from Figure 4.1 that the perpendicular distance of a point \mathbf{x} from a hyperplane defined by $y(\mathbf{x}) = 0$ where $y(\mathbf{x})$ takes the form (7.1) is given by $|y(\mathbf{x})|/\|\mathbf{w}\|$. Furthermore, we are only interested in solutions for which all data points are correctly classified, so that $t_n y(\mathbf{x}_n) > 0$ for all n . Thus the distance of a point \mathbf{x}_n to the decision surface is given by

$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}. \quad (7.2)$$

The margin is given by the perpendicular distance to the closest point \mathbf{x}_n from the data set, and we wish to optimize the parameters \mathbf{w} and b in order to maximize this distance. Thus the maximum margin solution is found by solving

$$\arg \max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n [t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b)] \right\} \quad (7.3)$$

where we have taken the factor $1/\|\mathbf{w}\|$ outside the optimization over n because \mathbf{w}

does not depend on n . Direct solution of this optimization problem would be very complex, and so we shall convert it into an equivalent problem that is much easier to solve. To do this we note that if we make the rescaling $\mathbf{w} \rightarrow \kappa\mathbf{w}$ and $b \rightarrow \kappa b$, then the distance from any point \mathbf{x}_n to the decision surface, given by $t_n y(\mathbf{x}_n)/\|\mathbf{w}\|$, is unchanged. We can use this freedom to set

$$t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b) = 1 \quad (7.4)$$

for the point that is closest to the surface. In this case, all data points will satisfy the constraints

$$t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b) \geq 1, \quad n = 1, \dots, N. \quad (7.5)$$

This is known as the canonical representation of the decision hyperplane. In the case of data points for which the equality holds, the constraints are said to be *active*, whereas for the remainder they are said to be *inactive*. By definition, there will always be at least one active constraint, because there will always be a closest point, and once the margin has been maximized there will be at least two active constraints. The optimization problem then simply requires that we maximize $\|\mathbf{w}\|^{-1}$, which is equivalent to minimizing $\|\mathbf{w}\|^2$, and so we have to solve the optimization problem

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad (7.6)$$

subject to the constraints given by (7.5). The factor of $1/2$ in (7.6) is included for later convenience. This is an example of a *quadratic programming* problem in which we are trying to minimize a quadratic function subject to a set of linear inequality constraints. It appears that the bias parameter b has disappeared from the optimization. However, it is determined implicitly via the constraints, because these require that changes to $\|\mathbf{w}\|$ be compensated by changes to b . We shall see how this works shortly.

In order to solve this constrained optimization problem, we introduce Lagrange multipliers $a_n \geq 0$, with one multiplier a_n for each of the constraints in (7.5), giving the Lagrangian function

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N a_n \{t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b) - 1\} \quad (7.7)$$

where $\mathbf{a} = (a_1, \dots, a_N)^T$. Note the minus sign in front of the Lagrange multiplier term, because we are minimizing with respect to \mathbf{w} and b , and maximizing with respect to \mathbf{a} . Setting the derivatives of $L(\mathbf{w}, b, \mathbf{a})$ with respect to \mathbf{w} and b equal to zero, we obtain the following two conditions

$$\mathbf{w} = \sum_{n=1}^N a_n t_n \phi(\mathbf{x}_n) \quad (7.8)$$

$$0 = \sum_{n=1}^N a_n t_n. \quad (7.9)$$

Eliminating \mathbf{w} and b from $L(\mathbf{w}, b, \mathbf{a})$ using these conditions then gives the *dual representation* of the maximum margin problem in which we maximize

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \quad (7.10)$$

with respect to \mathbf{a} subject to the constraints

$$a_n \geq 0, \quad n = 1, \dots, N, \quad (7.11)$$

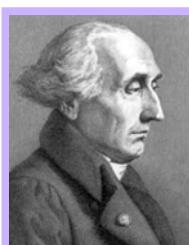
$$\sum_{n=1}^N a_n t_n = 0. \quad (7.12)$$

Here the kernel function is defined by $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$. Again, this takes the form of a quadratic programming problem in which we optimize a quadratic function of \mathbf{a} subject to a set of inequality constraints. We shall discuss techniques for solving such quadratic programming problems in Section 7.1.1.

The solution to a quadratic programming problem in M variables in general has computational complexity that is $O(M^3)$. In going to the dual formulation we have turned the original optimization problem, which involved minimizing (7.6) over M variables, into the dual problem (7.10), which has N variables. For a fixed set of basis functions whose number M is smaller than the number N of data points, the move to the dual problem appears disadvantageous. However, it allows the model to be reformulated using kernels, and so the maximum margin classifier can be applied efficiently to feature spaces whose dimensionality exceeds the number of data points, including infinite feature spaces. The kernel formulation also makes clear the role of the constraint that the kernel function $k(\mathbf{x}, \mathbf{x}')$ be positive definite, because this ensures that the Lagrangian function $\tilde{L}(\mathbf{a})$ is bounded below, giving rise to a well-defined optimization problem.

In order to classify new data points using the trained model, we evaluate the sign of $y(\mathbf{x})$ defined by (7.1). This can be expressed in terms of the parameters $\{a_n\}$ and the kernel function by substituting for \mathbf{w} using (7.8) to give

$$y(\mathbf{x}) = \sum_{n=1}^N a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b. \quad (7.13)$$



Joseph-Louis Lagrange
1736–1813

Although widely considered to be a French mathematician, Lagrange was born in Turin in Italy. By the age of nineteen, he had already made important contributions mathematics and had been appointed as Professor at the Royal Artillery School in Turin. For many

years, Euler worked hard to persuade Lagrange to move to Berlin, which he eventually did in 1766 where he succeeded Euler as Director of Mathematics at the Berlin Academy. Later he moved to Paris, narrowly escaping with his life during the French revolution thanks to the personal intervention of Lavoisier (the French chemist who discovered oxygen) who himself was later executed at the guillotine. Lagrange made key contributions to the calculus of variations and the foundations of dynamics.

In Appendix E, we show that a constrained optimization of this form satisfies the *Karush-Kuhn-Tucker* (KKT) conditions, which in this case require that the following three properties hold

$$a_n \geq 0 \quad (7.14)$$

$$t_n y(\mathbf{x}_n) - 1 \geq 0 \quad (7.15)$$

$$a_n \{t_n y(\mathbf{x}_n) - 1\} = 0. \quad (7.16)$$

Thus for every data point, either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1$. Any data point for which $a_n = 0$ will not appear in the sum in (7.13) and hence plays no role in making predictions for new data points. The remaining data points are called *support vectors*, and because they satisfy $t_n y(\mathbf{x}_n) = 1$, they correspond to points that lie on the maximum margin hyperplanes in feature space, as illustrated in Figure 7.1. This property is central to the practical applicability of support vector machines. Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained.

Having solved the quadratic programming problem and found a value for \mathbf{a} , we can then determine the value of the threshold parameter b by noting that any support vector \mathbf{x}_n satisfies $t_n y(\mathbf{x}_n) = 1$. Using (7.13) this gives

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1 \quad (7.17)$$

where \mathcal{S} denotes the set of indices of the support vectors. Although we can solve this equation for b using an arbitrarily chosen support vector \mathbf{x}_n , a numerically more stable solution is obtained by first multiplying through by t_n , making use of $t_n^2 = 1$, and then averaging these equations over all support vectors and solving for b to give

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right) \quad (7.18)$$

where $N_{\mathcal{S}}$ is the total number of support vectors.

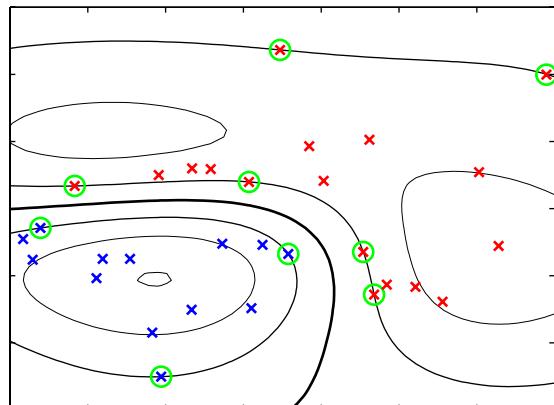
For later comparison with alternative models, we can express the maximum-margin classifier in terms of the minimization of an error function, with a simple quadratic regularizer, in the form

$$\sum_{n=1}^N E_\infty(y(\mathbf{x}_n)t_n - 1) + \lambda \|\mathbf{w}\|^2 \quad (7.19)$$

where $E_\infty(z)$ is a function that is zero if $z \geq 0$ and ∞ otherwise and ensures that the constraints (7.5) are satisfied. Note that as long as the regularization parameter satisfies $\lambda > 0$, its precise value plays no role.

Figure 7.2 shows an example of the classification resulting from training a support vector machine on a simple synthetic data set using a Gaussian kernel of the

Figure 7.2 Example of synthetic data from two classes in two dimensions showing contours of constant $y(\mathbf{x})$ obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.



form (6.23). Although the data set is not linearly separable in the two-dimensional data space \mathbf{x} , it is linearly separable in the nonlinear feature space defined implicitly by the nonlinear kernel function. Thus the training data points are perfectly separated in the original data space.

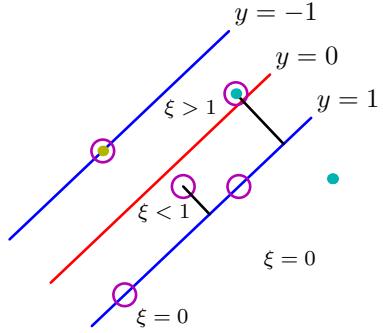
This example also provides a geometrical insight into the origin of sparsity in the SVM. The maximum margin hyperplane is defined by the location of the support vectors. Other data points can be moved around freely (so long as they remain outside the margin region) without changing the decision boundary, and so the solution will be independent of such data points.

7.1.1 Overlapping class distributions

So far, we have assumed that the training data points are linearly separable in the feature space $\phi(\mathbf{x})$. The resulting support vector machine will give exact separation of the training data in the original input space \mathbf{x} , although the corresponding decision boundary will be nonlinear. In practice, however, the class-conditional distributions may overlap, in which case exact separation of the training data can lead to poor generalization.

We therefore need a way to modify the support vector machine so as to allow some of the training points to be misclassified. From (7.19) we see that in the case of separable classes, we implicitly used an error function that gave infinite error if a data point was misclassified and zero error if it was classified correctly, and then optimized the model parameters to maximize the margin. We now modify this approach so that data points are allowed to be on the ‘wrong side’ of the margin boundary, but with a penalty that increases with the distance from that boundary. For the subsequent optimization problem, it is convenient to make this penalty a linear function of this distance. To do this, we introduce *slack variables*, $\xi_n \geq 0$ where $n = 1, \dots, N$, with one slack variable for each training data point (Bennett, 1992; Cortes and Vapnik, 1995). These are defined by $\xi_n = 0$ for data points that are on or inside the correct margin boundary and $\xi_n = |t_n - y(\mathbf{x}_n)|$ for other points. Thus a data point that is on the decision boundary $y(\mathbf{x}_n) = 0$ will have $\xi_n = 1$, and points

Figure 7.3 Illustration of the slack variables $\xi_n \geq 0$. Data points with circles around them are support vectors.



with $\xi_n > 1$ will be misclassified. The exact classification constraints (7.5) are then replaced with

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n, \quad n = 1, \dots, N \quad (7.20)$$

in which the slack variables are constrained to satisfy $\xi_n \geq 0$. Data points for which $\xi_n = 0$ are correctly classified and are either on the margin or on the correct side of the margin. Points for which $0 < \xi_n \leq 1$ lie inside the margin, but on the correct side of the decision boundary, and those data points for which $\xi_n > 1$ lie on the wrong side of the decision boundary and are misclassified, as illustrated in Figure 7.3. This is sometimes described as relaxing the hard margin constraint to give a *soft margin* and allows some of the training set data points to be misclassified. Note that while slack variables allow for overlapping class distributions, this framework is still sensitive to outliers because the penalty for misclassification increases linearly with ξ .

Our goal is now to maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary. We therefore minimize

$$C \sum_{n=1}^N \xi_n + \frac{1}{2} \|\mathbf{w}\|^2 \quad (7.21)$$

where the parameter $C > 0$ controls the trade-off between the slack variable penalty and the margin. Because any point that is misclassified has $\xi_n > 1$, it follows that $\sum_n \xi_n$ is an upper bound on the number of misclassified points. The parameter C is therefore analogous to (the inverse of) a regularization coefficient because it controls the trade-off between minimizing training errors and controlling model complexity. In the limit $C \rightarrow \infty$, we will recover the earlier support vector machine for separable data.

We now wish to minimize (7.21) subject to the constraints (7.20) together with $\xi_n \geq 0$. The corresponding Lagrangian is given by

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n - \sum_{n=1}^N a_n \{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^N \mu_n \xi_n \quad (7.22)$$

Appendix E

where $\{a_n \geq 0\}$ and $\{\mu_n \geq 0\}$ are Lagrange multipliers. The corresponding set of KKT conditions are given by

$$a_n \geq 0 \quad (7.23)$$

$$t_n y(\mathbf{x}_n) - 1 + \xi_n \geq 0 \quad (7.24)$$

$$a_n (t_n y(\mathbf{x}_n) - 1 + \xi_n) = 0 \quad (7.25)$$

$$\mu_n \geq 0 \quad (7.26)$$

$$\xi_n \geq 0 \quad (7.27)$$

$$\mu_n \xi_n = 0 \quad (7.28)$$

where $n = 1, \dots, N$.

We now optimize out \mathbf{w} , b , and $\{\xi_n\}$ making use of the definition (7.1) of $y(\mathbf{x})$ to give

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^N a_n t_n \phi(\mathbf{x}_n) \quad (7.29)$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{n=1}^N a_n t_n = 0 \quad (7.30)$$

$$\frac{\partial L}{\partial \xi_n} = 0 \Rightarrow a_n = C - \mu_n. \quad (7.31)$$

Using these results to eliminate \mathbf{w} , b , and $\{\xi_n\}$ from the Lagrangian, we obtain the dual Lagrangian in the form

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \quad (7.32)$$

which is identical to the separable case, except that the constraints are somewhat different. To see what these constraints are, we note that $a_n \geq 0$ is required because these are Lagrange multipliers. Furthermore, (7.31) together with $\mu_n \geq 0$ implies $a_n \leq C$. We therefore have to minimize (7.32) with respect to the dual variables $\{a_n\}$ subject to

$$0 \leq a_n \leq C \quad (7.33)$$

$$\sum_{n=1}^N a_n t_n = 0 \quad (7.34)$$

for $n = 1, \dots, N$, where (7.33) are known as *box constraints*. This again represents a quadratic programming problem. If we substitute (7.29) into (7.1), we see that predictions for new data points are again made by using (7.13).

We can now interpret the resulting solution. As before, a subset of the data points may have $a_n = 0$, in which case they do not contribute to the predictive

model (7.13). The remaining data points constitute the support vectors. These have $a_n > 0$ and hence from (7.25) must satisfy

$$t_n y(\mathbf{x}_n) = 1 - \xi_n. \quad (7.35)$$

If $a_n < C$, then (7.31) implies that $\mu_n > 0$, which from (7.28) requires $\xi_n = 0$ and hence such points lie on the margin. Points with $a_n = C$ can lie inside the margin and can either be correctly classified if $\xi_n \leq 1$ or misclassified if $\xi_n > 1$.

To determine the parameter b in (7.1), we note that those support vectors for which $0 < a_n < C$ have $\xi_n = 0$ so that $t_n y(\mathbf{x}_n) = 1$ and hence will satisfy

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1. \quad (7.36)$$

Again, a numerically stable solution is obtained by averaging to give

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right) \quad (7.37)$$

where \mathcal{M} denotes the set of indices of data points having $0 < a_n < C$.

An alternative, equivalent formulation of the support vector machine, known as the ν -SVM, has been proposed by Schölkopf *et al.* (2000). This involves maximizing

$$\tilde{L}(\mathbf{a}) = -\frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \quad (7.38)$$

subject to the constraints

$$0 \leq a_n \leq 1/N \quad (7.39)$$

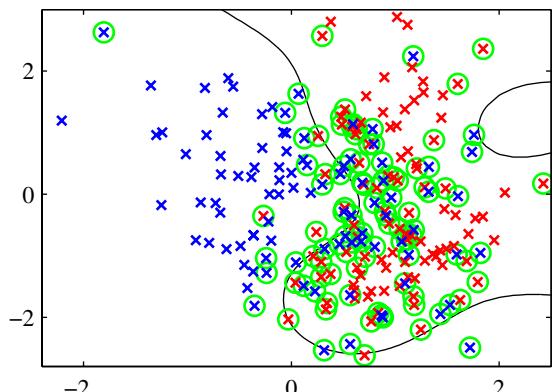
$$\sum_{n=1}^N a_n t_n = 0 \quad (7.40)$$

$$\sum_{n=1}^N a_n \geq \nu. \quad (7.41)$$

This approach has the advantage that the parameter ν , which replaces C , can be interpreted as both an upper bound on the fraction of *margin errors* (points for which $\xi_n > 0$ and hence which lie on the wrong side of the margin boundary and which may or may not be misclassified) and a lower bound on the fraction of support vectors. An example of the ν -SVM applied to a synthetic data set is shown in Figure 7.4. Here Gaussian kernels of the form $\exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$ have been used, with $\gamma = 0.45$.

Although predictions for new inputs are made using only the support vectors, the training phase (i.e., the determination of the parameters \mathbf{a} and b) makes use of the whole data set, and so it is important to have efficient algorithms for solving

Figure 7.4 Illustration of the ν -SVM applied to a nonseparable data set in two dimensions. The support vectors are indicated by circles.



the quadratic programming problem. We first note that the objective function $\tilde{L}(\mathbf{a})$ given by (7.10) or (7.32) is quadratic and so any local optimum will also be a global optimum provided the constraints define a convex region (which they do as a consequence of being linear). Direct solution of the quadratic programming problem using traditional techniques is often infeasible due to the demanding computation and memory requirements, and so more practical approaches need to be found. The technique of *chunking* (Vapnik, 1982) exploits the fact that the value of the Lagrangian is unchanged if we remove the rows and columns of the kernel matrix corresponding to Lagrange multipliers that have value zero. This allows the full quadratic programming problem to be broken down into a series of smaller ones, whose goal is eventually to identify all of the nonzero Lagrange multipliers and discard the others. Chunking can be implemented using *protected conjugate gradients* (Burges, 1998). Although chunking reduces the size of the matrix in the quadratic function from the number of data points squared to approximately the number of nonzero Lagrange multipliers squared, even this may be too big to fit in memory for large-scale applications. *Decomposition methods* (Osuna *et al.*, 1996) also solve a series of smaller quadratic programming problems but are designed so that each of these is of a fixed size, and so the technique can be applied to arbitrarily large data sets. However, it still involves numerical solution of quadratic programming subproblems and these can be problematic and expensive. One of the most popular approaches to training support vector machines is called *sequential minimal optimization*, or *SMO* (Platt, 1999). It takes the concept of chunking to the extreme limit and considers just two Lagrange multipliers at a time. In this case, the subproblem can be solved analytically, thereby avoiding numerical quadratic programming altogether. Heuristics are given for choosing the pair of Lagrange multipliers to be considered at each step. In practice, SMO is found to have a scaling with the number of data points that is somewhere between linear and quadratic depending on the particular application.

We have seen that kernel functions correspond to inner products in feature spaces that can have high, or even infinite, dimensionality. By working directly in terms of the kernel function, without introducing the feature space explicitly, it might therefore seem that support vector machines somehow manage to avoid the curse of di-

Section 1.4

mensionality. This is not the case, however, because there are constraints amongst the feature values that restrict the effective dimensionality of feature space. To see this consider a simple second-order polynomial kernel that we can expand in terms of its components

$$\begin{aligned}
 k(\mathbf{x}, \mathbf{z}) &= (1 + \mathbf{x}^T \mathbf{z})^2 = (1 + x_1 z_1 + x_2 z_2)^2 \\
 &= 1 + 2x_1 z_1 + 2x_2 z_2 + x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\
 &= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1 x_2, x_2^2)(1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, \sqrt{2}z_1 z_2, z_2^2)^T \\
 &= \phi(\mathbf{x})^T \phi(\mathbf{z}).
 \end{aligned} \tag{7.42}$$

This kernel function therefore represents an inner product in a feature space having six dimensions, in which the mapping from input space to feature space is described by the vector function $\phi(\mathbf{x})$. However, the coefficients weighting these different features are constrained to have specific forms. Thus any set of points in the original two-dimensional space \mathbf{x} would be constrained to lie exactly on a two-dimensional nonlinear manifold embedded in the six-dimensional feature space.

We have already highlighted the fact that the support vector machine does not provide probabilistic outputs but instead makes classification decisions for new input vectors. Veropoulos *et al.* (1999) discuss modifications to the SVM to allow the trade-off between false positive and false negative errors to be controlled. However, if we wish to use the SVM as a module in a larger probabilistic system, then probabilistic predictions of the class label t for new inputs \mathbf{x} are required.

To address this issue, Platt (2000) has proposed fitting a logistic sigmoid to the outputs of a previously trained support vector machine. Specifically, the required conditional probability is assumed to be of the form

$$p(t = 1|\mathbf{x}) = \sigma(Ay(\mathbf{x}) + B) \tag{7.43}$$

where $y(\mathbf{x})$ is defined by (7.1). Values for the parameters A and B are found by minimizing the cross-entropy error function defined by a training set consisting of pairs of values $y(\mathbf{x}_n)$ and t_n . The data used to fit the sigmoid needs to be independent of that used to train the original SVM in order to avoid severe over-fitting. This two-stage approach is equivalent to assuming that the output $y(\mathbf{x})$ of the support vector machine represents the log-odds of \mathbf{x} belonging to class $t = 1$. Because the SVM training procedure is not specifically intended to encourage this, the SVM can give a poor approximation to the posterior probabilities (Tipping, 2001).

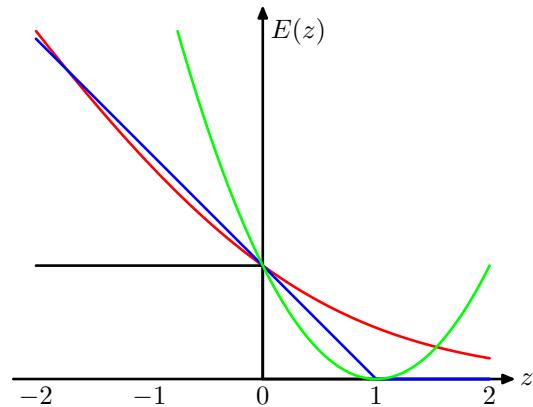
7.1.2 Relation to logistic regression

As with the separable case, we can re-cast the SVM for nonseparable distributions in terms of the minimization of a regularized error function. This will also allow us to highlight similarities, and differences, compared to the logistic regression model.

We have seen that for data points that are on the correct side of the margin boundary, and which therefore satisfy $y_n t_n \geq 1$, we have $\xi_n = 0$, and for the

Section 4.3.2

Figure 7.5 Plot of the ‘hinge’ error function used in support vector machines, shown in blue, along with the error function for logistic regression, rescaled by a factor of $1/\ln(2)$ so that it passes through the point $(0, 1)$, shown in red. Also shown are the misclassification error in black and the squared error in green.



remaining points we have $\xi_n = 1 - y_n t_n$. Thus the objective function (7.21) can be written (up to an overall multiplicative constant) in the form

$$\sum_{n=1}^N E_{\text{SV}}(y_n t_n) + \lambda \|\mathbf{w}\|^2 \quad (7.44)$$

where $\lambda = (2C)^{-1}$, and $E_{\text{SV}}(\cdot)$ is the *hinge* error function defined by

$$E_{\text{SV}}(y_n t_n) = [1 - y_n t_n]_+ \quad (7.45)$$

where $[\cdot]_+$ denotes the positive part. The hinge error function, so-called because of its shape, is plotted in Figure 7.5. It can be viewed as an approximation to the misclassification error, i.e., the error function that ideally we would like to minimize, which is also shown in Figure 7.5.

When we considered the logistic regression model in Section 4.3.2, we found it convenient to work with target variable $t \in \{0, 1\}$. For comparison with the support vector machine, we first reformulate maximum likelihood logistic regression using the target variable $t \in \{-1, 1\}$. To do this, we note that $p(t = 1|y) = \sigma(y)$ where $y(\mathbf{x})$ is given by (7.1), and $\sigma(y)$ is the logistic sigmoid function defined by (4.59). It follows that $p(t = -1|y) = 1 - \sigma(y) = \sigma(-y)$, where we have used the properties of the logistic sigmoid function, and so we can write

$$p(t|y) = \sigma(yt). \quad (7.46)$$

From this we can construct an error function by taking the negative logarithm of the likelihood function that, with a quadratic regularizer, takes the form

$$\sum_{n=1}^N E_{\text{LR}}(y_n t_n) + \lambda \|\mathbf{w}\|^2. \quad (7.47)$$

where

$$E_{\text{LR}}(yt) = \ln(1 + \exp(-yt)). \quad (7.48)$$

Exercise 7.6

For comparison with other error functions, we can divide by $\ln(2)$ so that the error function passes through the point $(0, 1)$. This rescaled error function is also plotted in Figure 7.5 and we see that it has a similar form to the support vector error function. The key difference is that the flat region in $E_{SV}(yt)$ leads to sparse solutions.

Both the logistic error and the hinge loss can be viewed as continuous approximations to the misclassification error. Another continuous error function that has sometimes been used to solve classification problems is the squared error, which is again plotted in Figure 7.5. It has the property, however, of placing increasing emphasis on data points that are correctly classified but that are a long way from the decision boundary on the correct side. Such points will be strongly weighted at the expense of misclassified points, and so if the objective is to minimize the misclassification rate, then a monotonically decreasing error function would be a better choice.

7.1.3 Multiclass SVMs

The support vector machine is fundamentally a two-class classifier. In practice, however, we often have to tackle problems involving $K > 2$ classes. Various methods have therefore been proposed for combining multiple two-class SVMs in order to build a multiclass classifier.

One commonly used approach (Vapnik, 1998) is to construct K separate SVMs, in which the k^{th} model $y_k(\mathbf{x})$ is trained using the data from class C_k as the positive examples and the data from the remaining $K - 1$ classes as the negative examples. This is known as the *one-versus-the-rest* approach. However, in Figure 4.2 we saw that using the decisions of the individual classifiers can lead to inconsistent results in which an input is assigned to multiple classes simultaneously. This problem is sometimes addressed by making predictions for new inputs \mathbf{x} using

$$y(\mathbf{x}) = \max_k y_k(\mathbf{x}). \quad (7.49)$$

Unfortunately, this heuristic approach suffers from the problem that the different classifiers were trained on different tasks, and there is no guarantee that the real-valued quantities $y_k(\mathbf{x})$ for different classifiers will have appropriate scales.

Another problem with the one-versus-the-rest approach is that the training sets are imbalanced. For instance, if we have ten classes each with equal numbers of training data points, then the individual classifiers are trained on data sets comprising 90% negative examples and only 10% positive examples, and the symmetry of the original problem is lost. A variant of the one-versus-the-rest scheme was proposed by Lee *et al.* (2001) who modify the target values so that the positive class has target +1 and the negative class has target $-1/(K - 1)$.

Weston and Watkins (1999) define a single objective function for training all K SVMs simultaneously, based on maximizing the margin from each to remaining classes. However, this can result in much slower training because, instead of solving K separate optimization problems each over N data points with an overall cost of $O(KN^2)$, a single optimization problem of size $(K - 1)N$ must be solved giving an overall cost of $O(K^2N^2)$.

Another approach is to train $K(K - 1)/2$ different 2-class SVMs on all possible pairs of classes, and then to classify test points according to which class has the highest number of ‘votes’, an approach that is sometimes called *one-versus-one*. Again, we saw in Figure 4.2 that this can lead to ambiguities in the resulting classification. Also, for large K this approach requires significantly more training time than the one-versus-the-rest approach. Similarly, to evaluate test points, significantly more computation is required.

The latter problem can be alleviated by organizing the pairwise classifiers into a directed acyclic graph (not to be confused with a probabilistic graphical model) leading to the DAGSVM (Platt *et al.*, 2000). For K classes, the DAGSVM has a total of $K(K - 1)/2$ classifiers, and to classify a new test point only $K - 1$ pairwise classifiers need to be evaluated, with the particular classifiers used depending on which path through the graph is traversed.

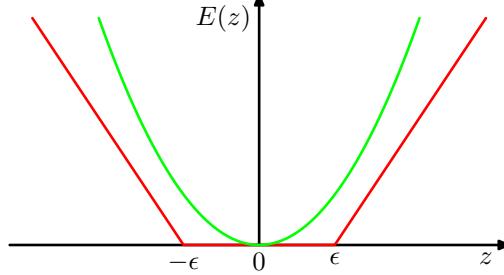
A different approach to multiclass classification, based on error-correcting output codes, was developed by Dietterich and Bakiri (1995) and applied to support vector machines by Allwein *et al.* (2000). This can be viewed as a generalization of the voting scheme of the one-versus-one approach in which more general partitions of the classes are used to train the individual classifiers. The K classes themselves are represented as particular sets of responses from the two-class classifiers chosen, and together with a suitable decoding scheme, this gives robustness to errors and to ambiguity in the outputs of the individual classifiers. Although the application of SVMs to multiclass classification problems remains an open issue, in practice the one-versus-the-rest approach is the most widely used in spite of its ad-hoc formulation and its practical limitations.

There are also *single-class* support vector machines, which solve an unsupervised learning problem related to probability density estimation. Instead of modelling the density of data, however, these methods aim to find a smooth boundary enclosing a region of high density. The boundary is chosen to represent a quantile of the density, that is, the probability that a data point drawn from the distribution will land inside that region is given by a fixed number between 0 and 1 that is specified in advance. This is a more restricted problem than estimating the full density but may be sufficient in specific applications. Two approaches to this problem using support vector machines have been proposed. The algorithm of Schölkopf *et al.* (2001) tries to find a hyperplane that separates all but a fixed fraction ν of the training data from the origin while at the same time maximizing the distance (margin) of the hyperplane from the origin, while Tax and Duin (1999) look for the smallest sphere in feature space that contains all but a fraction ν of the data points. For kernels $k(\mathbf{x}, \mathbf{x}')$ that are functions only of $\mathbf{x} - \mathbf{x}'$, the two algorithms are equivalent.

7.1.4 SVMs for regression

We now extend support vector machines to regression problems while at the same time preserving the property of sparseness. In simple linear regression, we

Figure 7.6 Plot of an ϵ -insensitive error function (in red) in which the error increases linearly with distance beyond the insensitive region. Also shown for comparison is the quadratic error function (in green).



minimize a regularized error function given by

$$\frac{1}{2} \sum_{n=1}^N \{y_n - t_n\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2. \quad (7.50)$$

To obtain sparse solutions, the quadratic error function is replaced by an ϵ -insensitive error function (Vapnik, 1995), which gives zero error if the absolute difference between the prediction $y(\mathbf{x})$ and the target t is less than ϵ where $\epsilon > 0$. A simple example of an ϵ -insensitive error function, having a linear cost associated with errors outside the insensitive region, is given by

$$E_\epsilon(y(\mathbf{x}) - t) = \begin{cases} 0, & \text{if } |y(\mathbf{x}) - t| < \epsilon; \\ |y(\mathbf{x}) - t| - \epsilon, & \text{otherwise} \end{cases} \quad (7.51)$$

and is illustrated in Figure 7.6.

We therefore minimize a regularized error function given by

$$C \sum_{n=1}^N E_\epsilon(y(\mathbf{x}_n) - t_n) + \frac{1}{2} \|\mathbf{w}\|^2 \quad (7.52)$$

where $y(\mathbf{x})$ is given by (7.1). By convention the (inverse) regularization parameter, denoted C , appears in front of the error term.

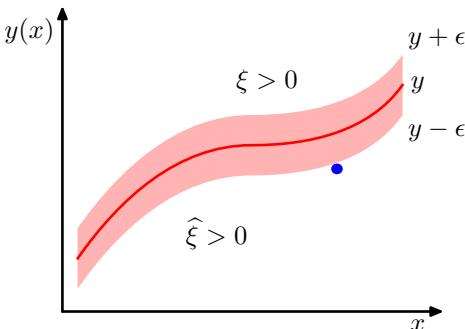
As before, we can re-express the optimization problem by introducing slack variables. For each data point \mathbf{x}_n , we now need two slack variables $\xi_n \geq 0$ and $\hat{\xi}_n \geq 0$, where $\xi_n > 0$ corresponds to a point for which $t_n > y(\mathbf{x}_n) + \epsilon$, and $\hat{\xi}_n > 0$ corresponds to a point for which $t_n < y(\mathbf{x}_n) - \epsilon$, as illustrated in Figure 7.7.

The condition for a target point to lie inside the ϵ -tube is that $y_n - \epsilon \leq t_n \leq y_n + \epsilon$, where $y_n = y(\mathbf{x}_n)$. Introducing the slack variables allows points to lie outside the tube provided the slack variables are nonzero, and the corresponding conditions are

$$t_n \leq y(\mathbf{x}_n) + \epsilon + \xi_n \quad (7.53)$$

$$t_n \geq y(\mathbf{x}_n) - \epsilon - \hat{\xi}_n. \quad (7.54)$$

Figure 7.7 Illustration of SVM regression, showing the regression curve together with the ϵ -insensitive ‘tube’. Also shown are examples of the slack variables ξ and $\hat{\xi}$. Points above the ϵ -tube have $\xi > 0$ and $\hat{\xi} = 0$, points below the ϵ -tube have $\xi = 0$ and $\hat{\xi} > 0$, and points inside the ϵ -tube have $\xi = \hat{\xi} = 0$.



The error function for support vector regression can then be written as

$$C \sum_{n=1}^N (\xi_n + \hat{\xi}_n) + \frac{1}{2} \|\mathbf{w}\|^2 \quad (7.55)$$

which must be minimized subject to the constraints $\xi_n \geq 0$ and $\hat{\xi}_n \geq 0$ as well as (7.53) and (7.54). This can be achieved by introducing Lagrange multipliers $a_n \geq 0$, $\hat{a}_n \geq 0$, $\mu_n \geq 0$, and $\hat{\mu}_n \geq 0$ and optimizing the Lagrangian

$$\begin{aligned} L = & C \sum_{n=1}^N (\xi_n + \hat{\xi}_n) + \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N (\mu_n \xi_n + \hat{\mu}_n \hat{\xi}_n) \\ & - \sum_{n=1}^N a_n (\epsilon + \xi_n + y_n - t_n) - \sum_{n=1}^N \hat{a}_n (\epsilon + \hat{\xi}_n - y_n + t_n). \end{aligned} \quad (7.56)$$

We now substitute for $y(\mathbf{x})$ using (7.1) and then set the derivatives of the Lagrangian with respect to \mathbf{w} , b , ξ_n , and $\hat{\xi}_n$ to zero, giving

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^N (a_n - \hat{a}_n) \phi(\mathbf{x}_n) \quad (7.57)$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{n=1}^N (a_n - \hat{a}_n) = 0 \quad (7.58)$$

$$\frac{\partial L}{\partial \xi_n} = 0 \Rightarrow a_n + \mu_n = C \quad (7.59)$$

$$\frac{\partial L}{\partial \hat{\xi}_n} = 0 \Rightarrow \hat{a}_n + \hat{\mu}_n = C. \quad (7.60)$$

Using these results to eliminate the corresponding variables from the Lagrangian, we see that the dual problem involves maximizing

$$\begin{aligned}\tilde{L}(\mathbf{a}, \hat{\mathbf{a}}) &= -\frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N (a_n - \hat{a}_n)(a_m - \hat{a}_m) k(\mathbf{x}_n, \mathbf{x}_m) \\ &\quad - \epsilon \sum_{n=1}^N (a_n + \hat{a}_n) + \sum_{n=1}^N (a_n - \hat{a}_n)t_n\end{aligned}\tag{7.61}$$

with respect to $\{a_n\}$ and $\{\hat{a}_n\}$, where we have introduced the kernel $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$. Again, this is a constrained maximization, and to find the constraints we note that $a_n \geq 0$ and $\hat{a}_n \geq 0$ are both required because these are Lagrange multipliers. Also $\mu_n \geq 0$ and $\hat{\mu}_n \geq 0$ together with (7.59) and (7.60), require $a_n \leq C$ and $\hat{a}_n \leq C$, and so again we have the box constraints

$$0 \leq a_n \leq C \tag{7.62}$$

$$0 \leq \hat{a}_n \leq C \tag{7.63}$$

together with the condition (7.58).

Substituting (7.57) into (7.1), we see that predictions for new inputs can be made using

$$y(\mathbf{x}) = \sum_{n=1}^N (a_n - \hat{a}_n) k(\mathbf{x}, \mathbf{x}_n) + b \tag{7.64}$$

which is again expressed in terms of the kernel function.

The corresponding Karush-Kuhn-Tucker (KKT) conditions, which state that at the solution the product of the dual variables and the constraints must vanish, are given by

$$a_n(\epsilon + \xi_n + y_n - t_n) = 0 \tag{7.65}$$

$$\hat{a}_n(\epsilon + \hat{\xi}_n - y_n + t_n) = 0 \tag{7.66}$$

$$(C - a_n)\xi_n = 0 \tag{7.67}$$

$$(C - \hat{a}_n)\hat{\xi}_n = 0. \tag{7.68}$$

From these we can obtain several useful results. First of all, we note that a coefficient a_n can only be nonzero if $\epsilon + \xi_n + y_n - t_n = 0$, which implies that the data point either lies on the upper boundary of the ϵ -tube ($\xi_n = 0$) or lies above the upper boundary ($\xi_n > 0$). Similarly, a nonzero value for \hat{a}_n implies $\epsilon + \hat{\xi}_n - y_n + t_n = 0$, and such points must lie either on or below the lower boundary of the ϵ -tube.

Furthermore, the two constraints $\epsilon + \xi_n + y_n - t_n = 0$ and $\epsilon + \hat{\xi}_n - y_n + t_n = 0$ are incompatible, as is easily seen by adding them together and noting that ξ_n and $\hat{\xi}_n$ are nonnegative while ϵ is strictly positive, and so for every data point \mathbf{x}_n , either a_n or \hat{a}_n (or both) must be zero.

The support vectors are those data points that contribute to predictions given by (7.64), in other words those for which either $a_n \neq 0$ or $\hat{a}_n \neq 0$. These are points that lie on the boundary of the ϵ -tube or outside the tube. All points within the tube have

$a_n = \hat{a}_n = 0$. We again have a sparse solution, and the only terms that have to be evaluated in the predictive model (7.64) are those that involve the support vectors.

The parameter b can be found by considering a data point for which $0 < a_n < C$, which from (7.67) must have $\xi_n = 0$, and from (7.65) must therefore satisfy $\epsilon + y_n - t_n = 0$. Using (7.1) and solving for b , we obtain

$$\begin{aligned} b &= t_n - \epsilon - \mathbf{w}^T \phi(\mathbf{x}_n) \\ &= t_n - \epsilon - \sum_{m=1}^N (a_m - \hat{a}_m) k(\mathbf{x}_n, \mathbf{x}_m) \end{aligned} \quad (7.69)$$

where we have used (7.57). We can obtain an analogous result by considering a point for which $0 < \hat{a}_n < C$. In practice, it is better to average over all such estimates of b .

As with the classification case, there is an alternative formulation of the SVM for regression in which the parameter governing complexity has a more intuitive interpretation (Schölkopf *et al.*, 2000). In particular, instead of fixing the width ϵ of the insensitive region, we fix instead a parameter ν that bounds the fraction of points lying outside the tube. This involves maximizing

$$\begin{aligned} \tilde{L}(\mathbf{a}, \hat{\mathbf{a}}) &= -\frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N (a_n - \hat{a}_n)(a_m - \hat{a}_m) k(\mathbf{x}_n, \mathbf{x}_m) \\ &\quad + \sum_{n=1}^N (a_n - \hat{a}_n)t_n \end{aligned} \quad (7.70)$$

subject to the constraints

$$0 \leq a_n \leq C/N \quad (7.71)$$

$$0 \leq \hat{a}_n \leq C/N \quad (7.72)$$

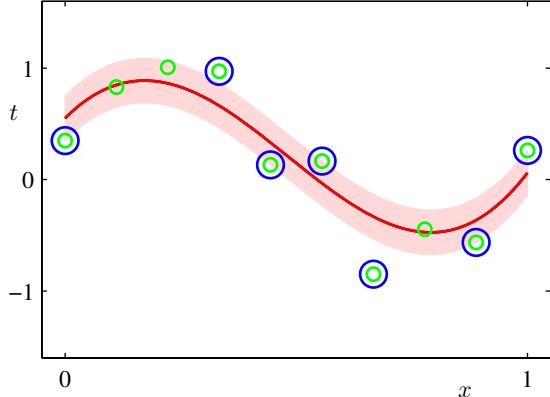
$$\sum_{n=1}^N (a_n - \hat{a}_n) = 0 \quad (7.73)$$

$$\sum_{n=1}^N (a_n + \hat{a}_n) \leq \nu C. \quad (7.74)$$

It can be shown that there are at most νN data points falling outside the insensitive tube, while at least νN data points are support vectors and so lie either on the tube or outside it.

The use of a support vector machine to solve a regression problem is illustrated using the sinusoidal data set in Figure 7.8. Here the parameters ν and C have been chosen by hand. In practice, their values would typically be determined by cross-validation.

Figure 7.8 Illustration of the ν -SVM for regression applied to the sinusoidal synthetic data set using Gaussian kernels. The predicted regression curve is shown by the red line, and the ϵ -insensitive tube corresponds to the shaded region. Also, the data points are shown in green, and those with support vectors are indicated by blue circles.



7.1.5 Computational learning theory

Historically, support vector machines have largely been motivated and analysed using a theoretical framework known as *computational learning theory*, also sometimes called *statistical learning theory* (Anthony and Biggs, 1992; Kearns and Vazirani, 1994; Vapnik, 1995; Vapnik, 1998). This has its origins with Valiant (1984) who formulated the *probably approximately correct*, or PAC, learning framework. The goal of the PAC framework is to understand how large a data set needs to be in order to give good generalization. It also gives bounds for the computational cost of learning, although we do not consider these here.

Suppose that a data set \mathcal{D} of size N is drawn from some joint distribution $p(\mathbf{x}, \mathbf{t})$ where \mathbf{x} is the input variable and \mathbf{t} represents the class label, and that we restrict attention to ‘noise free’ situations in which the class labels are determined by some (unknown) deterministic function $\mathbf{t} = \mathbf{g}(\mathbf{x})$. In PAC learning we say that a function $\mathbf{f}(\mathbf{x}; \mathcal{D})$, drawn from a space \mathcal{F} of such functions on the basis of the training set \mathcal{D} , has good generalization if its expected error rate is below some pre-specified threshold ϵ , so that

$$\mathbb{E}_{\mathbf{x}, \mathbf{t}} [I(\mathbf{f}(\mathbf{x}; \mathcal{D}) \neq \mathbf{t})] < \epsilon \quad (7.75)$$

where $I(\cdot)$ is the indicator function, and the expectation is with respect to the distribution $p(\mathbf{x}, \mathbf{t})$. The quantity on the left-hand side is a random variable, because it depends on the training set \mathcal{D} , and the PAC framework requires that (7.75) holds, with probability greater than $1 - \delta$, for a data set \mathcal{D} drawn randomly from $p(\mathbf{x}, \mathbf{t})$. Here δ is another pre-specified parameter, and the terminology ‘probably approximately correct’ comes from the requirement that with high probability (greater than $1 - \delta$), the error rate be small (less than ϵ). For a given choice of model space \mathcal{F} , and for given parameters ϵ and δ , PAC learning aims to provide bounds on the minimum size N of data set needed to meet this criterion. A key quantity in PAC learning is the *Vapnik-Chervonenkis dimension*, or VC dimension, which provides a measure of the complexity of a space of functions, and which allows the PAC framework to be extended to spaces containing an infinite number of functions.

The bounds derived within the PAC framework are often described as worst-

case, because they apply to *any* choice for the distribution $p(\mathbf{x}, t)$, so long as both the training and the test examples are drawn (independently) from the same distribution, and for *any* choice for the function $f(\mathbf{x})$ so long as it belongs to \mathcal{F} . In real-world applications of machine learning, we deal with distributions that have significant regularity, for example in which large regions of input space carry the same class label. As a consequence of the lack of any assumptions about the form of the distribution, the PAC bounds are very conservative, in other words they strongly over-estimate the size of data sets required to achieve a given generalization performance. For this reason, PAC bounds have found few, if any, practical applications.

One attempt to improve the tightness of the PAC bounds is the *PAC-Bayesian* framework (McAllester, 2003), which considers a distribution over the space \mathcal{F} of functions, somewhat analogous to the prior in a Bayesian treatment. This still considers any possible choice for $p(\mathbf{x}, t)$, and so although the bounds are tighter, they are still very conservative.

7.2. Relevance Vector Machines

Support vector machines have been used in a variety of classification and regression applications. Nevertheless, they suffer from a number of limitations, several of which have been highlighted already in this chapter. In particular, the outputs of an SVM represent decisions rather than posterior probabilities. Also, the SVM was originally formulated for two classes, and the extension to $K > 2$ classes is problematic. There is a complexity parameter C , or ν (as well as a parameter ϵ in the case of regression), that must be found using a hold-out method such as cross-validation. Finally, predictions are expressed as linear combinations of kernel functions that are centred on training data points and that are required to be positive definite.

The *relevance vector machine* or RVM (Tipping, 2001) is a Bayesian sparse kernel technique for regression and classification that shares many of the characteristics of the SVM whilst avoiding its principal limitations. Additionally, it typically leads to much sparser models resulting in correspondingly faster performance on test data whilst maintaining comparable generalization error.

In contrast to the SVM we shall find it more convenient to introduce the regression form of the RVM first and then consider the extension to classification tasks.

7.2.1 RVM for regression

The relevance vector machine for regression is a linear model of the form studied in Chapter 3 but with a modified prior that results in sparse solutions. The model defines a conditional distribution for a real-valued target variable t , given an input vector \mathbf{x} , which takes the form

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}), \beta^{-1}) \quad (7.76)$$

where $\beta = \sigma^{-2}$ is the noise precision (inverse noise variance), and the mean is given by a linear model of the form

$$y(\mathbf{x}) = \sum_{i=1}^M w_i \phi_i(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \quad (7.77)$$

with fixed nonlinear basis functions $\phi_i(\mathbf{x})$, which will typically include a constant term so that the corresponding weight parameter represents a ‘bias’.

The relevance vector machine is a specific instance of this model, which is intended to mirror the structure of the support vector machine. In particular, the basis functions are given by kernels, with one kernel associated with each of the data points from the training set. The general expression (7.77) then takes the SVM-like form

$$y(\mathbf{x}) = \sum_{n=1}^N w_n k(\mathbf{x}, \mathbf{x}_n) + b \quad (7.78)$$

where b is a bias parameter. The number of parameters in this case is $M = N + 1$, and $y(\mathbf{x})$ has the same form as the predictive model (7.64) for the SVM, except that the coefficients a_n are here denoted w_n . It should be emphasized that the subsequent analysis is valid for arbitrary choices of basis function, and for generality we shall work with the form (7.77). In contrast to the SVM, there is no restriction to positive-definite kernels, nor are the basis functions tied in either number or location to the training data points.

Suppose we are given a set of N observations of the input vector \mathbf{x} , which we denote collectively by a data matrix \mathbf{X} whose n^{th} row is \mathbf{x}_n^T with $n = 1, \dots, N$. The corresponding target values are given by $\mathbf{t} = (t_1, \dots, t_N)^T$. Thus, the likelihood function is given by

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N p(t_n|\mathbf{x}_n, \mathbf{w}, \beta^{-1}). \quad (7.79)$$

Next we introduce a prior distribution over the parameter vector \mathbf{w} and as in Chapter 3, we shall consider a zero-mean Gaussian prior. However, the key difference in the RVM is that we introduce a separate hyperparameter α_i for each of the weight parameters w_i instead of a single shared hyperparameter. Thus the weight prior takes the form

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{i=1}^M \mathcal{N}(w_i|0, \alpha_i^{-1}) \quad (7.80)$$

where α_i represents the precision of the corresponding parameter w_i , and $\boldsymbol{\alpha}$ denotes $(\alpha_1, \dots, \alpha_M)^T$. We shall see that, when we maximize the evidence with respect to these hyperparameters, a significant proportion of them go to infinity, and the corresponding weight parameters have posterior distributions that are concentrated at zero. The basis functions associated with these parameters therefore play no role

in the predictions made by the model and so are effectively pruned out, resulting in a sparse model.

Using the result (3.49) for linear regression models, we see that the posterior distribution for the weights is again Gaussian and takes the form

$$p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \boldsymbol{\alpha}, \beta) = \mathcal{N}(\mathbf{w}|\mathbf{m}, \boldsymbol{\Sigma}) \quad (7.81)$$

where the mean and covariance are given by

$$\mathbf{m} = \beta \boldsymbol{\Sigma} \boldsymbol{\Phi}^T \mathbf{t} \quad (7.82)$$

$$\boldsymbol{\Sigma} = (\mathbf{A} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \quad (7.83)$$

where $\boldsymbol{\Phi}$ is the $N \times M$ design matrix with elements $\Phi_{ni} = \phi_i(\mathbf{x}_n)$, and $\mathbf{A} = \text{diag}(\alpha_i)$. Note that in the specific case of the model (7.78), we have $\boldsymbol{\Phi} = \mathbf{K}$, where \mathbf{K} is the symmetric $(N+1) \times (N+1)$ kernel matrix with elements $k(\mathbf{x}_n, \mathbf{x}_m)$.

The values of $\boldsymbol{\alpha}$ and β are determined using type-2 maximum likelihood, also known as the *evidence approximation*, in which we maximize the marginal likelihood function obtained by integrating out the weight parameters

$$p(\mathbf{t}|\mathbf{X}, \boldsymbol{\alpha}, \beta) = \int p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) p(\mathbf{w}|\boldsymbol{\alpha}) d\mathbf{w}. \quad (7.84)$$

Section 3.5

Exercise 7.10

Because this represents the convolution of two Gaussians, it is readily evaluated to give the log marginal likelihood in the form

$$\begin{aligned} \ln p(\mathbf{t}|\mathbf{X}, \boldsymbol{\alpha}, \beta) &= \ln \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C}) \\ &= -\frac{1}{2} \{ N \ln(2\pi) + \ln |\mathbf{C}| + \mathbf{t}^T \mathbf{C}^{-1} \mathbf{t} \} \end{aligned} \quad (7.85)$$

where $\mathbf{t} = (t_1, \dots, t_N)^T$, and we have defined the $N \times N$ matrix \mathbf{C} given by

$$\mathbf{C} = \beta^{-1} \mathbf{I} + \boldsymbol{\Phi} \mathbf{A}^{-1} \boldsymbol{\Phi}^T. \quad (7.86)$$

Our goal is now to maximize (7.85) with respect to the hyperparameters $\boldsymbol{\alpha}$ and β . This requires only a small modification to the results obtained in Section 3.5 for the evidence approximation in the linear regression model. Again, we can identify two approaches. In the first, we simply set the required derivatives of the marginal likelihood to zero and obtain the following re-estimation equations

$$\alpha_i^{\text{new}} = \frac{\gamma_i}{m_i^2} \quad (7.87)$$

$$(\beta^{\text{new}})^{-1} = \frac{\|\mathbf{t} - \boldsymbol{\Phi} \mathbf{m}\|^2}{N - \sum_i \gamma_i} \quad (7.88)$$

where m_i is the i^{th} component of the posterior mean \mathbf{m} defined by (7.82). The quantity γ_i measures how well the corresponding parameter w_i is determined by the data and is defined by

Section 3.5.3

$$\gamma_i = 1 - \alpha_i \Sigma_{ii} \quad (7.89)$$

in which Σ_{ii} is the i^{th} diagonal component of the posterior covariance Σ given by (7.83). Learning therefore proceeds by choosing initial values for α and β , evaluating the mean and covariance of the posterior using (7.82) and (7.83), respectively, and then alternately re-estimating the hyperparameters, using (7.87) and (7.88), and re-estimating the posterior mean and covariance, using (7.82) and (7.83), until a suitable convergence criterion is satisfied.

The second approach is to use the EM algorithm, and is discussed in Section 9.3.4. These two approaches to finding the values of the hyperparameters that maximize the evidence are formally equivalent. Numerically, however, it is found that the direct optimization approach corresponding to (7.87) and (7.88) gives somewhat faster convergence (Tipping, 2001).

As a result of the optimization, we find that a proportion of the hyperparameters $\{\alpha_i\}$ are driven to large (in principle infinite) values, and so the weight parameters w_i corresponding to these hyperparameters have posterior distributions with mean and variance both zero. Thus those parameters, and the corresponding basis functions $\phi_i(\mathbf{x})$, are removed from the model and play no role in making predictions for new inputs. In the case of models of the form (7.78), the inputs \mathbf{x}_n corresponding to the remaining nonzero weights are called *relevance vectors*, because they are identified through the mechanism of automatic relevance determination, and are analogous to the support vectors of an SVM. It is worth emphasizing, however, that this mechanism for achieving sparsity in probabilistic models through automatic relevance determination is quite general and can be applied to any model expressed as an adaptive linear combination of basis functions.

Having found values α^* and β^* for the hyperparameters that maximize the marginal likelihood, we can evaluate the predictive distribution over t for a new input \mathbf{x} . Using (7.76) and (7.81), this is given by

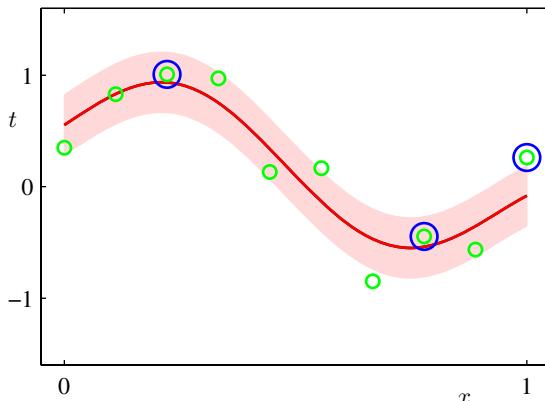
$$\begin{aligned} p(t|\mathbf{x}, \mathbf{X}, \mathbf{t}, \alpha^*, \beta^*) &= \int p(t|\mathbf{x}, \mathbf{w}, \beta^*) p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \alpha^*, \beta^*) d\mathbf{w} \\ &= \mathcal{N}(t|\mathbf{m}^T \phi(\mathbf{x}), \sigma^2(\mathbf{x})). \end{aligned} \quad (7.90)$$

Thus the predictive mean is given by (7.76) with \mathbf{w} set equal to the posterior mean \mathbf{m} , and the variance of the predictive distribution is given by

$$\sigma^2(\mathbf{x}) = (\beta^*)^{-1} + \phi(\mathbf{x})^T \Sigma \phi(\mathbf{x}) \quad (7.91)$$

where Σ is given by (7.83) in which α and β are set to their optimized values α^* and β^* . This is just the familiar result (3.59) obtained in the context of linear regression. Recall that for localized basis functions, the predictive variance for linear regression models becomes small in regions of input space where there are no basis functions. In the case of an RVM with the basis functions centred on data points, the model will therefore become increasingly certain of its predictions when extrapolating outside the domain of the data (Rasmussen and Quiñonero-Candela, 2005), which of course is undesirable. The predictive distribution in Gaussian process regression does not

Figure 7.9 Illustration of RVM regression using the same data set, and the same Gaussian kernel functions, as used in Figure 7.8 for the ν -SVM regression model. The mean of the predictive distribution for the RVM is shown by the red line, and the one standard-deviation predictive distribution is shown by the shaded region. Also, the data points are shown in green, and the relevance vectors are indicated by blue circles. Note that there are only 3 relevance vectors compared to 7 support vectors for the ν -SVM in Figure 7.8.



suffer from this problem. However, the computational cost of making predictions with a Gaussian processes is typically much higher than with an RVM.

Figure 7.9 shows an example of the RVM applied to the sinusoidal regression data set. Here the noise precision parameter β is also determined through evidence maximization. We see that the number of relevance vectors in the RVM is significantly smaller than the number of support vectors used by the SVM. For a wide range of regression and classification tasks, the RVM is found to give models that are typically an order of magnitude more compact than the corresponding support vector machine, resulting in a significant improvement in the speed of processing on test data. Remarkably, this greater sparsity is achieved with little or no reduction in generalization error compared with the corresponding SVM.

The principal disadvantage of the RVM compared to the SVM is that training involves optimizing a nonconvex function, and training times can be longer than for a comparable SVM. For a model with M basis functions, the RVM requires inversion of a matrix of size $M \times M$, which in general requires $O(M^3)$ computation. In the specific case of the SVM-like model (7.78), we have $M = N + 1$. As we have noted, there are techniques for training SVMs whose cost is roughly quadratic in N . Of course, in the case of the RVM we always have the option of starting with a smaller number of basis functions than $N + 1$. More significantly, in the relevance vector machine the parameters governing complexity and noise variance are determined automatically from a single training run, whereas in the support vector machine the parameters C and ϵ (or ν) are generally found using cross-validation, which involves multiple training runs. Furthermore, in the next section we shall derive an alternative procedure for training the relevance vector machine that improves training speed significantly.

7.2.2 Analysis of sparsity

We have noted earlier that the mechanism of *automatic relevance determination* causes a subset of parameters to be driven to zero. We now examine in more detail

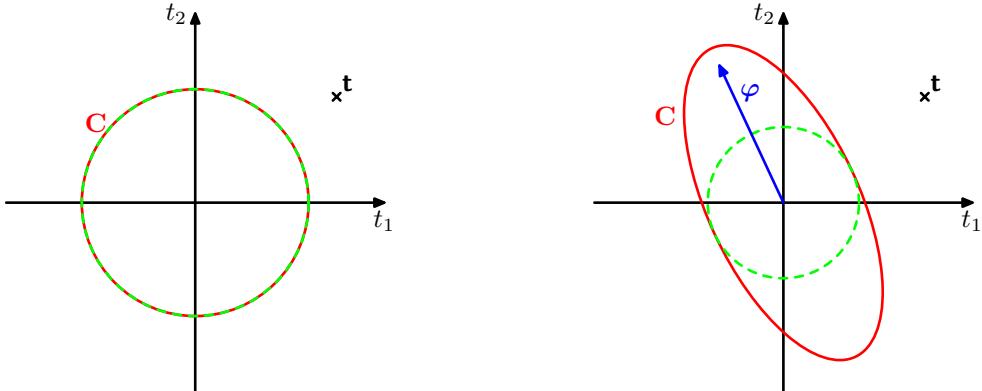


Figure 7.10 Illustration of the mechanism for sparsity in a Bayesian linear regression model, showing a training set vector of target values given by $\mathbf{t} = (t_1, t_2)^T$, indicated by the cross, for a model with one basis vector $\varphi = (\phi(\mathbf{x}_1), \phi(\mathbf{x}_2))^T$, which is poorly aligned with the target data vector \mathbf{t} . On the left we see a model having only isotropic noise, so that $\mathbf{C} = \beta^{-1}\mathbf{I}$, corresponding to $\alpha = \infty$, with β set to its most probable value. On the right we see the same model but with a finite value of α . In each case the red ellipse corresponds to unit Mahalanobis distance, with $|\mathbf{C}|$ taking the same value for both plots, while the dashed green circle shows the contribution arising from the noise term β^{-1} . We see that any finite value of α reduces the probability of the observed data, and so for the most probable solution the basis vector is removed.

the mechanism of sparsity in the context of the relevance vector machine. In the process, we will arrive at a significantly faster procedure for optimizing the hyperparameters compared to the direct techniques given above.

Before proceeding with a mathematical analysis, we first give some informal insight into the origin of sparsity in Bayesian linear models. Consider a data set comprising $N = 2$ observations t_1 and t_2 , together with a model having a single basis function $\phi(\mathbf{x})$, with hyperparameter α , along with isotropic noise having precision β . From (7.85), the marginal likelihood is given by $p(\mathbf{t}|\alpha, \beta) = \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C})$ in which the covariance matrix takes the form

$$\mathbf{C} = \frac{1}{\beta}\mathbf{I} + \frac{1}{\alpha}\varphi\varphi^T \quad (7.92)$$

where φ denotes the N -dimensional vector $(\phi(\mathbf{x}_1), \phi(\mathbf{x}_2))^T$, and similarly $\mathbf{t} = (t_1, t_2)^T$. Notice that this is just a zero-mean Gaussian process model over \mathbf{t} with covariance \mathbf{C} . Given a particular observation for \mathbf{t} , our goal is to find α^* and β^* by maximizing the marginal likelihood. We see from Figure 7.10 that, if there is a poor alignment between the direction of φ and that of the training data vector \mathbf{t} , then the corresponding hyperparameter α will be driven to ∞ , and the basis vector will be pruned from the model. This arises because any finite value for α will always assign a lower probability to the data, thereby decreasing the value of the density at \mathbf{t} , provided that β is set to its optimal value. We see that any finite value for α would cause the distribution to be elongated in a direction away from the data, thereby increasing the probability mass in regions away from the observed data and hence reducing the value of the density at the target data vector itself. For the more general case of M

basis vectors $\varphi_1, \dots, \varphi_M$ a similar intuition holds, namely that if a particular basis vector is poorly aligned with the data vector \mathbf{t} , then it is likely to be pruned from the model.

We now investigate the mechanism for sparsity from a more mathematical perspective, for a general case involving M basis functions. To motivate this analysis we first note that, in the result (7.87) for re-estimating the parameter α_i , the terms on the right-hand side are themselves also functions of α_i . These results therefore represent implicit solutions, and iteration would be required even to determine a single α_i with all other α_j for $j \neq i$ fixed.

This suggests a different approach to solving the optimization problem for the RVM, in which we make explicit all of the dependence of the marginal likelihood (7.85) on a particular α_i and then determine its stationary points explicitly (Faul and Tipping, 2002; Tipping and Faul, 2003). To do this, we first pull out the contribution from α_i in the matrix \mathbf{C} defined by (7.86) to give

$$\begin{aligned}\mathbf{C} &= \beta^{-1}\mathbf{I} + \sum_{j \neq i} \alpha_j^{-1} \varphi_j \varphi_j^T + \alpha_i^{-1} \varphi_i \varphi_i^T \\ &= \mathbf{C}_{-i} + \alpha_i^{-1} \varphi_i \varphi_i^T\end{aligned}\quad (7.93)$$

where φ_i denotes the i^{th} column of Φ , in other words the N -dimensional vector with elements $(\phi_i(\mathbf{x}_1), \dots, \phi_i(\mathbf{x}_N))$, in contrast to ϕ_n , which denotes the n^{th} row of Φ . The matrix \mathbf{C}_{-i} represents the matrix \mathbf{C} with the contribution from basis function i removed. Using the matrix identities (C.7) and (C.15), the determinant and inverse of \mathbf{C} can then be written

$$|\mathbf{C}| = |\mathbf{C}_{-i}| |1 + \alpha_i^{-1} \varphi_i^T \mathbf{C}_{-i}^{-1} \varphi_i| \quad (7.94)$$

$$\mathbf{C}^{-1} = \mathbf{C}_{-i}^{-1} - \frac{\mathbf{C}_{-i}^{-1} \varphi_i \varphi_i^T \mathbf{C}_{-i}^{-1}}{\alpha_i + \varphi_i^T \mathbf{C}_{-i}^{-1} \varphi_i}. \quad (7.95)$$

Using these results, we can then write the log marginal likelihood function (7.85) in the form

$$L(\boldsymbol{\alpha}) = L(\boldsymbol{\alpha}_{-i}) + \lambda(\alpha_i) \quad (7.96)$$

where $L(\boldsymbol{\alpha}_{-i})$ is simply the log marginal likelihood with basis function φ_i omitted, and the quantity $\lambda(\alpha_i)$ is defined by

$$\lambda(\alpha_i) = \frac{1}{2} \left[\ln \alpha_i - \ln (\alpha_i + s_i) + \frac{q_i^2}{\alpha_i + s_i} \right] \quad (7.97)$$

and contains all of the dependence on α_i . Here we have introduced the two quantities

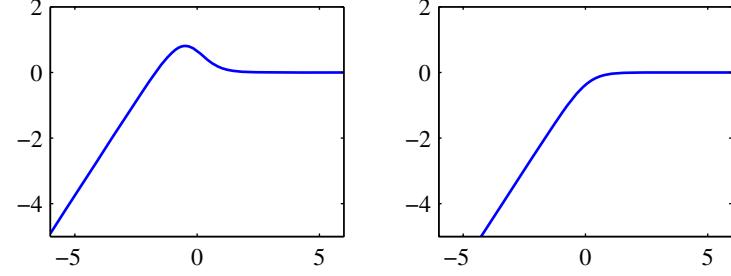
$$s_i = \varphi_i^T \mathbf{C}_{-i}^{-1} \varphi_i \quad (7.98)$$

$$q_i = \varphi_i^T \mathbf{C}_{-i}^{-1} \mathbf{t}. \quad (7.99)$$

Here s_i is called the *sparsity* and q_i is known as the *quality* of φ_i , and as we shall see, a large value of s_i relative to the value of q_i means that the basis function φ_i

Exercise 7.15

Figure 7.11 Plots of the log marginal likelihood $\lambda(\alpha_i)$ versus $\ln \alpha_i$ showing on the left, the single maximum at a finite α_i for $q_i^2 = 4$ and $s_i = 1$ (so that $q_i^2 > s_i$) and on the right, the maximum at $\alpha_i = \infty$ for $q_i^2 = 1$ and $s_i = 2$ (so that $q_i^2 < s_i$).



is more likely to be pruned from the model. The ‘sparsity’ measures the extent to which basis function φ_i overlaps with the other basis vectors in the model, and the ‘quality’ represents a measure of the alignment of the basis vector φ_n with the error between the training set values $\mathbf{t} = (t_1, \dots, t_N)^T$ and the vector \mathbf{y}_{-i} of predictions that would result from the model with the vector φ_i excluded (Tipping and Faul, 2003).

The stationary points of the marginal likelihood with respect to α_i occur when the derivative

$$\frac{d\lambda(\alpha_i)}{d\alpha_i} = \frac{\alpha_i^{-1}s_i^2 - (q_i^2 - s_i)}{2(\alpha_i + s_i)^2} \quad (7.100)$$

is equal to zero. There are two possible forms for the solution. Recalling that $\alpha_i \geq 0$, we see that if $q_i^2 < s_i$, then $\alpha_i \rightarrow \infty$ provides a solution. Conversely, if $q_i^2 > s_i$, we can solve for α_i to obtain

$$\alpha_i = \frac{s_i^2}{q_i^2 - s_i}. \quad (7.101)$$

These two solutions are illustrated in Figure 7.11. We see that the relative size of the quality and sparsity terms determines whether a particular basis vector will be pruned from the model or not. A more complete analysis (Faul and Tipping, 2002), based on the second derivatives of the marginal likelihood, confirms these solutions are indeed the unique maxima of $\lambda(\alpha_i)$.

Note that this approach has yielded a closed-form solution for α_i , for given values of the other hyperparameters. As well as providing insight into the origin of sparsity in the RVM, this analysis also leads to a practical algorithm for optimizing the hyperparameters that has significant speed advantages. This uses a fixed set of candidate basis vectors, and then cycles through them in turn to decide whether each vector should be included in the model or not. The resulting sequential sparse Bayesian learning algorithm is described below.

Sequential Sparse Bayesian Learning Algorithm

1. If solving a regression problem, initialize β .
2. Initialize using one basis function φ_1 , with hyperparameter α_1 set using (7.101), with the remaining hyperparameters α_j for $j \neq i$ initialized to infinity, so that only φ_1 is included in the model.

Exercise 7.16

3. Evaluate Σ and \mathbf{m} , along with q_i and s_i for all basis functions.
4. Select a candidate basis function φ_i .
5. If $q_i^2 > s_i$, and $\alpha_i < \infty$, so that the basis vector φ_i is already included in the model, then update α_i using (7.101).
6. If $q_i^2 > s_i$, and $\alpha_i = \infty$, then add φ_i to the model, and evaluate hyperparameter α_i using (7.101).
7. If $q_i^2 \leq s_i$, and $\alpha_i < \infty$ then remove basis function φ_i from the model, and set $\alpha_i = \infty$.
8. If solving a regression problem, update β .
9. If converged terminate, otherwise go to 3.

Note that if $q_i^2 \leq s_i$ and $\alpha_i = \infty$, then the basis function φ_i is already excluded from the model and no action is required.

In practice, it is convenient to evaluate the quantities

$$Q_i = \varphi_i^T \mathbf{C}^{-1} \mathbf{t} \quad (7.102)$$

$$S_i = \varphi_i^T \mathbf{C}^{-1} \varphi_i. \quad (7.103)$$

The quality and sparseness variables can then be expressed in the form

$$q_i = \frac{\alpha_i Q_i}{\alpha_i - S_i} \quad (7.104)$$

$$s_i = \frac{\alpha_i S_i}{\alpha_i - S_i}. \quad (7.105)$$

Exercise 7.17

Note that when $\alpha_i = \infty$, we have $q_i = Q_i$ and $s_i = S_i$. Using (C.7), we can write

$$Q_i = \beta \varphi_i^T \mathbf{t} - \beta^2 \varphi_i^T \Phi \Sigma \Phi^T \mathbf{t} \quad (7.106)$$

$$S_i = \beta \varphi_i^T \varphi_i - \beta^2 \varphi_i^T \Phi \Sigma \Phi^T \varphi_i \quad (7.107)$$

where Φ and Σ involve only those basis vectors that correspond to finite hyperparameters α_i . At each stage the required computations therefore scale like $O(M^3)$, where M is the number of active basis vectors in the model and is typically much smaller than the number N of training patterns.

7.2.3 RVM for classification

We can extend the relevance vector machine framework to classification problems by applying the ARD prior over weights to a probabilistic linear classification model of the kind studied in Chapter 4. To start with, we consider two-class problems with a binary target variable $t \in \{0, 1\}$. The model now takes the form of a linear combination of basis functions transformed by a logistic sigmoid function

$$y(\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T \phi(\mathbf{x})) \quad (7.108)$$

where $\sigma(\cdot)$ is the logistic sigmoid function defined by (4.59). If we introduce a Gaussian prior over the weight vector \mathbf{w} , then we obtain the model that has been considered already in Chapter 4. The difference here is that in the RVM, this model uses the ARD prior (7.80) in which there is a separate precision hyperparameter associated with each weight parameter.

In contrast to the regression model, we can no longer integrate analytically over the parameter vector \mathbf{w} . Here we follow Tipping (2001) and use the Laplace approximation, which was applied to the closely related problem of Bayesian logistic regression in Section 4.5.1.

We begin by initializing the hyperparameter vector α . For this given value of α , we then build a Gaussian approximation to the posterior distribution and thereby obtain an approximation to the marginal likelihood. Maximization of this approximate marginal likelihood then leads to a re-estimated value for α , and the process is repeated until convergence.

Let us consider the Laplace approximation for this model in more detail. For a fixed value of α , the mode of the posterior distribution over \mathbf{w} is obtained by maximizing

$$\begin{aligned} \ln p(\mathbf{w}|\mathbf{t}, \alpha) &= \ln \{p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\alpha)\} - \ln p(\mathbf{t}|\alpha) \\ &= \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} - \frac{1}{2} \mathbf{w}^T \mathbf{A} \mathbf{w} + \text{const} \end{aligned} \quad (7.109)$$

where $\mathbf{A} = \text{diag}(\alpha_i)$. This can be done using iterative reweighted least squares (IRLS) as discussed in Section 4.3.3. For this, we need the gradient vector and Hessian matrix of the log posterior distribution, which from (7.109) are given by

$$\nabla \ln p(\mathbf{w}|\mathbf{t}, \alpha) = \Phi^T(\mathbf{t} - \mathbf{y}) - \mathbf{A} \mathbf{w} \quad (7.110)$$

$$\nabla \nabla \ln p(\mathbf{w}|\mathbf{t}, \alpha) = -(\Phi^T \mathbf{B} \Phi + \mathbf{A}) \quad (7.111)$$

where \mathbf{B} is an $N \times N$ diagonal matrix with elements $b_n = y_n(1 - y_n)$, the vector $\mathbf{y} = (y_1, \dots, y_N)^T$, and Φ is the design matrix with elements $\Phi_{ni} = \phi_i(\mathbf{x}_n)$. Here we have used the property (4.88) for the derivative of the logistic sigmoid function. At convergence of the IRLS algorithm, the negative Hessian represents the inverse covariance matrix for the Gaussian approximation to the posterior distribution.

The mode of the resulting approximation to the posterior distribution, corresponding to the mean of the Gaussian approximation, is obtained setting (7.110) to zero, giving the mean and covariance of the Laplace approximation in the form

$$\mathbf{w}^* = \mathbf{A}^{-1} \Phi^T (\mathbf{t} - \mathbf{y}) \quad (7.112)$$

$$\Sigma = (\Phi^T \mathbf{B} \Phi + \mathbf{A})^{-1}. \quad (7.113)$$

Section 4.4

Exercise 7.18

We can now use this Laplace approximation to evaluate the marginal likelihood. Using the general result (4.135) for an integral evaluated using the Laplace approxi-

mation, we have

$$\begin{aligned} p(\mathbf{t}|\boldsymbol{\alpha}) &= \int p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha}) d\mathbf{w} \\ &\simeq p(\mathbf{t}|\mathbf{w}^*)p(\mathbf{w}^*|\boldsymbol{\alpha})(2\pi)^{M/2}|\boldsymbol{\Sigma}|^{1/2}. \end{aligned} \quad (7.114)$$

If we substitute for $p(\mathbf{t}|\mathbf{w}^*)$ and $p(\mathbf{w}^*|\boldsymbol{\alpha})$ and then set the derivative of the marginal likelihood with respect to α_i equal to zero, we obtain

$$-\frac{1}{2}(w_i^*)^2 + \frac{1}{2\alpha_i} - \frac{1}{2}\Sigma_{ii} = 0. \quad (7.115)$$

Defining $\gamma_i = 1 - \alpha_i\Sigma_{ii}$ and rearranging then gives

$$\alpha_i^{\text{new}} = \frac{\gamma_i}{(w_i^*)^2} \quad (7.116)$$

which is identical to the re-estimation formula (7.87) obtained for the regression RVM.

If we define

$$\hat{\mathbf{t}} = \Phi\mathbf{w}^* + \mathbf{B}^{-1}(\mathbf{t} - \mathbf{y}) \quad (7.117)$$

we can write the approximate log marginal likelihood in the form

$$\ln p(\mathbf{t}|\boldsymbol{\alpha}, \beta) = -\frac{1}{2} \left\{ N \ln(2\pi) + \ln |\mathbf{C}| + (\hat{\mathbf{t}})^T \mathbf{C}^{-1} \hat{\mathbf{t}} \right\} \quad (7.118)$$

where

$$\mathbf{C} = \mathbf{B} + \Phi\mathbf{A}\Phi^T. \quad (7.119)$$

This takes the same form as (7.85) in the regression case, and so we can apply the same analysis of sparsity and obtain the same fast learning algorithm in which we fully optimize a single hyperparameter α_i at each step.

Figure 7.12 shows the relevance vector machine applied to a synthetic classification data set. We see that the relevance vectors tend not to lie in the region of the decision boundary, in contrast to the support vector machine. This is consistent with our earlier discussion of sparsity in the RVM, because a basis function $\phi_i(\mathbf{x})$ centred on a data point near the boundary will have a vector φ_i that is poorly aligned with the training data vector \mathbf{t} .

One of the potential advantages of the relevance vector machine compared with the SVM is that it makes probabilistic predictions. For example, this allows the RVM to be used to help construct an emission density in a nonlinear extension of the linear dynamical system for tracking faces in video sequences (Williams *et al.*, 2005).

So far, we have considered the RVM for binary classification problems. For $K > 2$ classes, we again make use of the probabilistic approach in Section 4.3.4 in which there are K linear models of the form

$$a_k = \mathbf{w}_k^T \mathbf{x} \quad (7.120)$$

Exercise 7.19

Appendix A

Section 13.3

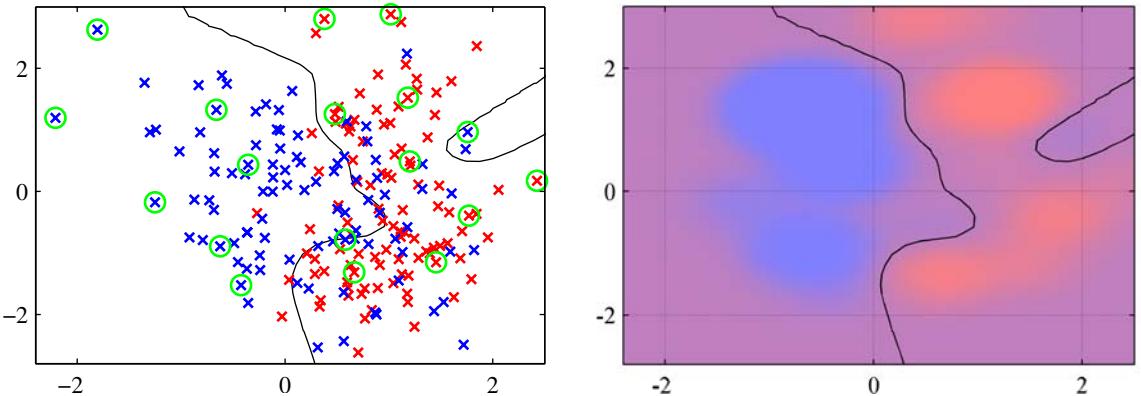


Figure 7.12 Example of the relevance vector machine applied to a synthetic data set, in which the left-hand plot shows the decision boundary and the data points, with the relevance vectors indicated by circles. Comparison with the results shown in Figure 7.4 for the corresponding support vector machine shows that the RVM gives a much sparser model. The right-hand plot shows the posterior probability given by the RVM output in which the proportion of red (blue) ink indicates the probability of that point belonging to the red (blue) class.

which are combined using a softmax function to give outputs

$$y_k(\mathbf{x}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}. \quad (7.121)$$

The log likelihood function is then given by

$$\ln p(\mathbf{T}|\mathbf{w}_1, \dots, \mathbf{w}_K) = \prod_{n=1}^N \prod_{k=1}^K y_{nk}^{t_{nk}} \quad (7.122)$$

where the target values t_{nk} have a 1-of- K coding for each data point n , and \mathbf{T} is a matrix with elements t_{nk} . Again, the Laplace approximation can be used to optimize the hyperparameters (Tipping, 2001), in which the model and its Hessian are found using IRLS. This gives a more principled approach to multiclass classification than the pairwise method used in the support vector machine and also provides probabilistic predictions for new data points. The principal disadvantage is that the Hessian matrix has size $MK \times MK$, where M is the number of active basis functions, which gives an additional factor of K^3 in the computational cost of training compared with the two-class RVM.

The principal disadvantage of the relevance vector machine is the relatively long training times compared with the SVM. This is offset, however, by the avoidance of cross-validation runs to set the model complexity parameters. Furthermore, because it yields sparser models, the computation time on test points, which is usually the more important consideration in practice, is typically much less.

Exercises

- 7.1** (**) **www** Suppose we have a data set of input vectors $\{\mathbf{x}_n\}$ with corresponding target values $t_n \in \{-1, 1\}$, and suppose that we model the density of input vectors within each class separately using a Parzen kernel density estimator (see Section 2.5.1) with a kernel $k(\mathbf{x}, \mathbf{x}')$. Write down the minimum misclassification-rate decision rule assuming the two classes have equal prior probability. Show also that, if the kernel is chosen to be $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$, then the classification rule reduces to simply assigning a new input vector to the class having the closest mean. Finally, show that, if the kernel takes the form $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$, that the classification is based on the closest mean in the feature space $\phi(\mathbf{x})$.
- 7.2** (*) Show that, if the 1 on the right-hand side of the constraint (7.5) is replaced by some arbitrary constant $\gamma > 0$, the solution for the maximum margin hyperplane is unchanged.
- 7.3** (**) Show that, irrespective of the dimensionality of the data space, a data set consisting of just two data points, one from each class, is sufficient to determine the location of the maximum-margin hyperplane.
- 7.4** (**) **www** Show that the value ρ of the margin for the maximum-margin hyperplane is given by

$$\frac{1}{\rho^2} = \sum_{n=1}^N a_n \quad (7.123)$$

where $\{a_n\}$ are given by maximizing (7.10) subject to the constraints (7.11) and (7.12).

- 7.5** (**) Show that the values of ρ and $\{a_n\}$ in the previous exercise also satisfy

$$\frac{1}{\rho^2} = 2\tilde{L}(\mathbf{a}) \quad (7.124)$$

where $\tilde{L}(\mathbf{a})$ is defined by (7.10). Similarly, show that

$$\frac{1}{\rho^2} = \|\mathbf{w}\|^2. \quad (7.125)$$

- 7.6** (*) Consider the logistic regression model with a target variable $t \in \{-1, 1\}$. If we define $p(t = 1|y) = \sigma(y)$ where $y(\mathbf{x})$ is given by (7.1), show that the negative log likelihood, with the addition of a quadratic regularization term, takes the form (7.47).
- 7.7** (*) Consider the Lagrangian (7.56) for the regression support vector machine. By setting the derivatives of the Lagrangian with respect to \mathbf{w} , b , ξ_n , and $\hat{\xi}_n$ to zero and then back substituting to eliminate the corresponding variables, show that the dual Lagrangian is given by (7.61).

7.8 (*) **www** For the regression support vector machine considered in Section 7.1.4, show that all training data points for which $\xi_n > 0$ will have $a_n = C$, and similarly all points for which $\widehat{\xi}_n > 0$ will have $\widehat{a}_n = C$.

7.9 (*) Verify the results (7.82) and (7.83) for the mean and covariance of the posterior distribution over weights in the regression RVM.

7.10 (**) **www** Derive the result (7.85) for the marginal likelihood function in the regression RVM, by performing the Gaussian integral over w in (7.84) using the technique of completing the square in the exponential.

7.11 (**) Repeat the above exercise, but this time make use of the general result (2.115).

7.12 (**) **www** Show that direct maximization of the log marginal likelihood (7.85) for the regression relevance vector machine leads to the re-estimation equations (7.87) and (7.88) where γ_i is defined by (7.89).

7.13 (**) In the evidence framework for RVM regression, we obtained the re-estimation formulae (7.87) and (7.88) by maximizing the marginal likelihood given by (7.85). Extend this approach by inclusion of hyperpriors given by gamma distributions of the form (B.26) and obtain the corresponding re-estimation formulae for α and β by maximizing the corresponding posterior probability $p(\mathbf{t}, \alpha, \beta | \mathbf{X})$ with respect to α and β .

7.14 (**) Derive the result (7.90) for the predictive distribution in the relevance vector machine for regression. Show that the predictive variance is given by (7.91).

7.15 (**) **www** Using the results (7.94) and (7.95), show that the marginal likelihood (7.85) can be written in the form (7.96), where $\lambda(\alpha_n)$ is defined by (7.97) and the sparsity and quality factors are defined by (7.98) and (7.99), respectively.

7.16 (*) By taking the second derivative of the log marginal likelihood (7.97) for the regression RVM with respect to the hyperparameter α_i , show that the stationary point given by (7.101) is a maximum of the marginal likelihood.

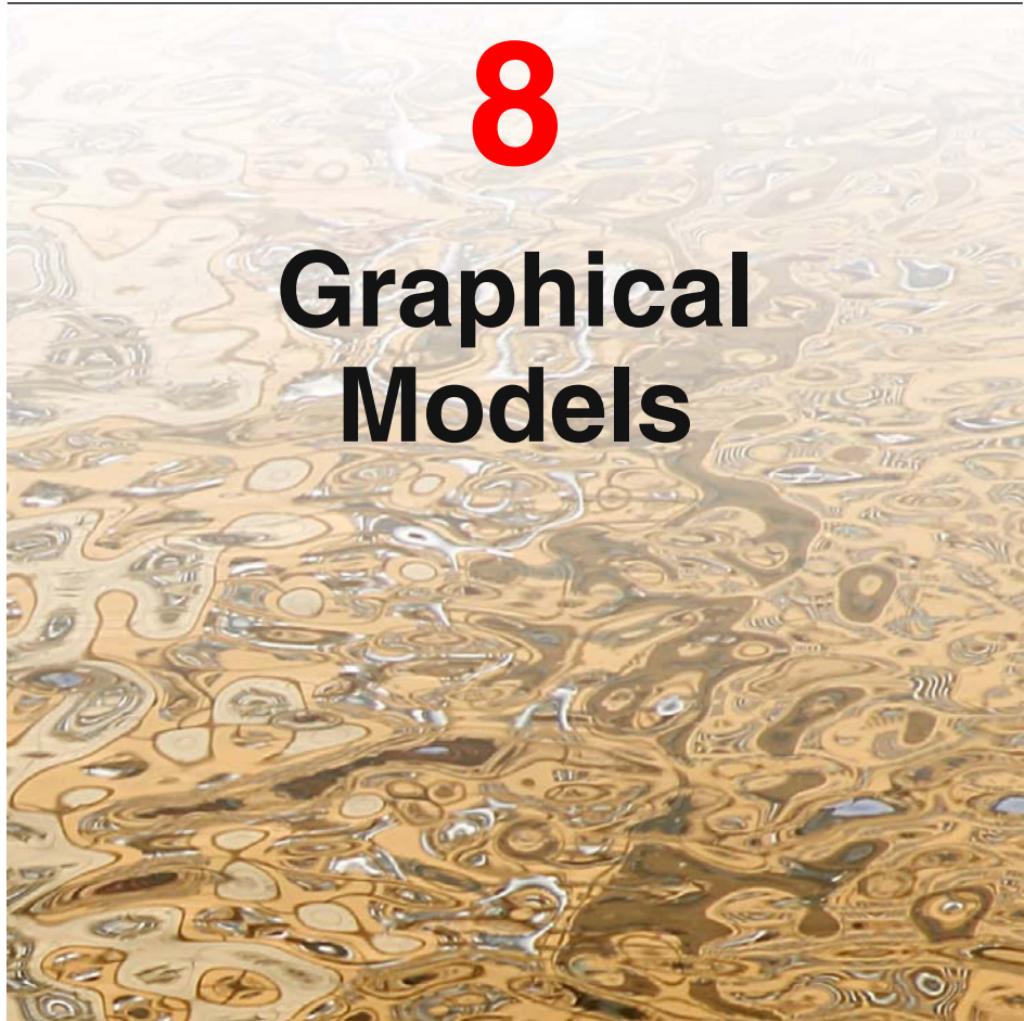
7.17 (**) Using (7.83) and (7.86), together with the matrix identity (C.7), show that the quantities S_n and Q_n defined by (7.102) and (7.103) can be written in the form (7.106) and (7.107).

7.18 (*) **www** Show that the gradient vector and Hessian matrix of the log posterior distribution (7.109) for the classification relevance vector machine are given by (7.110) and (7.111).

7.19 (**) Verify that maximization of the approximate log marginal likelihood function (7.114) for the classification relevance vector machine leads to the result (7.116) for re-estimation of the hyperparameters.

8

Graphical Models



Probabilities play a central role in modern pattern recognition. We have seen in Chapter 1 that probability theory can be expressed in terms of two simple equations corresponding to the sum rule and the product rule. All of the probabilistic inference and learning manipulations discussed in this book, no matter how complex, amount to repeated application of these two equations. We could therefore proceed to formulate and solve complicated probabilistic models purely by algebraic manipulation. However, we shall find it highly advantageous to augment the analysis using diagrammatic representations of probability distributions, called *probabilistic graphical models*. These offer several useful properties:

1. They provide a simple way to visualize the structure of a probabilistic model and can be used to design and motivate new models.
2. Insights into the properties of the model, including conditional independence properties, can be obtained by inspection of the graph.

3. Complex computations, required to perform inference and learning in sophisticated models, can be expressed in terms of graphical manipulations, in which underlying mathematical expressions are carried along implicitly.

A graph comprises *nodes* (also called *vertices*) connected by *links* (also known as *edges* or *arcs*). In a probabilistic graphical model, each node represents a random variable (or group of random variables), and the links express probabilistic relationships between these variables. The graph then captures the way in which the joint distribution over all of the random variables can be decomposed into a product of factors each depending only on a subset of the variables. We shall begin by discussing *Bayesian networks*, also known as *directed graphical models*, in which the links of the graphs have a particular directionality indicated by arrows. The other major class of graphical models are *Markov random fields*, also known as *undirected graphical models*, in which the links do not carry arrows and have no directional significance. Directed graphs are useful for expressing causal relationships between random variables, whereas undirected graphs are better suited to expressing soft constraints between random variables. For the purposes of solving inference problems, it is often convenient to convert both directed and undirected graphs into a different representation called a *factor graph*.

In this chapter, we shall focus on the key aspects of graphical models as needed for applications in pattern recognition and machine learning. More general treatments of graphical models can be found in the books by Whittaker (1990), Lauritzen (1996), Jensen (1996), Castillo *et al.* (1997), Jordan (1999), Cowell *et al.* (1999), and Jordan (2007).

8.1. Bayesian Networks

In order to motivate the use of directed graphs to describe probability distributions, consider first an arbitrary joint distribution $p(a, b, c)$ over three variables a , b , and c . Note that at this stage, we do not need to specify anything further about these variables, such as whether they are discrete or continuous. Indeed, one of the powerful aspects of graphical models is that a specific graph can make probabilistic statements for a broad class of distributions. By application of the product rule of probability (1.11), we can write the joint distribution in the form

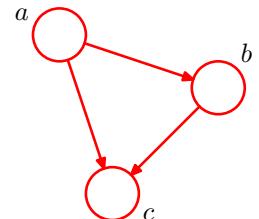
$$p(a, b, c) = p(c|a, b)p(a, b). \quad (8.1)$$

A second application of the product rule, this time to the second term on the right-hand side of (8.1), gives

$$p(a, b, c) = p(c|a, b)p(b|a)p(a). \quad (8.2)$$

Note that this decomposition holds for any choice of the joint distribution. We now represent the right-hand side of (8.2) in terms of a simple graphical model as follows. First we introduce a node for each of the random variables a , b , and c and associate each node with the corresponding conditional distribution on the right-hand side of

Figure 8.1 A directed graphical model representing the joint probability distribution over three variables a , b , and c , corresponding to the decomposition on the right-hand side of (8.2).



(8.2). Then, for each conditional distribution we add directed links (arrows) to the graph from the nodes corresponding to the variables on which the distribution is conditioned. Thus for the factor $p(c|a, b)$, there will be links from nodes a and b to node c , whereas for the factor $p(a)$ there will be no incoming links. The result is the graph shown in Figure 8.1. If there is a link going from a node a to a node b , then we say that node a is the *parent* of node b , and we say that node b is the *child* of node a . Note that we shall not make any formal distinction between a node and the variable to which it corresponds but will simply use the same symbol to refer to both.

An interesting point to note about (8.2) is that the left-hand side is symmetrical with respect to the three variables a , b , and c , whereas the right-hand side is not. Indeed, in making the decomposition in (8.2), we have implicitly chosen a particular ordering, namely a, b, c , and had we chosen a different ordering we would have obtained a different decomposition and hence a different graphical representation. We shall return to this point later.

For the moment let us extend the example of Figure 8.1 by considering the joint distribution over K variables given by $p(x_1, \dots, x_K)$. By repeated application of the product rule of probability, this joint distribution can be written as a product of conditional distributions, one for each of the variables

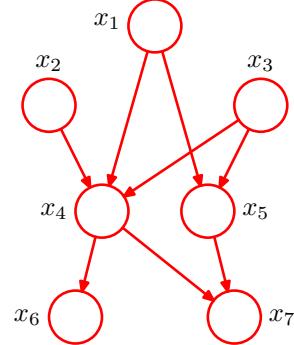
$$p(x_1, \dots, x_K) = p(x_K|x_1, \dots, x_{K-1}) \dots p(x_2|x_1)p(x_1). \quad (8.3)$$

For a given choice of K , we can again represent this as a directed graph having K nodes, one for each conditional distribution on the right-hand side of (8.3), with each node having incoming links from all lower numbered nodes. We say that this graph is *fully connected* because there is a link between every pair of nodes.

So far, we have worked with completely general joint distributions, so that the decompositions, and their representations as fully connected graphs, will be applicable to any choice of distribution. As we shall see shortly, it is the *absence* of links in the graph that conveys interesting information about the properties of the class of distributions that the graph represents. Consider the graph shown in Figure 8.2. This is not a fully connected graph because, for instance, there is no link from x_1 to x_2 or from x_3 to x_7 .

We shall now go from this graph to the corresponding representation of the joint probability distribution written in terms of the product of a set of conditional distributions, one for each node in the graph. Each such conditional distribution will be conditioned only on the parents of the corresponding node in the graph. For instance, x_5 will be conditioned on x_1 and x_3 . The joint distribution of all 7 variables

Figure 8.2 Example of a directed acyclic graph describing the joint distribution over variables x_1, \dots, x_7 . The corresponding decomposition of the joint distribution is given by (8.4).



is therefore given by

$$p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3)p(x_5|x_1, x_3)p(x_6|x_4)p(x_7|x_4, x_5). \quad (8.4)$$

The reader should take a moment to study carefully the correspondence between (8.4) and Figure 8.2.

We can now state in general terms the relationship between a given directed graph and the corresponding distribution over the variables. The joint distribution defined by a graph is given by the product, over all of the nodes of the graph, of a conditional distribution for each node conditioned on the variables corresponding to the parents of that node in the graph. Thus, for a graph with K nodes, the joint distribution is given by

$$p(\mathbf{x}) = \prod_{k=1}^K p(x_k|\text{pa}_k) \quad (8.5)$$

where pa_k denotes the set of parents of x_k , and $\mathbf{x} = \{x_1, \dots, x_K\}$. This key equation expresses the *factorization* properties of the joint distribution for a directed graphical model. Although we have considered each node to correspond to a single variable, we can equally well associate sets of variables and vector-valued variables with the nodes of a graph. It is easy to show that the representation on the right-hand side of (8.5) is always correctly normalized provided the individual conditional distributions are normalized.

The directed graphs that we are considering are subject to an important restriction namely that there must be no *directed cycles*, in other words there are no closed paths within the graph such that we can move from node to node along links following the direction of the arrows and end up back at the starting node. Such graphs are also called *directed acyclic graphs*, or *DAGs*. This is equivalent to the statement that there exists an ordering of the nodes such that there are no links that go from any node to any lower numbered node.

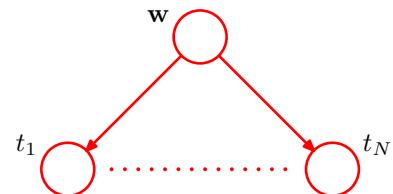
8.1.1 Example: Polynomial regression

As an illustration of the use of directed graphs to describe probability distributions, we consider the Bayesian polynomial regression model introduced in Sec-

Exercise 8.1

Exercise 8.2

Figure 8.3 Directed graphical model representing the joint distribution (8.6) corresponding to the Bayesian polynomial regression model introduced in Section 1.2.6.



tion 1.2.6. The random variables in this model are the vector of polynomial coefficients \mathbf{w} and the observed data $\mathbf{t} = (t_1, \dots, t_N)^T$. In addition, this model contains the input data $\mathbf{x} = (x_1, \dots, x_N)^T$, the noise variance σ^2 , and the hyperparameter α representing the precision of the Gaussian prior over \mathbf{w} , all of which are parameters of the model rather than random variables. Focussing just on the random variables for the moment, we see that the joint distribution is given by the product of the prior $p(\mathbf{w})$ and N conditional distributions $p(t_n|\mathbf{w})$ for $n = 1, \dots, N$ so that

$$p(\mathbf{t}, \mathbf{w}) = p(\mathbf{w}) \prod_{n=1}^N p(t_n|\mathbf{w}). \quad (8.6)$$

This joint distribution can be represented by a graphical model shown in Figure 8.3.

When we start to deal with more complex models later in the book, we shall find it inconvenient to have to write out multiple nodes of the form t_1, \dots, t_N explicitly as in Figure 8.3. We therefore introduce a graphical notation that allows such multiple nodes to be expressed more compactly, in which we draw a single representative node t_n and then surround this with a box, called a *plate*, labelled with N indicating that there are N nodes of this kind. Re-writing the graph of Figure 8.3 in this way, we obtain the graph shown in Figure 8.4.

We shall sometimes find it helpful to make the parameters of a model, as well as its stochastic variables, explicit. In this case, (8.6) becomes

$$p(\mathbf{t}, \mathbf{w} | \mathbf{x}, \alpha, \sigma^2) = p(\mathbf{w} | \alpha) \prod_{n=1}^N p(t_n | \mathbf{w}, x_n, \sigma^2).$$

Correspondingly, we can make \mathbf{x} and α explicit in the graphical representation. To do this, we shall adopt the convention that random variables will be denoted by open circles, and deterministic parameters will be denoted by smaller solid circles. If we take the graph of Figure 8.4 and include the deterministic parameters, we obtain the graph shown in Figure 8.5.

When we apply a graphical model to a problem in machine learning or pattern recognition, we will typically set some of the random variables to specific observed

Figure 8.4 An alternative, more compact, representation of the graph shown in Figure 8.3 in which we have introduced a *plate* (the box labelled N) that represents N nodes of which only a single example t_n is shown explicitly.

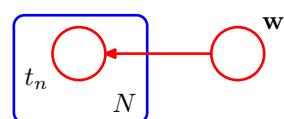
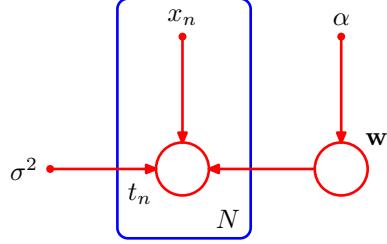


Figure 8.5 This shows the same model as in Figure 8.4 but with the deterministic parameters shown explicitly by the smaller solid nodes.



values, for example the variables $\{t_n\}$ from the training set in the case of polynomial curve fitting. In a graphical model, we will denote such *observed variables* by shading the corresponding nodes. Thus the graph corresponding to Figure 8.5 in which the variables $\{t_n\}$ are observed is shown in Figure 8.6. Note that the value of w is not observed, and so w is an example of a *latent variable*, also known as a *hidden variable*. Such variables play a crucial role in many probabilistic models and will form the focus of Chapters 9 and 12.

Having observed the values $\{t_n\}$ we can, if desired, evaluate the posterior distribution of the polynomial coefficients w as discussed in Section 1.2.5. For the moment, we note that this involves a straightforward application of Bayes' theorem

$$p(w|T) \propto p(w) \prod_{n=1}^N p(t_n|w) \quad (8.7)$$

where again we have omitted the deterministic parameters in order to keep the notation uncluttered.

In general, model parameters such as w are of little direct interest in themselves, because our ultimate goal is to make predictions for new input values. Suppose we are given a new input value \hat{x} and we wish to find the corresponding probability distribution for \hat{t} conditioned on the observed data. The graphical model that describes this problem is shown in Figure 8.7, and the corresponding joint distribution of all of the random variables in this model, conditioned on the deterministic parameters, is then given by

$$p(\hat{t}, t, w | \hat{x}, x, \alpha, \sigma^2) = \left[\prod_{n=1}^N p(t_n | x_n, w, \sigma^2) \right] p(w | \alpha) p(\hat{t} | \hat{x}, w, \sigma^2). \quad (8.8)$$

Figure 8.6 As in Figure 8.5 but with the nodes $\{t_n\}$ shaded to indicate that the corresponding random variables have been set to their observed (training set) values.

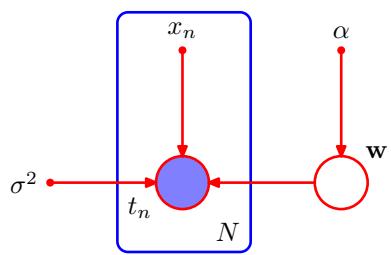
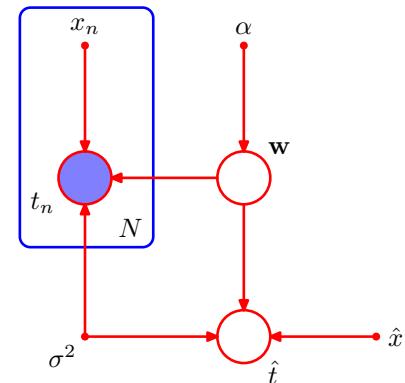


Figure 8.7 The polynomial regression model, corresponding to Figure 8.6, showing also a new input value \hat{x} together with the corresponding model prediction \hat{t} .



The required predictive distribution for \hat{t} is then obtained, from the sum rule of probability, by integrating out the model parameters w so that

$$p(\hat{t}|\hat{x}, \mathbf{x}, \mathbf{t}, \alpha, \sigma^2) \propto \int p(\hat{t}, \mathbf{t}, \mathbf{w}|\hat{x}, \mathbf{x}, \alpha, \sigma^2) d\mathbf{w}$$

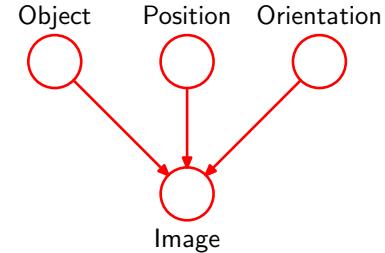
where we are implicitly setting the random variables in \mathbf{t} to the specific values observed in the data set. The details of this calculation were discussed in Chapter 3.

8.1.2 Generative models

There are many situations in which we wish to draw samples from a given probability distribution. Although we shall devote the whole of Chapter 11 to a detailed discussion of sampling methods, it is instructive to outline here one technique, called *ancestral sampling*, which is particularly relevant to graphical models. Consider a joint distribution $p(x_1, \dots, x_K)$ over K variables that factorizes according to (8.5) corresponding to a directed acyclic graph. We shall suppose that the variables have been ordered such that there are no links from any node to any lower numbered node, in other words each node has a higher number than any of its parents. Our goal is to draw a sample $\hat{x}_1, \dots, \hat{x}_K$ from the joint distribution.

To do this, we start with the lowest-numbered node and draw a sample from the distribution $p(x_1)$, which we call \hat{x}_1 . We then work through each of the nodes in order, so that for node n we draw a sample from the conditional distribution $p(x_n|pa_n)$ in which the parent variables have been set to their sampled values. Note that at each stage, these parent values will always be available because they correspond to lower-numbered nodes that have already been sampled. Techniques for sampling from specific distributions will be discussed in detail in Chapter 11. Once we have sampled from the final variable x_K , we will have achieved our objective of obtaining a sample from the joint distribution. To obtain a sample from some marginal distribution corresponding to a subset of the variables, we simply take the sampled values for the required nodes and ignore the sampled values for the remaining nodes. For example, to draw a sample from the distribution $p(x_2, x_4)$, we simply sample from the full joint distribution and then retain the values \hat{x}_2, \hat{x}_4 and discard the remaining values $\{\hat{x}_{j \neq 2,4}\}$.

Figure 8.8 A graphical model representing the process by which images of objects are created, in which the identity of an object (a discrete variable) and the position and orientation of that object (continuous variables) have independent prior probabilities. The image (a vector of pixel intensities) has a probability distribution that is dependent on the identity of the object as well as on its position and orientation.



For practical applications of probabilistic models, it will typically be the higher-numbered variables corresponding to terminal nodes of the graph that represent the observations, with lower-numbered nodes corresponding to latent variables. The primary role of the latent variables is to allow a complicated distribution over the observed variables to be represented in terms of a model constructed from simpler (typically exponential family) conditional distributions.

We can interpret such models as expressing the processes by which the observed data arose. For instance, consider an object recognition task in which each observed data point corresponds to an image (comprising a vector of pixel intensities) of one of the objects. In this case, the latent variables might have an interpretation as the position and orientation of the object. Given a particular observed image, our goal is to find the posterior distribution over objects, in which we integrate over all possible positions and orientations. We can represent this problem using a graphical model of the form shown in Figure 8.8.

The graphical model captures the *causal* process (Pearl, 1988) by which the observed data was generated. For this reason, such models are often called *generative* models. By contrast, the polynomial regression model described by Figure 8.5 is not generative because there is no probability distribution associated with the input variable x , and so it is not possible to generate synthetic data points from this model. We could make it generative by introducing a suitable prior distribution $p(x)$, at the expense of a more complex model.

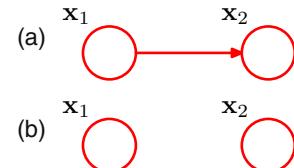
The hidden variables in a probabilistic model need not, however, have any explicit physical interpretation but may be introduced simply to allow a more complex joint distribution to be constructed from simpler components. In either case, the technique of ancestral sampling applied to a generative model mimics the creation of the observed data and would therefore give rise to ‘fantasy’ data whose probability distribution (if the model were a perfect representation of reality) would be the same as that of the observed data. In practice, producing synthetic observations from a generative model can prove informative in understanding the form of the probability distribution represented by that model.

8.1.3 Discrete variables

We have discussed the importance of probability distributions that are members of the exponential family, and we have seen that this family includes many well-known distributions as particular cases. Although such distributions are relatively simple, they form useful building blocks for constructing more complex probability

Figure 8.9

(a) This fully-connected graph describes a general distribution over two K -state discrete variables having a total of $K^2 - 1$ parameters. (b) By dropping the link between the nodes, the number of parameters is reduced to $2(K - 1)$.



distributions, and the framework of graphical models is very useful in expressing the way in which these building blocks are linked together.

Such models have particularly nice properties if we choose the relationship between each parent-child pair in a directed graph to be conjugate, and we shall explore several examples of this shortly. Two cases are particularly worthy of note, namely when the parent and child node each correspond to discrete variables and when they each correspond to Gaussian variables, because in these two cases the relationship can be extended hierarchically to construct arbitrarily complex directed acyclic graphs. We begin by examining the discrete case.

The probability distribution $p(\mathbf{x}|\boldsymbol{\mu})$ for a single discrete variable \mathbf{x} having K possible states (using the 1-of- K representation) is given by

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^K \mu_k^{x_k} \quad (8.9)$$

and is governed by the parameters $\boldsymbol{\mu} = (\mu_1, \dots, \mu_K)^T$. Due to the constraint $\sum_k \mu_k = 1$, only $K - 1$ values for μ_k need to be specified in order to define the distribution.

Now suppose that we have two discrete variables, \mathbf{x}_1 and \mathbf{x}_2 , each of which has K states, and we wish to model their joint distribution. We denote the probability of observing both $x_{1k} = 1$ and $x_{2l} = 1$ by the parameter μ_{kl} , where x_{1k} denotes the k^{th} component of \mathbf{x}_1 , and similarly for x_{2l} . The joint distribution can be written

$$p(\mathbf{x}_1, \mathbf{x}_2 | \boldsymbol{\mu}) = \prod_{k=1}^K \prod_{l=1}^K \mu_{kl}^{x_{1k} x_{2l}}.$$

Because the parameters μ_{kl} are subject to the constraint $\sum_k \sum_l \mu_{kl} = 1$, this distribution is governed by $K^2 - 1$ parameters. It is easily seen that the total number of parameters that must be specified for an arbitrary joint distribution over M variables is $K^M - 1$ and therefore grows exponentially with the number M of variables.

Using the product rule, we can factor the joint distribution $p(\mathbf{x}_1, \mathbf{x}_2)$ in the form $p(\mathbf{x}_2 | \mathbf{x}_1)p(\mathbf{x}_1)$, which corresponds to a two-node graph with a link going from the \mathbf{x}_1 node to the \mathbf{x}_2 node as shown in Figure 8.9(a). The marginal distribution $p(\mathbf{x}_1)$ is governed by $K - 1$ parameters, as before. Similarly, the conditional distribution $p(\mathbf{x}_2 | \mathbf{x}_1)$ requires the specification of $K - 1$ parameters for each of the K possible values of \mathbf{x}_1 . The total number of parameters that must be specified in the joint distribution is therefore $(K - 1) + K(K - 1) = K^2 - 1$ as before.

Now suppose that the variables \mathbf{x}_1 and \mathbf{x}_2 were independent, corresponding to the graphical model shown in Figure 8.9(b). Each variable is then described by

Figure 8.10 This chain of M discrete nodes, each having K states, requires the specification of $K - 1 + (M - 1)K(K - 1)$ parameters, which grows linearly with the length M of the chain. In contrast, a fully connected graph of M nodes would have $K^M - 1$ parameters, which grows exponentially with M .



a separate multinomial distribution, and the total number of parameters would be $2(K - 1)$. For a distribution over M independent discrete variables, each having K states, the total number of parameters would be $M(K - 1)$, which therefore grows linearly with the number of variables. From a graphical perspective, we have reduced the number of parameters by dropping links in the graph, at the expense of having a restricted class of distributions.

More generally, if we have M discrete variables $\mathbf{x}_1, \dots, \mathbf{x}_M$, we can model the joint distribution using a directed graph with one variable corresponding to each node. The conditional distribution at each node is given by a set of nonnegative parameters subject to the usual normalization constraint. If the graph is fully connected then we have a completely general distribution having $K^M - 1$ parameters, whereas if there are no links in the graph the joint distribution factorizes into the product of the marginals, and the total number of parameters is $M(K - 1)$. Graphs having intermediate levels of connectivity allow for more general distributions than the fully factorized one while requiring fewer parameters than the general joint distribution. As an illustration, consider the chain of nodes shown in Figure 8.10. The marginal distribution $p(\mathbf{x}_1)$ requires $K - 1$ parameters, whereas each of the $M - 1$ conditional distributions $p(\mathbf{x}_i|\mathbf{x}_{i-1})$, for $i = 2, \dots, M$, requires $K(K - 1)$ parameters. This gives a total parameter count of $K - 1 + (M - 1)K(K - 1)$, which is quadratic in K and which grows linearly (rather than exponentially) with the length M of the chain.

An alternative way to reduce the number of independent parameters in a model is by *sharing* parameters (also known as *tying* of parameters). For instance, in the chain example of Figure 8.10, we can arrange that all of the conditional distributions $p(\mathbf{x}_i|\mathbf{x}_{i-1})$, for $i = 2, \dots, M$, are governed by the same set of $K(K - 1)$ parameters. Together with the $K - 1$ parameters governing the distribution of \mathbf{x}_1 , this gives a total of $K^2 - 1$ parameters that must be specified in order to define the joint distribution.

We can turn a graph over discrete variables into a Bayesian model by introducing Dirichlet priors for the parameters. From a graphical point of view, each node then acquires an additional parent representing the Dirichlet distribution over the parameters associated with the corresponding discrete node. This is illustrated for the chain model in Figure 8.11. The corresponding model in which we tie the parameters governing the conditional distributions $p(\mathbf{x}_i|\mathbf{x}_{i-1})$, for $i = 2, \dots, M$, is shown in Figure 8.12.

Another way of controlling the exponential growth in the number of parameters in models of discrete variables is to use parameterized models for the conditional distributions instead of complete tables of conditional probability values. To illustrate this idea, consider the graph in Figure 8.13 in which all of the nodes represent binary variables. Each of the parent variables x_i is governed by a single parame-

Figure 8.11 An extension of the model of Figure 8.10 to include Dirichlet priors over the parameters governing the discrete distributions.

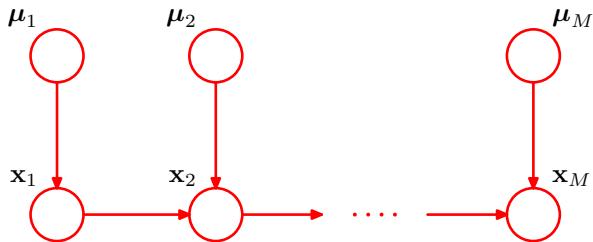
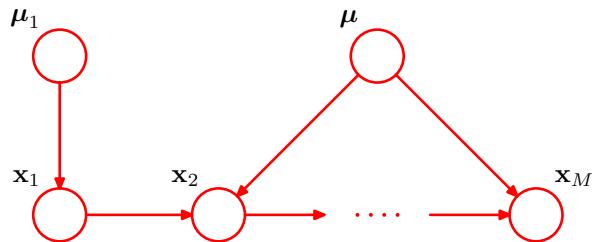


Figure 8.12 As in Figure 8.11 but with a single set of parameters μ shared amongst all of the conditional distributions $p(x_i|x_{i-1})$.

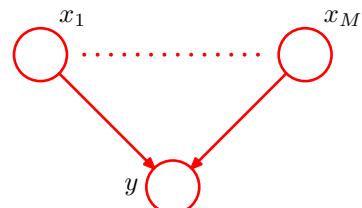


parameter μ_i representing the probability $p(x_i = 1)$, giving M parameters in total for the parent nodes. The conditional distribution $p(y|x_1, \dots, x_M)$, however, would require 2^M parameters representing the probability $p(y = 1)$ for each of the 2^M possible settings of the parent variables. Thus in general the number of parameters required to specify this conditional distribution will grow exponentially with M . We can obtain a more parsimonious form for the conditional distribution by using a logistic sigmoid function acting on a linear combination of the parent variables, giving

$$p(y = 1|x_1, \dots, x_M) = \sigma \left(w_0 + \sum_{i=1}^M w_i x_i \right) = \sigma(\mathbf{w}^T \mathbf{x}) \quad (8.10)$$

where $\sigma(a) = (1 + \exp(-a))^{-1}$ is the logistic sigmoid, $\mathbf{x} = (x_0, x_1, \dots, x_M)^T$ is an $(M + 1)$ -dimensional vector of parent states augmented with an additional variable x_0 whose value is clamped to 1, and $\mathbf{w} = (w_0, w_1, \dots, w_M)^T$ is a vector of $M + 1$ parameters. This is a more restricted form of conditional distribution than the general case but is now governed by a number of parameters that grows linearly with M . In this sense, it is analogous to the choice of a restrictive form of covariance matrix (for example, a diagonal matrix) in a multivariate Gaussian distribution. The motivation for the logistic sigmoid representation was discussed in Section 4.2.

Figure 8.13 A graph comprising M parents x_1, \dots, x_M and a single child y , used to illustrate the idea of parameterized conditional distributions for discrete variables.



Section 2.4

8.1.4 Linear-Gaussian models

In the previous section, we saw how to construct joint probability distributions over a set of discrete variables by expressing the variables as nodes in a directed acyclic graph. Here we show how a multivariate Gaussian can be expressed as a directed graph corresponding to a linear-Gaussian model over the component variables. This allows us to impose interesting structure on the distribution, with the general Gaussian and the diagonal covariance Gaussian representing opposite extremes. Several widely used techniques are examples of linear-Gaussian models, such as probabilistic principal component analysis, factor analysis, and linear dynamical systems (Roweis and Ghahramani, 1999). We shall make extensive use of the results of this section in later chapters when we consider some of these techniques in detail.

Consider an arbitrary directed acyclic graph over D variables in which node i represents a single continuous random variable x_i having a Gaussian distribution. The mean of this distribution is taken to be a linear combination of the states of its parent nodes pa_i of node i

$$p(x_i|\text{pa}_i) = \mathcal{N}\left(x_i \middle| \sum_{j \in \text{pa}_i} w_{ij}x_j + b_i, v_i\right) \quad (8.11)$$

where w_{ij} and b_i are parameters governing the mean, and v_i is the variance of the conditional distribution for x_i . The log of the joint distribution is then the log of the product of these conditionals over all nodes in the graph and hence takes the form

$$\ln p(\mathbf{x}) = \sum_{i=1}^D \ln p(x_i|\text{pa}_i) \quad (8.12)$$

$$= -\sum_{i=1}^D \frac{1}{2v_i} \left(x_i - \sum_{j \in \text{pa}_i} w_{ij}x_j - b_i \right)^2 + \text{const} \quad (8.13)$$

where $\mathbf{x} = (x_1, \dots, x_D)^T$ and ‘const’ denotes terms independent of \mathbf{x} . We see that this is a quadratic function of the components of \mathbf{x} , and hence the joint distribution $p(\mathbf{x})$ is a multivariate Gaussian.

We can determine the mean and covariance of the joint distribution recursively as follows. Each variable x_i has (conditional on the states of its parents) a Gaussian distribution of the form (8.11) and so

$$x_i = \sum_{j \in \text{pa}_i} w_{ij}x_j + b_i + \sqrt{v_i}\epsilon_i \quad (8.14)$$

where ϵ_i is a zero mean, unit variance Gaussian random variable satisfying $\mathbb{E}[\epsilon_i] = 0$ and $\mathbb{E}[\epsilon_i \epsilon_j] = I_{ij}$, where I_{ij} is the i, j element of the identity matrix. Taking the expectation of (8.14), we have

$$\mathbb{E}[x_i] = \sum_{j \in \text{pa}_i} w_{ij}\mathbb{E}[x_j] + b_i. \quad (8.15)$$

Figure 8.14 A directed graph over three Gaussian variables, with one missing link.



Thus we can find the components of $\mathbb{E}[\mathbf{x}] = (\mathbb{E}[x_1], \dots, \mathbb{E}[x_D])^T$ by starting at the lowest numbered node and working recursively through the graph (here we again assume that the nodes are numbered such that each node has a higher number than its parents). Similarly, we can use (8.14) and (8.15) to obtain the i, j element of the covariance matrix for $p(\mathbf{x})$ in the form of a recursion relation

$$\begin{aligned}\text{cov}[x_i, x_j] &= \mathbb{E}[(x_i - \mathbb{E}[x_i])(x_j - \mathbb{E}[x_j])] \\ &= \mathbb{E}\left[(x_i - \mathbb{E}[x_i]) \left\{ \sum_{k \in \text{pa}_j} w_{jk}(x_k - \mathbb{E}[x_k]) + \sqrt{v_j} \epsilon_j \right\}\right] \\ &= \sum_{k \in \text{pa}_j} w_{jk} \text{cov}[x_i, x_k] + I_{ij} v_j\end{aligned}\quad (8.16)$$

and so the covariance can similarly be evaluated recursively starting from the lowest numbered node.

Let us consider two extreme cases. First of all, suppose that there are no links in the graph, which therefore comprises D isolated nodes. In this case, there are no parameters w_{ij} and so there are just D parameters b_i and D parameters v_i . From the recursion relations (8.15) and (8.16), we see that the mean of $p(\mathbf{x})$ is given by $(b_1, \dots, b_D)^T$ and the covariance matrix is diagonal of the form $\text{diag}(v_1, \dots, v_D)$. The joint distribution has a total of $2D$ parameters and represents a set of D independent univariate Gaussian distributions.

Now consider a fully connected graph in which each node has all lower numbered nodes as parents. The matrix w_{ij} then has $i - 1$ entries on the i^{th} row and hence is a lower triangular matrix (with no entries on the leading diagonal). Then the total number of parameters w_{ij} is obtained by taking the number D^2 of elements in a $D \times D$ matrix, subtracting D to account for the absence of elements on the leading diagonal, and then dividing by 2 because the matrix has elements only below the diagonal, giving a total of $D(D - 1)/2$. The total number of independent parameters $\{w_{ij}\}$ and $\{v_i\}$ in the covariance matrix is therefore $D(D + 1)/2$ corresponding to a general symmetric covariance matrix.

Graphs having some intermediate level of complexity correspond to joint Gaussian distributions with partially constrained covariance matrices. Consider for example the graph shown in Figure 8.14, which has a link missing between variables x_1 and x_3 . Using the recursion relations (8.15) and (8.16), we see that the mean and covariance of the joint distribution are given by

$$\boldsymbol{\mu} = (b_1, b_2 + w_{21}b_1, b_3 + w_{32}b_2 + w_{32}w_{21}b_1)^T \quad (8.17)$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} v_1 & w_{21}v_1 & w_{32}w_{21}v_1 \\ w_{21}v_1 & v_2 + w_{21}^2v_1 & w_{32}(v_2 + w_{21}^2v_1) \\ w_{32}w_{21}v_1 & w_{32}(v_2 + w_{21}^2v_1) & v_3 + w_{32}^2(v_2 + w_{21}^2v_1) \end{pmatrix}. \quad (8.18)$$

Section 2.3

Exercise 8.7

We can readily extend the linear-Gaussian graphical model to the case in which the nodes of the graph represent multivariate Gaussian variables. In this case, we can write the conditional distribution for node i in the form

$$p(\mathbf{x}_i | \text{pa}_i) = \mathcal{N} \left(\mathbf{x}_i \middle| \sum_{j \in \text{pa}_i} \mathbf{W}_{ij} \mathbf{x}_j + \mathbf{b}_i, \boldsymbol{\Sigma}_i \right) \quad (8.19)$$

where now \mathbf{W}_{ij} is a matrix (which is nonsquare if \mathbf{x}_i and \mathbf{x}_j have different dimensionalities). Again it is easy to verify that the joint distribution over all variables is Gaussian.

Note that we have already encountered a specific example of the linear-Gaussian relationship when we saw that the conjugate prior for the mean μ of a Gaussian variable x is itself a Gaussian distribution over μ . The joint distribution over x and μ is therefore Gaussian. This corresponds to a simple two-node graph in which the node representing μ is the parent of the node representing x . The mean of the distribution over μ is a parameter controlling a prior, and so it can be viewed as a hyperparameter. Because the value of this hyperparameter may itself be unknown, we can again treat it from a Bayesian perspective by introducing a prior over the hyperparameter, sometimes called a *hyperprior*, which is again given by a Gaussian distribution. This type of construction can be extended in principle to any level and is an illustration of a *hierarchical Bayesian model*, of which we shall encounter further examples in later chapters.

Section 2.3.6

8.2. Conditional Independence

An important concept for probability distributions over multiple variables is that of *conditional independence* (Dawid, 1980). Consider three variables a , b , and c , and suppose that the conditional distribution of a , given b and c , is such that it does not depend on the value of b , so that

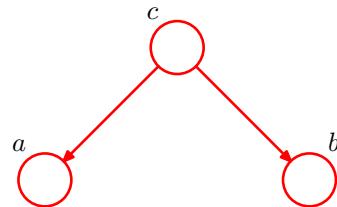
$$p(a|b, c) = p(a|c). \quad (8.20)$$

We say that a is conditionally independent of b given c . This can be expressed in a slightly different way if we consider the joint distribution of a and b conditioned on c , which we can write in the form

$$\begin{aligned} p(a, b|c) &= p(a|b, c)p(b|c) \\ &= p(a|c)p(b|c). \end{aligned} \quad (8.21)$$

where we have used the product rule of probability together with (8.20). Thus we see that, conditioned on c , the joint distribution of a and b factorizes into the product of the marginal distribution of a and the marginal distribution of b (again both conditioned on c). This says that the variables a and b are statistically independent, given c . Note that our definition of conditional independence will require that (8.20),

Figure 8.15 The first of three examples of graphs over three variables a , b , and c used to discuss conditional independence properties of directed graphical models.



or equivalently (8.21), must hold for every possible value of c , and not just for some values. We shall sometimes use a shorthand notation for conditional independence (Dawid, 1979) in which

$$a \perp\!\!\!\perp b \mid c \quad (8.22)$$

denotes that a is conditionally independent of b given c and is equivalent to (8.20).

Conditional independence properties play an important role in using probabilistic models for pattern recognition by simplifying both the structure of a model and the computations needed to perform inference and learning under that model. We shall see examples of this shortly.

If we are given an expression for the joint distribution over a set of variables in terms of a product of conditional distributions (i.e., the mathematical representation underlying a directed graph), then we could in principle test whether any potential conditional independence property holds by repeated application of the sum and product rules of probability. In practice, such an approach would be very time consuming. An important and elegant feature of graphical models is that conditional independence properties of the joint distribution can be read directly from the graph without having to perform any analytical manipulations. The general framework for achieving this is called *d-separation*, where the ‘d’ stands for ‘directed’ (Pearl, 1988). Here we shall motivate the concept of d-separation and give a general statement of the d-separation criterion. A formal proof can be found in Lauritzen (1996).

8.2.1 Three example graphs

We begin our discussion of the conditional independence properties of directed graphs by considering three simple examples each involving graphs having just three nodes. Together, these will motivate and illustrate the key concepts of d-separation. The first of the three examples is shown in Figure 8.15, and the joint distribution corresponding to this graph is easily written down using the general result (8.5) to give

$$p(a, b, c) = p(a|c)p(b|c)p(c). \quad (8.23)$$

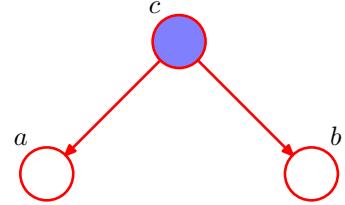
If none of the variables are observed, then we can investigate whether a and b are independent by marginalizing both sides of (8.23) with respect to c to give

$$p(a, b) = \sum_c p(a|c)p(b|c)p(c). \quad (8.24)$$

In general, this does not factorize into the product $p(a)p(b)$, and so

$$a \not\perp\!\!\!\perp b \mid \emptyset \quad (8.25)$$

Figure 8.16 As in Figure 8.15 but where we have conditioned on the value of variable c .



where \emptyset denotes the empty set, and the symbol \nparallel means that the conditional independence property does not hold in general. Of course, it may hold for a particular distribution by virtue of the specific numerical values associated with the various conditional probabilities, but it does not follow in general from the structure of the graph.

Now suppose we condition on the variable c , as represented by the graph of Figure 8.16. From (8.23), we can easily write down the conditional distribution of a and b , given c , in the form

$$\begin{aligned} p(a, b|c) &= \frac{p(a, b, c)}{p(c)} \\ &= p(a|c)p(b|c) \end{aligned}$$

and so we obtain the conditional independence property

$$a \perp\!\!\!\perp b | c.$$

We can provide a simple graphical interpretation of this result by considering the path from node a to node b via c . The node c is said to be *tail-to-tail* with respect to this path because the node is connected to the tails of the two arrows, and the presence of such a path connecting nodes a and b causes these nodes to be dependent. However, when we condition on node c , as in Figure 8.16, the conditioned node ‘blocks’ the path from a to b and causes a and b to become (conditionally) independent.

We can similarly consider the graph shown in Figure 8.17. The joint distribution corresponding to this graph is again obtained from our general formula (8.5) to give

$$p(a, b, c) = p(a)p(c|a)p(b|c). \quad (8.26)$$

First of all, suppose that none of the variables are observed. Again, we can test to see if a and b are independent by marginalizing over c to give

$$p(a, b) = p(a) \sum_c p(c|a)p(b|c) = p(a)p(b|a).$$

Figure 8.17

The second of our three examples of 3-node graphs used to motivate the conditional independence framework for directed graphical models.

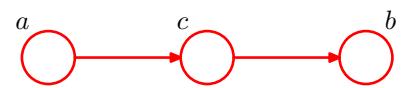
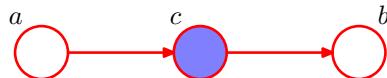


Figure 8.18 As in Figure 8.17 but now conditioning on node c .

which in general does not factorize into $p(a)p(b)$, and so

$$a \not\perp\!\!\!\perp b \mid \emptyset \quad (8.27)$$

as before.

Now suppose we condition on node c , as shown in Figure 8.18. Using Bayes' theorem, together with (8.26), we obtain

$$\begin{aligned} p(a, b|c) &= \frac{p(a, b, c)}{p(c)} \\ &= \frac{p(a)p(c|a)p(b|c)}{p(c)} \\ &= p(a|c)p(b|c) \end{aligned}$$

and so again we obtain the conditional independence property

$$a \perp\!\!\!\perp b \mid c.$$

As before, we can interpret these results graphically. The node c is said to be *head-to-tail* with respect to the path from node a to node b . Such a path connects nodes a and b and renders them dependent. If we now observe c , as in Figure 8.18, then this observation ‘blocks’ the path from a to b and so we obtain the conditional independence property $a \perp\!\!\!\perp b \mid c$.

Finally, we consider the third of our 3-node examples, shown by the graph in Figure 8.19. As we shall see, this has a more subtle behaviour than the two previous graphs.

The joint distribution can again be written down using our general result (8.5) to give

$$p(a, b, c) = p(a)p(b)p(c|a, b). \quad (8.28)$$

Consider first the case where none of the variables are observed. Marginalizing both sides of (8.28) over c we obtain

$$p(a, b) = p(a)p(b)$$

Figure 8.19

The last of our three examples of 3-node graphs used to explore conditional independence properties in graphical models. This graph has rather different properties from the two previous examples.

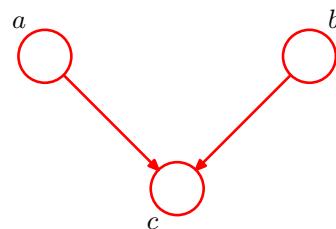
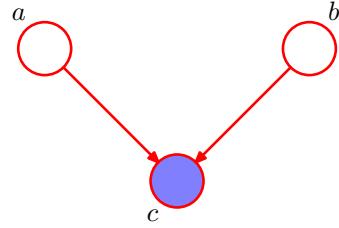


Figure 8.20 As in Figure 8.19 but conditioning on the value of node c . In this graph, the act of conditioning induces a dependence between a and b .



and so a and b are independent with no variables observed, in contrast to the two previous examples. We can write this result as

$$a \perp\!\!\!\perp b \mid \emptyset. \quad (8.29)$$

Now suppose we condition on c , as indicated in Figure 8.20. The conditional distribution of a and b is then given by

$$\begin{aligned} p(a, b|c) &= \frac{p(a, b, c)}{p(c)} \\ &= \frac{p(a)p(b)p(c|a, b)}{p(c)} \end{aligned}$$

which in general does not factorize into the product $p(a)p(b)$, and so

$$a \not\perp\!\!\!\perp b \mid c.$$

Thus our third example has the opposite behaviour from the first two. Graphically, we say that node c is *head-to-head* with respect to the path from a to b because it connects to the heads of the two arrows. When node c is unobserved, it ‘blocks’ the path, and the variables a and b are independent. However, conditioning on c ‘unblocks’ the path and renders a and b dependent.

There is one more subtlety associated with this third example that we need to consider. First we introduce some more terminology. We say that node y is a *descendant* of node x if there is a path from x to y in which each step of the path follows the directions of the arrows. Then it can be shown that a head-to-head path will become unblocked if either the node, *or any of its descendants*, is observed.

In summary, a tail-to-tail node or a head-to-tail node leaves a path unblocked unless it is observed in which case it blocks the path. By contrast, a head-to-head node blocks a path if it is unobserved, but once the node, and/or at least one of its descendants, is observed the path becomes unblocked.

It is worth spending a moment to understand further the unusual behaviour of the graph of Figure 8.20. Consider a particular instance of such a graph corresponding to a problem with three binary random variables relating to the fuel system on a car, as shown in Figure 8.21. The variables are called B , representing the state of a battery that is either charged ($B = 1$) or flat ($B = 0$), F representing the state of the fuel tank that is either full of fuel ($F = 1$) or empty ($F = 0$), and G , which is the state of an electric fuel gauge and which indicates either full ($G = 1$) or empty

Exercise 8.10

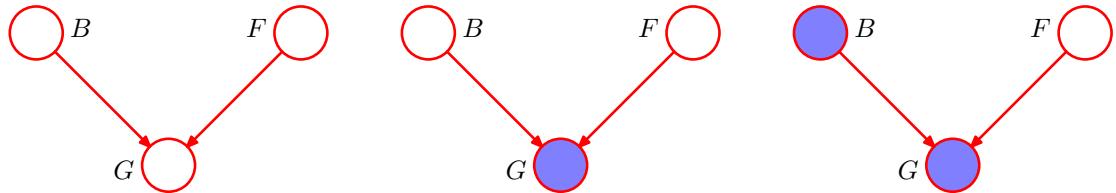


Figure 8.21 An example of a 3-node graph used to illustrate the phenomenon of ‘explaining away’. The three nodes represent the state of the battery (B), the state of the fuel tank (F) and the reading on the electric fuel gauge (G). See the text for details.

($G = 0$). The battery is either charged or flat, and independently the fuel tank is either full or empty, with prior probabilities

$$\begin{aligned} p(B = 1) &= 0.9 \\ p(F = 1) &= 0.9. \end{aligned}$$

Given the state of the fuel tank and the battery, the fuel gauge reads full with probabilities given by

$$\begin{aligned} p(G = 1|B = 1, F = 1) &= 0.8 \\ p(G = 1|B = 1, F = 0) &= 0.2 \\ p(G = 1|B = 0, F = 1) &= 0.2 \\ p(G = 1|B = 0, F = 0) &= 0.1 \end{aligned}$$

so this is a rather unreliable fuel gauge! All remaining probabilities are determined by the requirement that probabilities sum to one, and so we have a complete specification of the probabilistic model.

Before we observe any data, the prior probability of the fuel tank being empty is $p(F = 0) = 0.1$. Now suppose that we observe the fuel gauge and discover that it reads empty, i.e., $G = 0$, corresponding to the middle graph in Figure 8.21. We can use Bayes’ theorem to evaluate the posterior probability of the fuel tank being empty. First we evaluate the denominator for Bayes’ theorem given by

$$p(G = 0) = \sum_{B \in \{0,1\}} \sum_{F \in \{0,1\}} p(G = 0|B, F)p(B)p(F) = 0.315 \quad (8.30)$$

and similarly we evaluate

$$p(G = 0|F = 0) = \sum_{B \in \{0,1\}} p(G = 0|B, F = 0)p(B) = 0.81 \quad (8.31)$$

and using these results we have

$$p(F = 0|G = 0) = \frac{p(G = 0|F = 0)p(F = 0)}{p(G = 0)} \simeq 0.257 \quad (8.32)$$

and so $p(F = 0|G = 0) > p(F = 0)$. Thus observing that the gauge reads empty makes it more likely that the tank is indeed empty, as we would intuitively expect. Next suppose that we also check the state of the battery and find that it is flat, i.e., $B = 0$. We have now observed the states of both the fuel gauge and the battery, as shown by the right-hand graph in Figure 8.21. The posterior probability that the fuel tank is empty given the observations of both the fuel gauge and the battery state is then given by

$$p(F = 0|G = 0, B = 0) = \frac{p(G = 0|B = 0, F = 0)p(F = 0)}{\sum_{F \in \{0,1\}} p(G = 0|B = 0, F)p(F)} \simeq 0.111 \quad (8.33)$$

where the prior probability $p(B = 0)$ has cancelled between numerator and denominator. Thus the probability that the tank is empty has *decreased* (from 0.257 to 0.111) as a result of the observation of the state of the battery. This accords with our intuition that finding out that the battery is flat *explains away* the observation that the fuel gauge reads empty. We see that the state of the fuel tank and that of the battery have indeed become dependent on each other as a result of observing the reading on the fuel gauge. In fact, this would also be the case if, instead of observing the fuel gauge directly, we observed the state of some descendant of G . Note that the probability $p(F = 0|G = 0, B = 0) \simeq 0.111$ is greater than the prior probability $p(F = 0) = 0.1$ because the observation that the fuel gauge reads zero still provides some evidence in favour of an empty fuel tank.

8.2.2 D-separation

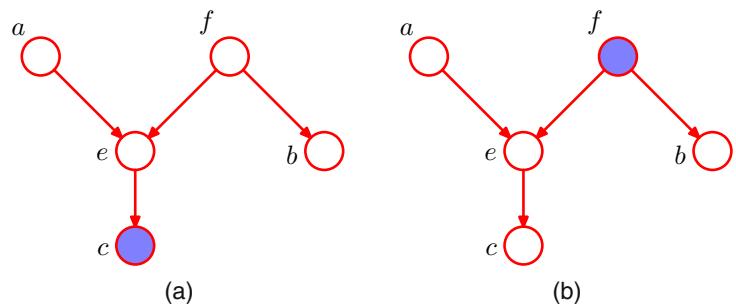
We now give a general statement of the d-separation property (Pearl, 1988) for directed graphs. Consider a general directed graph in which A , B , and C are arbitrary nonintersecting sets of nodes (whose union may be smaller than the complete set of nodes in the graph). We wish to ascertain whether a particular conditional independence statement $A \perp\!\!\!\perp B | C$ is implied by a given directed acyclic graph. To do so, we consider all possible paths from any node in A to any node in B . Any such path is said to be *blocked* if it includes a node such that either

- (a) the arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in the set C , or
- (b) the arrows meet head-to-head at the node, and neither the node, nor any of its descendants, is in the set C .

If all paths are blocked, then A is said to be d-separated from B by C , and the joint distribution over all of the variables in the graph will satisfy $A \perp\!\!\!\perp B | C$.

The concept of d-separation is illustrated in Figure 8.22. In graph (a), the path from a to b is not blocked by node f because it is a tail-to-tail node for this path and is not observed, nor is it blocked by node e because, although the latter is a head-to-head node, it has a descendant c because c is in the conditioning set. Thus the conditional independence statement $a \perp\!\!\!\perp b | c$ does *not* follow from this graph. In graph (b), the path from a to b is blocked by node f because this is a tail-to-tail node that is observed, and so the conditional independence property $a \perp\!\!\!\perp b | f$ will

Figure 8.22 Illustration of the concept of d-separation. See the text for details.



be satisfied by any distribution that factorizes according to this graph. Note that this path is also blocked by node e because e is a head-to-head node and neither it nor its descendant are in the conditioning set.

For the purposes of d-separation, parameters such as α and σ^2 in Figure 8.5, indicated by small filled circles, behave in the same way as observed nodes. However, there are no marginal distributions associated with such nodes. Consequently parameter nodes never themselves have parents and so all paths through these nodes will always be tail-to-tail and hence blocked. Consequently they play no role in d-separation.

Another example of conditional independence and d-separation is provided by the concept of i.i.d. (independent identically distributed) data introduced in Section 1.2.4. Consider the problem of finding the posterior distribution for the mean of a univariate Gaussian distribution. This can be represented by the directed graph shown in Figure 8.23 in which the joint distribution is defined by a prior $p(\mu)$ together with a set of conditional distributions $p(x_n|\mu)$ for $n = 1, \dots, N$. In practice, we observe $\mathcal{D} = \{x_1, \dots, x_N\}$ and our goal is to infer μ . Suppose, for a moment, that we condition on μ and consider the joint distribution of the observations. Using d-separation, we note that there is a unique path from any x_i to any other $x_j \neq i$ and that this path is tail-to-tail with respect to the observed node μ . Every such path is blocked and so the observations $D = \{x_1, \dots, x_N\}$ are independent given μ , so that

$$p(\mathcal{D}|\mu) = \prod_{n=1}^N p(x_n|\mu). \quad (8.34)$$

Figure 8.23 (a) Directed graph corresponding to the problem of inferring the mean μ of a univariate Gaussian distribution from observations x_1, \dots, x_N . (b) The same graph drawn using the plate notation.

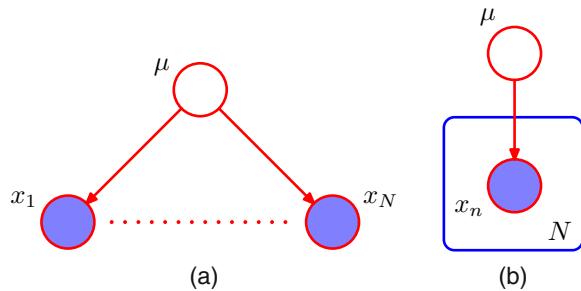
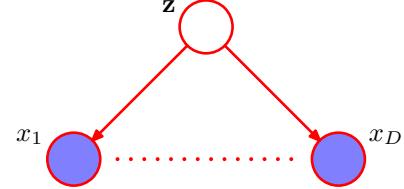


Figure 8.24 A graphical representation of the ‘naive Bayes’ model for classification. Conditioned on the class label \mathbf{z} , the components of the observed vector $\mathbf{x} = (x_1, \dots, x_D)^T$ are assumed to be independent.



However, if we integrate over μ , the observations are in general no longer independent

$$p(\mathcal{D}) = \int_0^\infty p(\mathcal{D}|\mu)p(\mu) d\mu \neq \prod_{n=1}^N p(x_n). \quad (8.35)$$

Here μ is a latent variable, because its value is not observed.

Another example of a model representing i.i.d. data is the graph in Figure 8.7 corresponding to Bayesian polynomial regression. Here the stochastic nodes correspond to $\{t_n\}$, \mathbf{w} and \hat{t} . We see that the node for \mathbf{w} is tail-to-tail with respect to the path from \hat{t} to any one of the nodes t_n and so we have the following conditional independence property

$$\hat{t} \perp\!\!\!\perp t_n \mid \mathbf{w}. \quad (8.36)$$

Thus, conditioned on the polynomial coefficients \mathbf{w} , the predictive distribution for \hat{t} is independent of the training data $\{t_1, \dots, t_N\}$. We can therefore first use the training data to determine the posterior distribution over the coefficients \mathbf{w} and then we can discard the training data and use the posterior distribution for \mathbf{w} to make predictions of \hat{t} for new input observations \hat{x} .

A related graphical structure arises in an approach to classification called the *naive Bayes* model, in which we use conditional independence assumptions to simplify the model structure. Suppose our observed variable consists of a D -dimensional vector $\mathbf{x} = (x_1, \dots, x_D)^T$, and we wish to assign observed values of \mathbf{x} to one of K classes. Using the 1-of- K encoding scheme, we can represent these classes by a K -dimensional binary vector \mathbf{z} . We can then define a generative model by introducing a multinomial prior $p(\mathbf{z}|\mu)$ over the class labels, where the k^{th} component μ_k of μ is the prior probability of class C_k , together with a conditional distribution $p(\mathbf{x}|\mathbf{z})$ for the observed vector \mathbf{x} . The key assumption of the naive Bayes model is that, conditioned on the class \mathbf{z} , the distributions of the input variables x_1, \dots, x_D are independent. The graphical representation of this model is shown in Figure 8.24. We see that observation of \mathbf{z} blocks the path between x_i and x_j for $j \neq i$ (because such paths are tail-to-tail at the node \mathbf{z}) and so x_i and x_j are conditionally independent given \mathbf{z} . If, however, we marginalize out \mathbf{z} (so that \mathbf{z} is unobserved) the tail-to-tail path from x_i to x_j is no longer blocked. This tells us that in general the marginal density $p(\mathbf{x})$ will not factorize with respect to the components of \mathbf{x} . We encountered a simple application of the naive Bayes model in the context of fusing data from different sources for medical diagnosis in Section 1.5.

If we are given a labelled training set, comprising inputs $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ together with their class labels, then we can fit the naive Bayes model to the training data

Section 3.3

using maximum likelihood assuming that the data are drawn independently from the model. The solution is obtained by fitting the model for each class separately using the correspondingly labelled data. As an example, suppose that the probability density within each class is chosen to be Gaussian. In this case, the naive Bayes assumption then implies that the covariance matrix for each Gaussian is diagonal, and the contours of constant density within each class will be axis-aligned ellipsoids. The marginal density, however, is given by a superposition of diagonal Gaussians (with weighting coefficients given by the class priors) and so will no longer factorize with respect to its components.

The naive Bayes assumption is helpful when the dimensionality D of the input space is high, making density estimation in the full D -dimensional space more challenging. It is also useful if the input vector contains both discrete and continuous variables, since each can be represented separately using appropriate models (e.g., Bernoulli distributions for binary observations or Gaussians for real-valued variables). The conditional independence assumption of this model is clearly a strong one that may lead to rather poor representations of the class-conditional densities. Nevertheless, even if this assumption is not precisely satisfied, the model may still give good classification performance in practice because the decision boundaries can be insensitive to some of the details in the class-conditional densities, as illustrated in Figure 1.27.

We have seen that a particular directed graph represents a specific decomposition of a joint probability distribution into a product of conditional probabilities. The graph also expresses a set of conditional independence statements obtained through the d-separation criterion, and the d-separation theorem is really an expression of the equivalence of these two properties. In order to make this clear, it is helpful to think of a directed graph as a filter. Suppose we consider a particular joint probability distribution $p(\mathbf{x})$ over the variables \mathbf{x} corresponding to the (nonobserved) nodes of the graph. The filter will allow this distribution to pass through if, and only if, it can be expressed in terms of the factorization (8.5) implied by the graph. If we present to the filter the set of all possible distributions $p(\mathbf{x})$ over the set of variables \mathbf{x} , then the subset of distributions that are passed by the filter will be denoted \mathcal{DF} , for *directed factorization*. This is illustrated in Figure 8.25. Alternatively, we can use the graph as a different kind of filter by first listing all of the conditional independence properties obtained by applying the d-separation criterion to the graph, and then allowing a distribution to pass only if it satisfies all of these properties. If we present all possible distributions $p(\mathbf{x})$ to this second kind of filter, then the d-separation theorem tells us that the set of distributions that will be allowed through is precisely the set \mathcal{DF} .

It should be emphasized that the conditional independence properties obtained from d-separation apply to any probabilistic model described by that particular directed graph. This will be true, for instance, whether the variables are discrete or continuous or a combination of these. Again, we see that a particular graph is describing a whole family of probability distributions.

At one extreme we have a fully connected graph that exhibits no conditional independence properties at all, and which can represent any possible joint probability distribution over the given variables. The set \mathcal{DF} will contain all possible distribu-

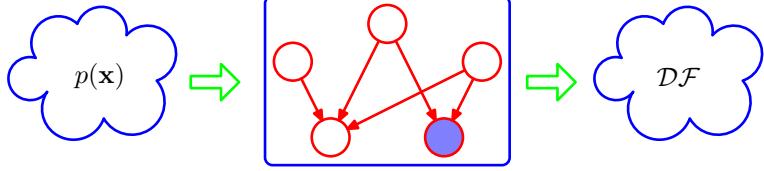


Figure 8.25 We can view a graphical model (in this case a directed graph) as a filter in which a probability distribution $p(\mathbf{x})$ is allowed through the filter if, and only if, it satisfies the directed factorization property (8.5). The set of all possible probability distributions $p(\mathbf{x})$ that pass through the filter is denoted \mathcal{DF} . We can alternatively use the graph to filter distributions according to whether they respect all of the conditional independencies implied by the d-separation properties of the graph. The d-separation theorem says that it is the same set of distributions \mathcal{DF} that will be allowed through this second kind of filter.

tions $p(\mathbf{x})$. At the other extreme, we have the fully disconnected graph, i.e., one having no links at all. This corresponds to joint distributions which factorize into the product of the marginal distributions over the variables comprising the nodes of the graph.

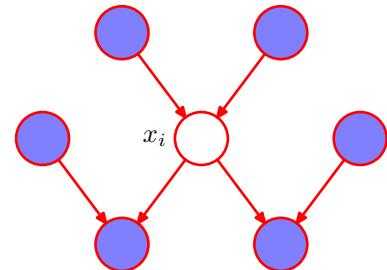
Note that for any given graph, the set of distributions \mathcal{DF} will include any distributions that have additional independence properties beyond those described by the graph. For instance, a fully factorized distribution will always be passed through the filter implied by any graph over the corresponding set of variables.

We end our discussion of conditional independence properties by exploring the concept of a *Markov blanket* or *Markov boundary*. Consider a joint distribution $p(\mathbf{x}_1, \dots, \mathbf{x}_D)$ represented by a directed graph having D nodes, and consider the conditional distribution of a particular node with variables \mathbf{x}_i conditioned on all of the remaining variables $\mathbf{x}_{j \neq i}$. Using the factorization property (8.5), we can express this conditional distribution in the form

$$\begin{aligned} p(\mathbf{x}_i | \mathbf{x}_{\{j \neq i\}}) &= \frac{p(\mathbf{x}_1, \dots, \mathbf{x}_D)}{\int p(\mathbf{x}_1, \dots, \mathbf{x}_D) d\mathbf{x}_i} \\ &= \frac{\prod_k p(\mathbf{x}_k | \text{pa}_k)}{\int \prod_k p(\mathbf{x}_k | \text{pa}_k) d\mathbf{x}_i} \end{aligned}$$

in which the integral is replaced by a summation in the case of discrete variables. We now observe that any factor $p(\mathbf{x}_k | \text{pa}_k)$ that does not have any functional dependence on \mathbf{x}_i can be taken outside the integral over \mathbf{x}_i , and will therefore cancel between numerator and denominator. The only factors that remain will be the conditional distribution $p(\mathbf{x}_i | \text{pa}_i)$ for node \mathbf{x}_i itself, together with the conditional distributions for any nodes \mathbf{x}_k such that node \mathbf{x}_i is in the conditioning set of $p(\mathbf{x}_k | \text{pa}_k)$, in other words for which \mathbf{x}_i is a parent of \mathbf{x}_k . The conditional $p(\mathbf{x}_i | \text{pa}_i)$ will depend on the parents of node \mathbf{x}_i , whereas the conditionals $p(\mathbf{x}_k | \text{pa}_k)$ will depend on the children

Figure 8.26 The Markov blanket of a node x_i comprises the set of parents, children and co-parents of the node. It has the property that the conditional distribution of x_i , conditioned on all the remaining variables in the graph, is dependent only on the variables in the Markov blanket.



of x_i as well as on the *co-parents*, in other words variables corresponding to parents of node x_k other than node x_i . The set of nodes comprising the parents, the children and the co-parents is called the Markov blanket and is illustrated in Figure 8.26. We can think of the Markov blanket of a node x_i as being the minimal set of nodes that isolates x_i from the rest of the graph. Note that it is not sufficient to include only the parents and children of node x_i because the phenomenon of explaining away means that observations of the child nodes will not block paths to the co-parents. We must therefore observe the co-parent nodes also.

8.3. Markov Random Fields

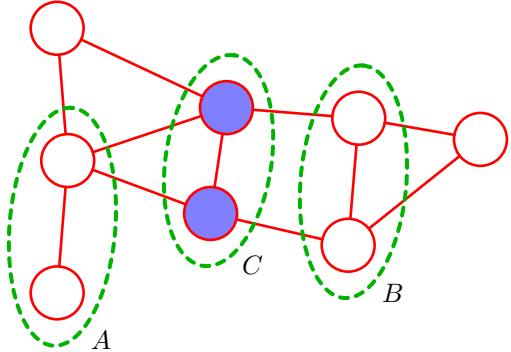
We have seen that directed graphical models specify a factorization of the joint distribution over a set of variables into a product of local conditional distributions. They also define a set of conditional independence properties that must be satisfied by any distribution that factorizes according to the graph. We turn now to the second major class of graphical models that are described by undirected graphs and that again specify both a factorization and a set of conditional independence relations.

A *Markov random field*, also known as a *Markov network* or an *undirected graphical model* (Kindermann and Snell, 1980), has a set of nodes each of which corresponds to a variable or group of variables, as well as a set of links each of which connects a pair of nodes. The links are undirected, that is they do not carry arrows. In the case of undirected graphs, it is convenient to begin with a discussion of conditional independence properties.

8.3.1 Conditional independence properties

In the case of directed graphs, we saw that it was possible to test whether a particular conditional independence property holds by applying a graphical test called d-separation. This involved testing whether or not the paths connecting two sets of nodes were ‘blocked’. The definition of blocked, however, was somewhat subtle due to the presence of paths having head-to-head nodes. We might ask whether it is possible to define an alternative graphical semantics for probability distributions such that conditional independence is determined by simple graph separation. This is indeed the case and corresponds to undirected graphical models. By removing the

Figure 8.27 An example of an undirected graph in which every path from any node in set A to any node in set B passes through at least one node in set C . Consequently the conditional independence property $A \perp\!\!\!\perp B | C$ holds for any probability distribution described by this graph.



directionality from the links of the graph, the asymmetry between parent and child nodes is removed, and so the subtleties associated with head-to-head nodes no longer arise.

Suppose that in an undirected graph we identify three sets of nodes, denoted A , B , and C , and that we consider the conditional independence property

$$A \perp\!\!\!\perp B | C. \quad (8.37)$$

To test whether this property is satisfied by a probability distribution defined by a graph we consider all possible paths that connect nodes in set A to nodes in set B . If all such paths pass through one or more nodes in set C , then all such paths are ‘blocked’ and so the conditional independence property holds. However, if there is at least one such path that is not blocked, then the property does not necessarily hold, or more precisely there will exist at least some distributions corresponding to the graph that do not satisfy this conditional independence relation. This is illustrated with an example in Figure 8.27. Note that this is exactly the same as the d-separation criterion except that there is no ‘explaining away’ phenomenon. Testing for conditional independence in undirected graphs is therefore simpler than in directed graphs.

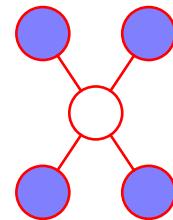
An alternative way to view the conditional independence test is to imagine removing all nodes in set C from the graph together with any links that connect to those nodes. We then ask if there exists a path that connects any node in A to any node in B . If there are no such paths, then the conditional independence property must hold.

The Markov blanket for an undirected graph takes a particularly simple form, because a node will be conditionally independent of all other nodes conditioned only on the neighbouring nodes, as illustrated in Figure 8.28.

8.3.2 Factorization properties

We now seek a factorization rule for undirected graphs that will correspond to the above conditional independence test. Again, this will involve expressing the joint distribution $p(\mathbf{x})$ as a product of functions defined over sets of variables that are local to the graph. We therefore need to decide what is the appropriate notion of locality in this case.

Figure 8.28 For an undirected graph, the Markov blanket of a node x_i consists of the set of neighbouring nodes. It has the property that the conditional distribution of x_i , conditioned on all the remaining variables in the graph, is dependent only on the variables in the Markov blanket.



If we consider two nodes x_i and x_j that are not connected by a link, then these variables must be conditionally independent given all other nodes in the graph. This follows from the fact that there is no direct path between the two nodes, and all other paths pass through nodes that are observed, and hence those paths are blocked. This conditional independence property can be expressed as

$$p(x_i, x_j | \mathbf{x}_{\setminus \{i,j\}}) = p(x_i | \mathbf{x}_{\setminus \{i,j\}})p(x_j | \mathbf{x}_{\setminus \{i,j\}}) \quad (8.38)$$

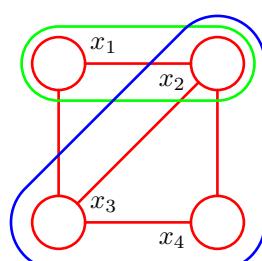
where $\mathbf{x}_{\setminus \{i,j\}}$ denotes the set \mathbf{x} of all variables with x_i and x_j removed. The factorization of the joint distribution must therefore be such that x_i and x_j do not appear in the same factor in order for the conditional independence property to hold for all possible distributions belonging to the graph.

This leads us to consider a graphical concept called a *clique*, which is defined as a subset of the nodes in a graph such that there exists a link between all pairs of nodes in the subset. In other words, the set of nodes in a clique is fully connected. Furthermore, a *maximal clique* is a clique such that it is not possible to include any other nodes from the graph in the set without it ceasing to be a clique. These concepts are illustrated by the undirected graph over four variables shown in Figure 8.29. This graph has five cliques of two nodes given by $\{x_1, x_2\}$, $\{x_2, x_3\}$, $\{x_3, x_4\}$, $\{x_4, x_2\}$, and $\{x_1, x_3\}$, as well as two maximal cliques given by $\{x_1, x_2, x_3\}$ and $\{x_2, x_3, x_4\}$. The set $\{x_1, x_2, x_3, x_4\}$ is not a clique because of the missing link from x_1 to x_4 .

We can therefore define the factors in the decomposition of the joint distribution to be functions of the variables in the cliques. In fact, we can consider functions of the maximal cliques, without loss of generality, because other cliques must be subsets of maximal cliques. Thus, if $\{x_1, x_2, x_3\}$ is a maximal clique and we define an arbitrary function over this clique, then including another factor defined over a subset of these variables would be redundant.

Let us denote a clique by C and the set of variables in that clique by \mathbf{x}_C . Then

Figure 8.29 A four-node undirected graph showing a clique (outlined in green) and a maximal clique (outlined in blue).



the joint distribution is written as a product of *potential functions* $\psi_C(\mathbf{x}_C)$ over the maximal cliques of the graph

$$p(\mathbf{x}) = \frac{1}{Z} \prod_C \psi_C(\mathbf{x}_C). \quad (8.39)$$

Here the quantity Z , sometimes called the *partition function*, is a normalization constant and is given by

$$Z = \sum_{\mathbf{x}} \prod_C \psi_C(\mathbf{x}_C) \quad (8.40)$$

which ensures that the distribution $p(\mathbf{x})$ given by (8.39) is correctly normalized. By considering only potential functions which satisfy $\psi_C(\mathbf{x}_C) \geq 0$ we ensure that $p(\mathbf{x}) \geq 0$. In (8.40) we have assumed that \mathbf{x} comprises discrete variables, but the framework is equally applicable to continuous variables, or a combination of the two, in which the summation is replaced by the appropriate combination of summation and integration.

Note that we do not restrict the choice of potential functions to those that have a specific probabilistic interpretation as marginal or conditional distributions. This is in contrast to directed graphs in which each factor represents the conditional distribution of the corresponding variable, conditioned on the state of its parents. However, in special cases, for instance where the undirected graph is constructed by starting with a directed graph, the potential functions may indeed have such an interpretation, as we shall see shortly.

One consequence of the generality of the potential functions $\psi_C(\mathbf{x}_C)$ is that their product will in general not be correctly normalized. We therefore have to introduce an explicit normalization factor given by (8.40). Recall that for directed graphs, the joint distribution was automatically normalized as a consequence of the normalization of each of the conditional distributions in the factorization.

The presence of this normalization constant is one of the major limitations of undirected graphs. If we have a model with M discrete nodes each having K states, then the evaluation of the normalization term involves summing over K^M states and so (in the worst case) is exponential in the size of the model. The partition function is needed for parameter learning because it will be a function of any parameters that govern the potential functions $\psi_C(\mathbf{x}_C)$. However, for evaluation of local conditional distributions, the partition function is not needed because a conditional is the ratio of two marginals, and the partition function cancels between numerator and denominator when evaluating this ratio. Similarly, for evaluating local marginal probabilities we can work with the unnormalized joint distribution and then normalize the marginals explicitly at the end. Provided the marginals only involves a small number of variables, the evaluation of their normalization coefficient will be feasible.

So far, we have discussed the notion of conditional independence based on simple graph separation and we have proposed a factorization of the joint distribution that is intended to correspond to this conditional independence structure. However, we have not made any formal connection between conditional independence and factorization for undirected graphs. To do so we need to restrict attention to potential functions $\psi_C(\mathbf{x}_C)$ that are strictly positive (i.e., never zero or negative for any

choice of \mathbf{x}_C). Given this restriction, we can make a precise relationship between factorization and conditional independence.

To do this we again return to the concept of a graphical model as a filter, corresponding to Figure 8.25. Consider the set of all possible distributions defined over a fixed set of variables corresponding to the nodes of a particular undirected graph. We can define $\mathcal{U}\mathcal{I}$ to be the set of such distributions that are consistent with the set of conditional independence statements that can be read from the graph using graph separation. Similarly, we can define $\mathcal{U}\mathcal{F}$ to be the set of such distributions that can be expressed as a factorization of the form (8.39) with respect to the maximal cliques of the graph. The *Hammersley-Clifford* theorem (Clifford, 1990) states that the sets $\mathcal{U}\mathcal{I}$ and $\mathcal{U}\mathcal{F}$ are identical.

Because we are restricted to potential functions which are strictly positive it is convenient to express them as exponentials, so that

$$\psi_C(\mathbf{x}_C) = \exp\{-E(\mathbf{x}_C)\} \quad (8.41)$$

where $E(\mathbf{x}_C)$ is called an *energy function*, and the exponential representation is called the *Boltzmann distribution*. The joint distribution is defined as the product of potentials, and so the total energy is obtained by adding the energies of each of the maximal cliques.

In contrast to the factors in the joint distribution for a directed graph, the potentials in an undirected graph do not have a specific probabilistic interpretation. Although this gives greater flexibility in choosing the potential functions, because there is no normalization constraint, it does raise the question of how to motivate a choice of potential function for a particular application. This can be done by viewing the potential function as expressing which configurations of the local variables are preferred to others. Global configurations that have a relatively high probability are those that find a good balance in satisfying the (possibly conflicting) influences of the clique potentials. We turn now to a specific example to illustrate the use of undirected graphs.

8.3.3 Illustration: Image de-noising

We can illustrate the application of undirected graphs using an example of noise removal from a binary image (Besag, 1974; Geman and Geman, 1984; Besag, 1986). Although a very simple example, this is typical of more sophisticated applications. Let the observed noisy image be described by an array of binary pixel values $y_i \in \{-1, +1\}$, where the index $i = 1, \dots, D$ runs over all pixels. We shall suppose that the image is obtained by taking an unknown noise-free image, described by binary pixel values $x_i \in \{-1, +1\}$ and randomly flipping the sign of pixels with some small probability. An example binary image, together with a noise corrupted image obtained by flipping the sign of the pixels with probability 10%, is shown in Figure 8.30. Given the noisy image, our goal is to recover the original noise-free image.

Because the noise level is small, we know that there will be a strong correlation between x_i and y_i . We also know that neighbouring pixels x_i and x_j in an image are strongly correlated. This prior knowledge can be captured using the Markov

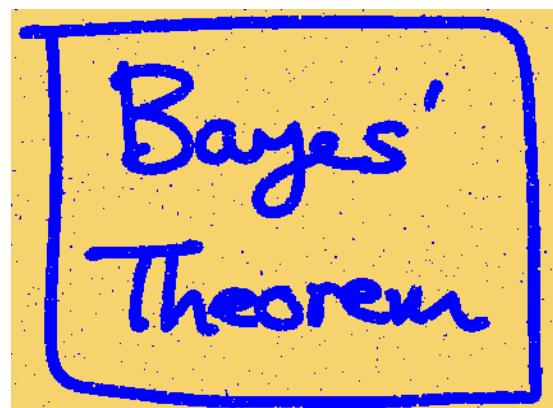
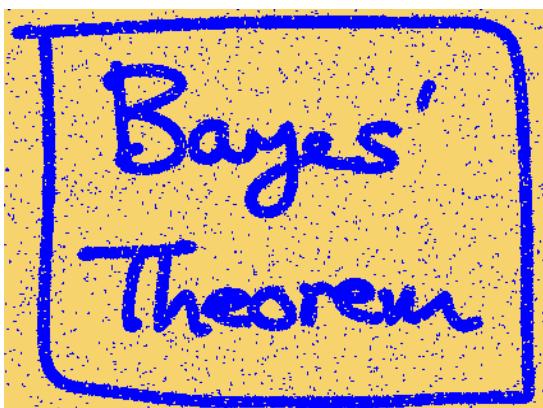
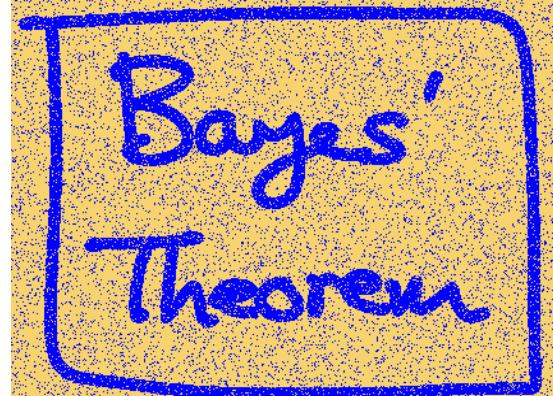


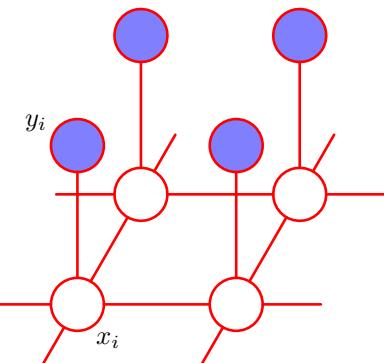
Figure 8.30 Illustration of image de-noising using a Markov random field. The top row shows the original binary image on the left and the corrupted image after randomly changing 10% of the pixels on the right. The bottom row shows the restored images obtained using iterated conditional models (ICM) on the left and using the graph-cut algorithm on the right. ICM produces an image where 96% of the pixels agree with the original image, whereas the corresponding number for graph-cut is 99%.

random field model whose undirected graph is shown in Figure 8.31. This graph has two types of cliques, each of which contains two variables. The cliques of the form $\{x_i, y_i\}$ have an associated energy function that expresses the correlation between these variables. We choose a very simple energy function for these cliques of the form $-\eta x_i y_i$ where η is a positive constant. This has the desired effect of giving a lower energy (thus encouraging a higher probability) when x_i and y_i have the same sign and a higher energy when they have the opposite sign.

The remaining cliques comprise pairs of variables $\{x_i, x_j\}$ where i and j are indices of neighbouring pixels. Again, we want the energy to be lower when the pixels have the same sign than when they have the opposite sign, and so we choose an energy given by $-\beta x_i x_j$ where β is a positive constant.

Because a potential function is an arbitrary, nonnegative function over a maximal clique, we can multiply it by any nonnegative functions of subsets of the clique, or

Figure 8.31 An undirected graphical model representing a Markov random field for image de-noising, in which x_i is a binary variable denoting the state of pixel i in the unknown noise-free image, and y_i denotes the corresponding value of pixel i in the observed noisy image.



equivalently we can add the corresponding energies. In this example, this allows us to add an extra term hx_i for each pixel i in the noise-free image. Such a term has the effect of biasing the model towards pixel values that have one particular sign in preference to the other.

The complete energy function for the model then takes the form

$$E(\mathbf{x}, \mathbf{y}) = h \sum_i x_i - \beta \sum_{\{i,j\}} x_i x_j - \eta \sum_i x_i y_i \quad (8.42)$$

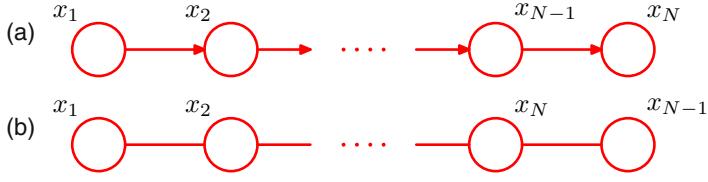
which defines a joint distribution over \mathbf{x} and \mathbf{y} given by

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \exp\{-E(\mathbf{x}, \mathbf{y})\}. \quad (8.43)$$

We now fix the elements of \mathbf{y} to the observed values given by the pixels of the noisy image, which implicitly defines a conditional distribution $p(\mathbf{x}|\mathbf{y})$ over noise-free images. This is an example of the *Ising model*, which has been widely studied in statistical physics. For the purposes of image restoration, we wish to find an image \mathbf{x} having a high probability (ideally the maximum probability). To do this we shall use a simple iterative technique called *iterated conditional modes*, or *ICM* (Kittler and Föglein, 1984), which is simply an application of coordinate-wise gradient ascent. The idea is first to initialize the variables $\{x_i\}$, which we do by simply setting $x_i = y_i$ for all i . Then we take one node x_j at a time and we evaluate the total energy for the two possible states $x_j = +1$ and $x_j = -1$, keeping all other node variables fixed, and set x_j to whichever state has the lower energy. This will either leave the probability unchanged, if x_j is unchanged, or will increase it. Because only one variable is changed, this is a simple local computation that can be performed efficiently. We then repeat the update for another site, and so on, until some suitable stopping criterion is satisfied. The nodes may be updated in a systematic way, for instance by repeatedly raster scanning through the image, or by choosing nodes at random.

If we have a sequence of updates in which every site is visited at least once, and in which no changes to the variables are made, then by definition the algorithm

Figure 8.32 (a) Example of a directed graph. (b) The equivalent undirected graph.



will have converged to a local maximum of the probability. This need not, however, correspond to the global maximum.

For the purposes of this simple illustration, we have fixed the parameters to be $\beta = 1.0$, $\eta = 2.1$ and $h = 0$. Note that leaving $h = 0$ simply means that the prior probabilities of the two states of x_i are equal. Starting with the observed noisy image as the initial configuration, we run ICM until convergence, leading to the de-noised image shown in the lower left panel of Figure 8.30. Note that if we set $\beta = 0$, which effectively removes the links between neighbouring pixels, then the global most probable solution is given by $x_i = y_i$ for all i , corresponding to the observed noisy image.

Later we shall discuss a more effective algorithm for finding high probability solutions called the max-product algorithm, which typically leads to better solutions, although this is still not guaranteed to find the global maximum of the posterior distribution. However, for certain classes of model, including the one given by (8.42), there exist efficient algorithms based on *graph cuts* that are guaranteed to find the global maximum (Greig *et al.*, 1989; Boykov *et al.*, 2001; Kolmogorov and Zabih, 2004). The lower right panel of Figure 8.30 shows the result of applying a graph-cut algorithm to the de-noising problem.

8.3.4 Relation to directed graphs

We have introduced two graphical frameworks for representing probability distributions, corresponding to directed and undirected graphs, and it is instructive to discuss the relation between these. Consider first the problem of taking a model that is specified using a directed graph and trying to convert it to an undirected graph. In some cases this is straightforward, as in the simple example in Figure 8.32. Here the joint distribution for the directed graph is given as a product of conditionals in the form

$$p(\mathbf{x}) = p(x_1)p(x_2|x_1)p(x_3|x_2)\cdots p(x_N|x_{N-1}). \quad (8.44)$$

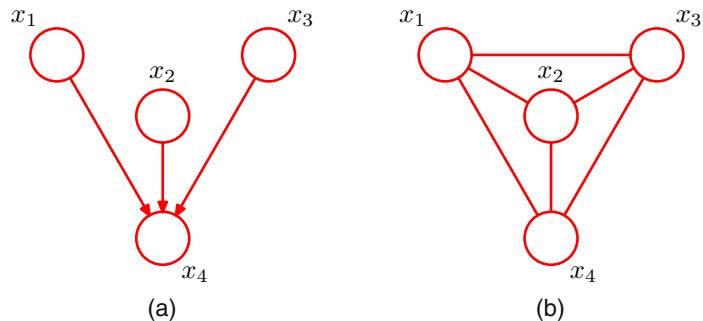
Now let us convert this to an undirected graph representation, as shown in Figure 8.32. In the undirected graph, the maximal cliques are simply the pairs of neighbouring nodes, and so from (8.39) we wish to write the joint distribution in the form

$$p(\mathbf{x}) = \frac{1}{Z} \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N). \quad (8.45)$$

Exercise 8.14

Section 8.4

Figure 8.33 Example of a simple directed graph (a) and the corresponding moral graph (b).



This is easily done by identifying

$$\begin{aligned}\psi_{1,2}(x_1, x_2) &= p(x_1)p(x_2|x_1) \\ \psi_{2,3}(x_2, x_3) &= p(x_3|x_2) \\ &\vdots \\ \psi_{N-1,N}(x_{N-1}, x_N) &= p(x_N|x_{N-1})\end{aligned}$$

where we have absorbed the marginal $p(x_1)$ for the first node into the first potential function. Note that in this case, the partition function $Z = 1$.

Let us consider how to generalize this construction, so that we can convert any distribution specified by a factorization over a directed graph into one specified by a factorization over an undirected graph. This can be achieved if the clique potentials of the undirected graph are given by the conditional distributions of the directed graph. In order for this to be valid, we must ensure that the set of variables that appears in each of the conditional distributions is a member of at least one clique of the undirected graph. For nodes on the directed graph having just one parent, this is achieved simply by replacing the directed link with an undirected link. However, for nodes in the directed graph having more than one parent, this is not sufficient. These are nodes that have ‘head-to-head’ paths encountered in our discussion of conditional independence. Consider a simple directed graph over 4 nodes shown in Figure 8.33. The joint distribution for the directed graph takes the form

$$p(\mathbf{x}) = p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3). \quad (8.46)$$

We see that the factor $p(x_4|x_1, x_2, x_3)$ involves the four variables x_1, x_2, x_3 , and x_4 , and so these must all belong to a single clique if this conditional distribution is to be absorbed into a clique potential. To ensure this, we add extra links between all pairs of parents of the node x_4 . Anachronistically, this process of ‘marrying the parents’ has become known as *moralization*, and the resulting undirected graph, after dropping the arrows, is called the *moral graph*. It is important to observe that the moral graph in this example is fully connected and so exhibits no conditional independence properties, in contrast to the original directed graph.

Thus in general to convert a directed graph into an undirected graph, we first add additional undirected links between all pairs of parents for each node in the graph and

then drop the arrows on the original links to give the moral graph. Then we initialize all of the clique potentials of the moral graph to 1. We then take each conditional distribution factor in the original directed graph and multiply it into one of the clique potentials. There will always exist at least one maximal clique that contains all of the variables in the factor as a result of the moralization step. Note that in all cases the partition function is given by $Z = 1$.

The process of converting a directed graph into an undirected graph plays an important role in exact inference techniques such as the *junction tree algorithm*. Converting from an undirected to a directed representation is much less common and in general presents problems due to the normalization constraints.

We saw that in going from a directed to an undirected representation we had to discard some conditional independence properties from the graph. Of course, we could always trivially convert any distribution over a directed graph into one over an undirected graph by simply using a fully connected undirected graph. This would, however, discard all conditional independence properties and so would be vacuous. The process of moralization adds the fewest extra links and so retains the maximum number of independence properties.

We have seen that the procedure for determining the conditional independence properties is different between directed and undirected graphs. It turns out that the two types of graph can express different conditional independence properties, and it is worth exploring this issue in more detail. To do so, we return to the view of a specific (directed or undirected) graph as a filter, so that the set of all possible distributions over the given variables could be reduced to a subset that respects the conditional independencies implied by the graph. A graph is said to be a *D map* (for ‘dependency map’) of a distribution if every conditional independence statement satisfied by the distribution is reflected in the graph. Thus a completely disconnected graph (no links) will be a trivial D map for any distribution.

Alternatively, we can consider a specific distribution and ask which graphs have the appropriate conditional independence properties. If every conditional independence statement implied by a graph is satisfied by a specific distribution, then the graph is said to be an *I map* (for ‘independence map’) of that distribution. Clearly a fully connected graph will be a trivial I map for any distribution.

If it is the case that every conditional independence property of the distribution is reflected in the graph, and vice versa, then the graph is said to be a *perfect map* for

Figure 8.34 Venn diagram illustrating the set of all distributions P over a given set of variables, together with the set of distributions D that can be represented as a perfect map using a directed graph, and the set U that can be represented as a perfect map using an undirected graph.

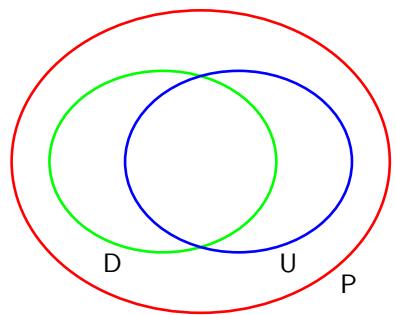
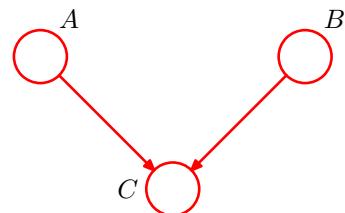


Figure 8.35 A directed graph whose conditional independence properties cannot be expressed using an undirected graph over the same three variables.



that distribution. A perfect map is therefore both an I map and a D map.

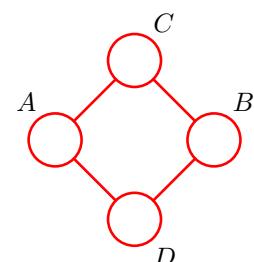
Consider the set of distributions such that for each distribution there exists a directed graph that is a perfect map. This set is distinct from the set of distributions such that for each distribution there exists an undirected graph that is a perfect map. In addition there are distributions for which neither directed nor undirected graphs offer a perfect map. This is illustrated as a Venn diagram in Figure 8.34.

Figure 8.35 shows an example of a directed graph that is a perfect map for a distribution satisfying the conditional independence properties $A \perp\!\!\!\perp B | \emptyset$ and $A \not\perp\!\!\!\perp B | C$. There is no corresponding undirected graph over the same three variables that is a perfect map.

Conversely, consider the undirected graph over four variables shown in Figure 8.36. This graph exhibits the properties $A \not\perp\!\!\!\perp B | \emptyset$, $C \perp\!\!\!\perp D | A \cup B$ and $A \perp\!\!\!\perp B | C \cup D$. There is no directed graph over four variables that implies the same set of conditional independence properties.

The graphical framework can be extended in a consistent way to graphs that include both directed and undirected links. These are called *chain graphs* (Lauritzen and Wermuth, 1989; Frydenberg, 1990), and contain the directed and undirected graphs considered so far as special cases. Although such graphs can represent a broader class of distributions than either directed or undirected alone, there remain distributions for which even a chain graph cannot provide a perfect map. Chain graphs are not discussed further in this book.

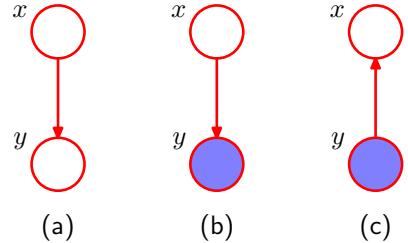
Figure 8.36 An undirected graph whose conditional independence properties cannot be expressed in terms of a directed graph over the same variables.



8.4. Inference in Graphical Models

We turn now to the problem of inference in graphical models, in which some of the nodes in a graph are clamped to observed values, and we wish to compute the posterior distributions of one or more subsets of other nodes. As we shall see, we can exploit the graphical structure both to find efficient algorithms for inference, and

Figure 8.37 A graphical representation of Bayes' theorem.
See the text for details.



to make the structure of those algorithms transparent. Specifically, we shall see that many algorithms can be expressed in terms of the propagation of local *messages* around the graph. In this section, we shall focus primarily on techniques for exact inference, and in Chapter 10 we shall consider a number of approximate inference algorithms.

To start with, let us consider the graphical interpretation of Bayes' theorem. Suppose we decompose the joint distribution $p(x, y)$ over two variables x and y into a product of factors in the form $p(x, y) = p(x)p(y|x)$. This can be represented by the directed graph shown in Figure 8.37(a). Now suppose we observe the value of y , as indicated by the shaded node in Figure 8.37(b). We can view the marginal distribution $p(x)$ as a prior over the latent variable x , and our goal is to infer the corresponding posterior distribution over x . Using the sum and product rules of probability we can evaluate

$$p(y) = \sum_{x'} p(y|x')p(x') \quad (8.47)$$

which can then be used in Bayes' theorem to calculate

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}. \quad (8.48)$$

Thus the joint distribution is now expressed in terms of $p(y)$ and $p(x|y)$. From a graphical perspective, the joint distribution $p(x, y)$ is now represented by the graph shown in Figure 8.37(c), in which the direction of the arrow is reversed. This is the simplest example of an inference problem for a graphical model.

8.4.1 Inference on a chain

Now consider a more complex problem involving the chain of nodes of the form shown in Figure 8.32. This example will lay the foundation for a discussion of exact inference in more general graphs later in this section.

Specifically, we shall consider the undirected graph in Figure 8.32(b). We have already seen that the directed chain can be transformed into an equivalent undirected chain. Because the directed graph does not have any nodes with more than one parent, this does not require the addition of any extra links, and the directed and undirected versions of this graph express exactly the same set of conditional independence statements.

The joint distribution for this graph takes the form

$$p(\mathbf{x}) = \frac{1}{Z} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N). \quad (8.49)$$

We shall consider the specific case in which the N nodes represent discrete variables each having K states, in which case each potential function $\psi_{n-1,n}(x_{n-1}, x_n)$ comprises an $K \times K$ table, and so the joint distribution has $(N - 1)K^2$ parameters.

Let us consider the inference problem of finding the marginal distribution $p(x_n)$ for a specific node x_n that is part way along the chain. Note that, for the moment, there are no observed nodes. By definition, the required marginal is obtained by summing the joint distribution over all variables except x_n , so that

$$p(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} p(\mathbf{x}). \quad (8.50)$$

In a naive implementation, we would first evaluate the joint distribution and then perform the summations explicitly. The joint distribution can be represented as a set of numbers, one for each possible value for \mathbf{x} . Because there are N variables each with K states, there are K^N values for \mathbf{x} and so evaluation and storage of the joint distribution, as well as marginalization to obtain $p(x_n)$, all involve storage and computation that scale exponentially with the length N of the chain.

We can, however, obtain a much more efficient algorithm by exploiting the conditional independence properties of the graphical model. If we substitute the factorized expression (8.49) for the joint distribution into (8.50), then we can rearrange the order of the summations and the multiplications to allow the required marginal to be evaluated much more efficiently. Consider for instance the summation over x_N . The potential $\psi_{N-1,N}(x_{N-1}, x_N)$ is the only one that depends on x_N , and so we can perform the summation

$$\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \quad (8.51)$$

first to give a function of x_{N-1} . We can then use this to perform the summation over x_{N-1} , which will involve only this new function together with the potential $\psi_{N-2,N-1}(x_{N-2}, x_{N-1})$, because this is the only other place that x_{N-1} appears. Similarly, the summation over x_1 involves only the potential $\psi_{1,2}(x_1, x_2)$ and so can be performed separately to give a function of x_2 , and so on. Because each summation effectively removes a variable from the distribution, this can be viewed as the removal of a node from the graph.

If we group the potentials and summations together in this way, we can express

the desired marginal in the form

$$p(x_n) = \frac{1}{Z} \underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \cdots \left[\sum_{x_2} \psi_{2,3}(x_2, x_3) \left[\sum_{x_1} \psi_{1,2}(x_1, x_2) \right] \right] \cdots \right]}_{\mu_\alpha(x_n)} \\ \underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right] \cdots \right]}_{\mu_\beta(x_n)}. \quad (8.52)$$

The reader is encouraged to study this re-ordering carefully as the underlying idea forms the basis for the later discussion of the general sum-product algorithm. Here the key concept that we are exploiting is that multiplication is distributive over addition, so that

$$ab + ac = a(b + c) \quad (8.53)$$

in which the left-hand side involves three arithmetic operations whereas the right-hand side reduces this to two operations.

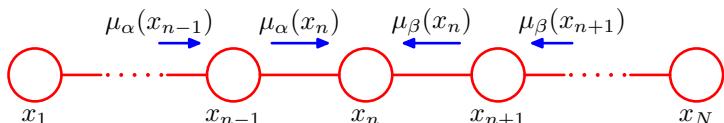
Let us work out the computational cost of evaluating the required marginal using this re-ordered expression. We have to perform $N - 1$ summations each of which is over K states and each of which involves a function of two variables. For instance, the summation over x_1 involves only the function $\psi_{1,2}(x_1, x_2)$, which is a table of $K \times K$ numbers. We have to sum this table over x_1 for each value of x_2 and so this has $O(K^2)$ cost. The resulting vector of K numbers is multiplied by the matrix of numbers $\psi_{2,3}(x_2, x_3)$ and so is again $O(K^2)$. Because there are $N - 1$ summations and multiplications of this kind, the total cost of evaluating the marginal $p(x_n)$ is $O(NK^2)$. This is linear in the length of the chain, in contrast to the exponential cost of a naive approach. We have therefore been able to exploit the many conditional independence properties of this simple graph in order to obtain an efficient calculation. If the graph had been fully connected, there would have been no conditional independence properties, and we would have been forced to work directly with the full joint distribution.

We now give a powerful interpretation of this calculation in terms of the passing of local *messages* around on the graph. From (8.52) we see that the expression for the marginal $p(x_n)$ decomposes into the product of two factors times the normalization constant

$$p(x_n) = \frac{1}{Z} \mu_\alpha(x_n) \mu_\beta(x_n). \quad (8.54)$$

We shall interpret $\mu_\alpha(x_n)$ as a message passed forwards along the chain from node x_{n-1} to node x_n . Similarly, $\mu_\beta(x_n)$ can be viewed as a message passed backwards

Figure 8.38 The marginal distribution $p(x_n)$ for a node x_n along the chain is obtained by multiplying the two messages $\mu_\alpha(x_n)$ and $\mu_\beta(x_n)$, and then normalizing. These messages can themselves be evaluated recursively by passing messages from both ends of the chain towards node x_n .



along the chain to node x_n from node x_{n+1} . Note that each of the messages comprises a set of K values, one for each choice of x_n , and so the product of two messages should be interpreted as the point-wise multiplication of the elements of the two messages to give another set of K values.

The message $\mu_\alpha(x_n)$ can be evaluated recursively because

$$\begin{aligned}\mu_\alpha(x_n) &= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \left[\sum_{x_{n-2}} \dots \right] \\ &= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \mu_\alpha(x_{n-1}).\end{aligned}\quad (8.55)$$

We therefore first evaluate

$$\mu_\alpha(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2) \quad (8.56)$$

and then apply (8.55) repeatedly until we reach the desired node. Note carefully the structure of the message passing equation. The outgoing message $\mu_\alpha(x_n)$ in (8.55) is obtained by multiplying the incoming message $\mu_\alpha(x_{n-1})$ by the local potential involving the node variable and the outgoing variable and then summing over the node variable.

Similarly, the message $\mu_\beta(x_n)$ can be evaluated recursively by starting with node x_N and using

$$\begin{aligned}\mu_\beta(x_n) &= \sum_{x_{n+1}} \psi_{n+1,n}(x_{n+1}, x_n) \left[\sum_{x_{n+2}} \dots \right] \\ &= \sum_{x_{n+1}} \psi_{n+1,n}(x_{n+1}, x_n) \mu_\beta(x_{n+1}).\end{aligned}\quad (8.57)$$

This recursive message passing is illustrated in Figure 8.38. The normalization constant Z is easily evaluated by summing the right-hand side of (8.54) over all states of x_n , an operation that requires only $O(K)$ computation.

Graphs of the form shown in Figure 8.38 are called *Markov chains*, and the corresponding message passing equations represent an example of the *Chapman-Kolmogorov* equations for Markov processes (Papoulis, 1984).

Now suppose we wish to evaluate the marginals $p(x_n)$ for every node $n \in \{1, \dots, N\}$ in the chain. Simply applying the above procedure separately for each node will have computational cost that is $O(N^2 M^2)$. However, such an approach would be very wasteful of computation. For instance, to find $p(x_1)$ we need to propagate a message $\mu_\beta(\cdot)$ from node x_N back to node x_2 . Similarly, to evaluate $p(x_2)$ we need to propagate a messages $\mu_\beta(\cdot)$ from node x_N back to node x_3 . This will involve much duplicated computation because most of the messages will be identical in the two cases.

Suppose instead we first launch a message $\mu_\beta(x_{N-1})$ starting from node x_N and propagate corresponding messages all the way back to node x_1 , and suppose we similarly launch a message $\mu_\alpha(x_2)$ starting from node x_1 and propagate the corresponding messages all the way forward to node x_N . Provided we store all of the intermediate messages along the way, then any node can evaluate its marginal simply by applying (8.54). The computational cost is only twice that for finding the marginal of a single node, rather than N times as much. Observe that a message has passed once in each direction across each link in the graph. Note also that the normalization constant Z need be evaluated only once, using any convenient node.

If some of the nodes in the graph are observed, then the corresponding variables are simply clamped to their observed values and there is no summation. To see this, note that the effect of clamping a variable x_n to an observed value \hat{x}_n can be expressed by multiplying the joint distribution by (one or more copies of) an additional function $I(x_n, \hat{x}_n)$, which takes the value 1 when $x_n = \hat{x}_n$ and the value 0 otherwise. One such function can then be absorbed into each of the potentials that contain x_n . Summations over x_n then contain only one term in which $x_n = \hat{x}_n$.

Now suppose we wish to calculate the joint distribution $p(x_{n-1}, x_n)$ for two neighbouring nodes on the chain. This is similar to the evaluation of the marginal for a single node, except that there are now two variables that are not summed out. A few moments thought will show that the required joint distribution can be written in the form

$$p(x_{n-1}, x_n) = \frac{1}{Z} \mu_\alpha(x_{n-1}) \psi_{n-1,n}(x_{n-1}, x_n) \mu_\beta(x_n). \quad (8.58)$$

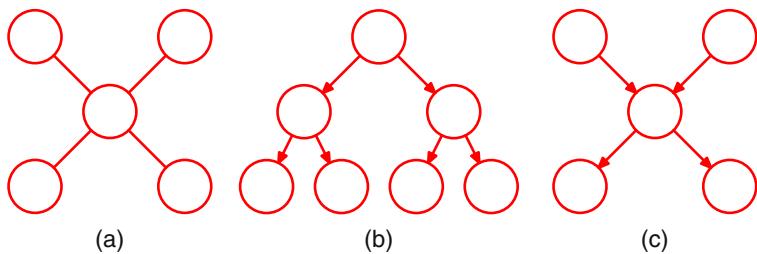
Thus we can obtain the joint distributions over all of the sets of variables in each of the potentials directly once we have completed the message passing required to obtain the marginals.

This is a useful result because in practice we may wish to use parametric forms for the clique potentials, or equivalently for the conditional distributions if we started from a directed graph. In order to learn the parameters of these potentials in situations where not all of the variables are observed, we can employ the *EM algorithm*, and it turns out that the local joint distributions of the cliques, conditioned on any observed data, is precisely what is needed in the E step. We shall consider some examples of this in detail in Chapter 13.

8.4.2 Trees

We have seen that exact inference on a graph comprising a chain of nodes can be performed efficiently in time that is linear in the number of nodes, using an algorithm

Figure 8.39 Examples of tree-structured graphs, showing (a) an undirected tree, (b) a directed tree, and (c) a directed polytree.



that can be interpreted in terms of messages passed along the chain. More generally, inference can be performed efficiently using local message passing on a broader class of graphs called *trees*. In particular, we shall shortly generalize the message passing formalism derived above for chains to give the *sum-product* algorithm, which provides an efficient framework for exact inference in tree-structured graphs.

In the case of an undirected graph, a tree is defined as a graph in which there is one, and only one, path between any pair of nodes. Such graphs therefore do not have loops. In the case of directed graphs, a tree is defined such that there is a single node, called the *root*, which has no parents, and all other nodes have one parent. If we convert a directed tree into an undirected graph, we see that the moralization step will not add any links as all nodes have at most one parent, and as a consequence the corresponding moralized graph will be an undirected tree. Examples of undirected and directed trees are shown in Figure 8.39(a) and 8.39(b). Note that a distribution represented as a directed tree can easily be converted into one represented by an undirected tree, and vice versa.

If there are nodes in a directed graph that have more than one parent, but there is still only one path (ignoring the direction of the arrows) between any two nodes, then the graph is called a *polytree*, as illustrated in Figure 8.39(c). Such a graph will have more than one node with the property of having no parents, and furthermore, the corresponding moralized undirected graph will have loops.

8.4.3 Factor graphs

The sum-product algorithm that we derive in the next section is applicable to undirected and directed trees and to polytrees. It can be cast in a particularly simple and general form if we first introduce a new graphical construction called a *factor graph* (Frey, 1998; Kschischang *et al.*, 2001).

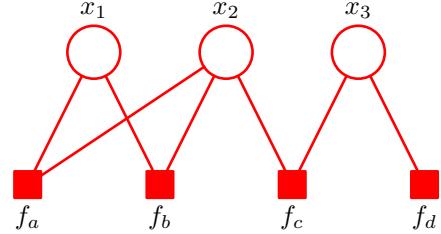
Both directed and undirected graphs allow a global function of several variables to be expressed as a product of factors over subsets of those variables. Factor graphs make this decomposition explicit by introducing additional nodes for the factors themselves in addition to the nodes representing the variables. They also allow us to be more explicit about the details of the factorization, as we shall see.

Let us write the joint distribution over a set of variables in the form of a product of factors

$$p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s) \quad (8.59)$$

where \mathbf{x}_s denotes a subset of the variables. For convenience, we shall denote the

Figure 8.40 Example of a factor graph, which corresponds to the factorization (8.60).



individual variables by x_i , however, as in earlier discussions, these can comprise groups of variables (such as vectors or matrices). Each factor f_s is a function of a corresponding set of variables \mathbf{x}_s .

Directed graphs, whose factorization is defined by (8.5), represent special cases of (8.59) in which the factors $f_s(\mathbf{x}_s)$ are local conditional distributions. Similarly, undirected graphs, given by (8.39), are a special case in which the factors are potential functions over the maximal cliques (the normalizing coefficient $1/Z$ can be viewed as a factor defined over the empty set of variables).

In a factor graph, there is a node (depicted as usual by a circle) for every variable in the distribution, as was the case for directed and undirected graphs. There are also additional nodes (depicted by small squares) for each factor $f_s(\mathbf{x}_s)$ in the joint distribution. Finally, there are undirected links connecting each factor node to all of the variables nodes on which that factor depends. Consider, for example, a distribution that is expressed in terms of the factorization

$$p(\mathbf{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3). \quad (8.60)$$

This can be expressed by the factor graph shown in Figure 8.40. Note that there are two factors $f_a(x_1, x_2)$ and $f_b(x_1, x_2)$ that are defined over the same set of variables. In an undirected graph, the product of two such factors would simply be lumped together into the same clique potential. Similarly, $f_c(x_2, x_3)$ and $f_d(x_3)$ could be combined into a single potential over x_2 and x_3 . The factor graph, however, keeps such factors explicit and so is able to convey more detailed information about the underlying factorization.

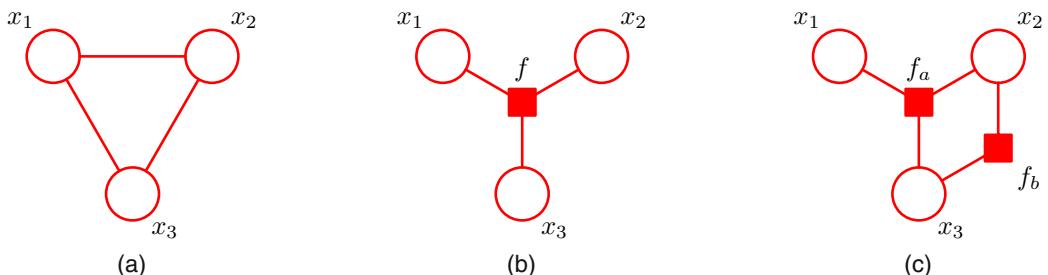


Figure 8.41 (a) An undirected graph with a single clique potential $\psi(x_1, x_2, x_3)$. (b) A factor graph with factor $f(x_1, x_2, x_3) = \psi(x_1, x_2, x_3)$ representing the same distribution as the undirected graph. (c) A different factor graph representing the same distribution, whose factors satisfy $f_a(x_1, x_2, x_3)f_b(x_1, x_2) = \psi(x_1, x_2, x_3)$.

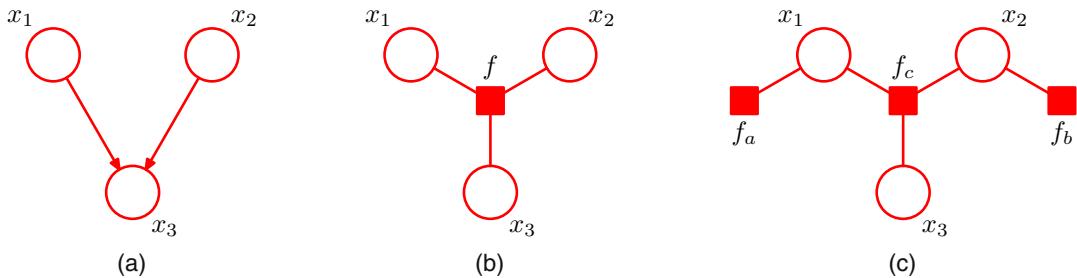


Figure 8.42 (a) A directed graph with the factorization $p(x_1)p(x_2)p(x_3|x_1, x_2)$. (b) A factor graph representing the same distribution as the directed graph, whose factor satisfies $f(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3|x_1, x_2)$. (c) A different factor graph representing the same distribution with factors $f_a(x_1) = p(x_1)$, $f_b(x_2) = p(x_2)$ and $f_c(x_1, x_2, x_3) = p(x_3|x_1, x_2)$.

Factor graphs are said to be *bipartite* because they consist of two distinct kinds of nodes, and all links go between nodes of opposite type. In general, factor graphs can therefore always be drawn as two rows of nodes (variable nodes at the top and factor nodes at the bottom) with links between the rows, as shown in the example in Figure 8.40. In some situations, however, other ways of laying out the graph may be more intuitive, for example when the factor graph is derived from a directed or undirected graph, as we shall see.

If we are given a distribution that is expressed in terms of an undirected graph, then we can readily convert it to a factor graph. To do this, we create variable nodes corresponding to the nodes in the original undirected graph, and then create additional factor nodes corresponding to the maximal cliques x_s . The factors $f_s(x_s)$ are then set equal to the clique potentials. Note that there may be several different factor graphs that correspond to the same undirected graph. These concepts are illustrated in Figure 8.41.

Similarly, to convert a directed graph to a factor graph, we simply create variable nodes in the factor graph corresponding to the nodes of the directed graph, and then create factor nodes corresponding to the conditional distributions, and then finally add the appropriate links. Again, there can be multiple factor graphs all of which correspond to the same directed graph. The conversion of a directed graph to a factor graph is illustrated in Figure 8.42.

We have already noted the importance of tree-structured graphs for performing efficient inference. If we take a directed or undirected tree and convert it into a factor graph, then the result will again be a tree (in other words, the factor graph will have no loops, and there will be one and only one path connecting any two nodes). In the case of a directed polytree, conversion to an undirected graph results in loops due to the moralization step, whereas conversion to a factor graph again results in a tree, as illustrated in Figure 8.43. In fact, local cycles in a directed graph due to links connecting parents of a node can be removed on conversion to a factor graph by defining the appropriate factor function, as shown in Figure 8.44.

We have seen that multiple different factor graphs can represent the same directed or undirected graph. This allows factor graphs to be more specific about the

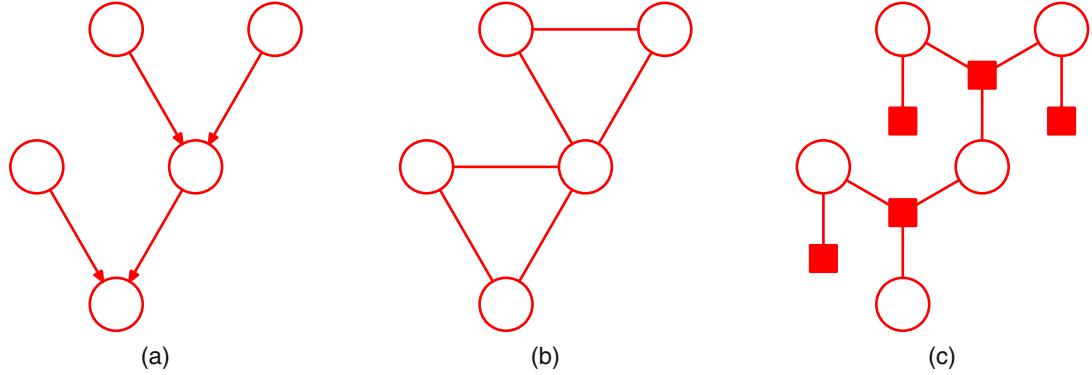


Figure 8.43 (a) A directed polytree. (b) The result of converting the polytree into an undirected graph showing the creation of loops. (c) The result of converting the polytree into a factor graph, which retains the tree structure.

precise form of the factorization. Figure 8.45 shows an example of a fully connected undirected graph along with two different factor graphs. In (b), the joint distribution is given by a general form $p(\mathbf{x}) = f(x_1, x_2, x_3)$, whereas in (c), it is given by the more specific factorization $p(\mathbf{x}) = f_a(x_1, x_2)f_b(x_1, x_3)f_c(x_2, x_3)$. It should be emphasized that the factorization in (c) does not correspond to any conditional independence properties.

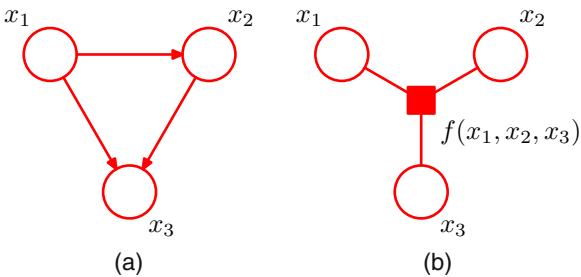
8.4.4 The sum-product algorithm

We shall now make use of the factor graph framework to derive a powerful class of efficient, exact inference algorithms that are applicable to tree-structured graphs. Here we shall focus on the problem of evaluating local marginals over nodes or subsets of nodes, which will lead us to the *sum-product* algorithm. Later we shall modify the technique to allow the most probable state to be found, giving rise to the *max-sum* algorithm.

Also we shall suppose that all of the variables in the model are discrete, and so marginalization corresponds to performing sums. The framework, however, is equally applicable to linear-Gaussian models in which case marginalization involves integration, and we shall consider an example of this in detail when we discuss linear dynamical systems.

Section 13.3

Figure 8.44 (a) A fragment of a directed graph having a local cycle. (b) Conversion to a fragment of a factor graph having a tree structure, in which $f(x_1, x_2, x_3) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)$.



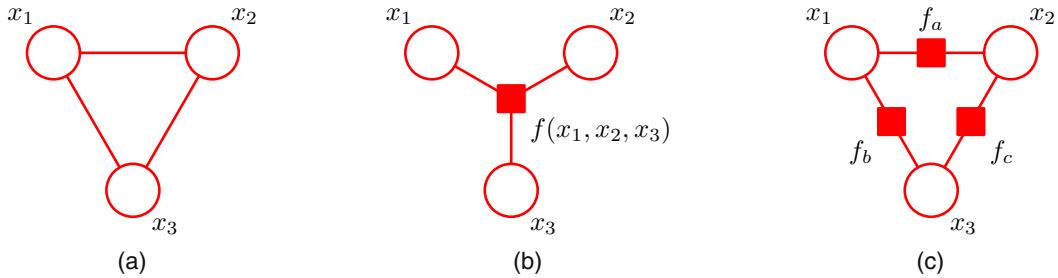


Figure 8.45 (a) A fully connected undirected graph. (b) and (c) Two factor graphs each of which corresponds to the undirected graph in (a).

There is an algorithm for exact inference on directed graphs without loops known as *belief propagation* (Pearl, 1988; Lauritzen and Spiegelhalter, 1988), and is equivalent to a special case of the sum-product algorithm. Here we shall consider only the sum-product algorithm because it is simpler to derive and to apply, as well as being more general.

We shall assume that the original graph is an undirected tree or a directed tree or polytree, so that the corresponding factor graph has a tree structure. We first convert the original graph into a factor graph so that we can deal with both directed and undirected models using the same framework. Our goal is to exploit the structure of the graph to achieve two things: (i) to obtain an efficient, exact inference algorithm for finding marginals; (ii) in situations where several marginals are required to allow computations to be shared efficiently.

We begin by considering the problem of finding the marginal $p(x)$ for particular variable node x . For the moment, we shall suppose that all of the variables are hidden. Later we shall see how to modify the algorithm to incorporate evidence corresponding to observed variables. By definition, the marginal is obtained by summing the joint distribution over all variables except x so that

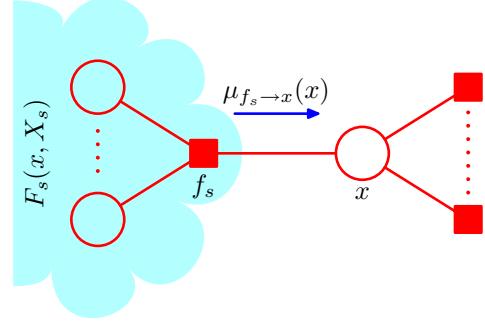
$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x}) \quad (8.61)$$

where $\mathbf{x} \setminus x$ denotes the set of variables in \mathbf{x} with variable x omitted. The idea is to substitute for $p(\mathbf{x})$ using the factor graph expression (8.59) and then interchange summations and products in order to obtain an efficient algorithm. Consider the fragment of graph shown in Figure 8.46 in which we see that the tree structure of the graph allows us to partition the factors in the joint distribution into groups, with one group associated with each of the factor nodes that is a neighbour of the variable node x . We see that the joint distribution can be written as a product of the form

$$p(\mathbf{x}) = \prod_{s \in \text{ne}(x)} F_s(x, X_s) \quad (8.62)$$

$\text{ne}(x)$ denotes the set of factor nodes that are neighbours of x , and X_s denotes the set of all variables in the subtree connected to the variable node x via the factor node

Figure 8.46 A fragment of a factor graph illustrating the evaluation of the marginal $p(x)$.



f_s , and $F_s(x, X_s)$ represents the product of all the factors in the group associated with factor f_s .

Substituting (8.62) into (8.61) and interchanging the sums and products, we obtain

$$\begin{aligned} p(x) &= \prod_{s \in \text{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right] \\ &= \prod_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x). \end{aligned} \quad (8.63)$$

Here we have introduced a set of functions $\mu_{f_s \rightarrow x}(x)$, defined by

$$\mu_{f_s \rightarrow x}(x) \equiv \sum_{X_s} F_s(x, X_s) \quad (8.64)$$

which can be viewed as *messages* from the factor nodes f_s to the variable node x . We see that the required marginal $p(x)$ is given by the product of all the incoming messages arriving at node x .

In order to evaluate these messages, we again turn to Figure 8.46 and note that each factor $F_s(x, X_s)$ is described by a factor (sub-)graph and so can itself be factorized. In particular, we can write

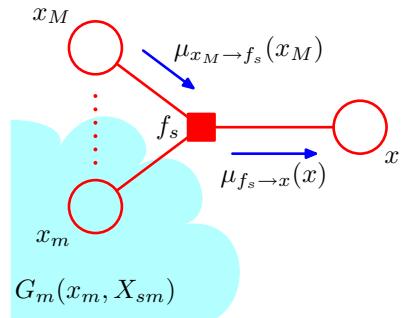
$$F_s(x, X_s) = f_s(x, x_1, \dots, x_M) G_1(x_1, X_{s1}) \dots G_M(x_M, X_{sM}) \quad (8.65)$$

where, for convenience, we have denoted the variables associated with factor f_s , in addition to x , by x_1, \dots, x_M . This factorization is illustrated in Figure 8.47. Note that the set of variables $\{x, x_1, \dots, x_M\}$ is the set of variables on which the factor f_s depends, and so it can also be denoted \mathbf{x}_s , using the notation of (8.59).

Substituting (8.65) into (8.64) we obtain

$$\begin{aligned} \mu_{f_s \rightarrow x}(x) &= \sum_{x_1} \dots \sum_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m \in \text{ne}(f_s) \setminus x} \left[\sum_{X_{sm}} G_m(x_m, X_{sm}) \right] \\ &= \sum_{x_1} \dots \sum_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \rightarrow f_s}(x_m) \end{aligned} \quad (8.66)$$

Figure 8.47 Illustration of the factorization of the subgraph associated with factor node f_s .



where $\text{ne}(f_s)$ denotes the set of variable nodes that are neighbours of the factor node f_s , and $\text{ne}(f_s) \setminus x$ denotes the same set but with node x removed. Here we have defined the following messages from variable nodes to factor nodes

$$\mu_{x_m \rightarrow f_s}(x_m) \equiv \sum_{X_{sm}} G_m(x_m, X_{sm}). \quad (8.67)$$

We have therefore introduced two distinct kinds of message, those that go from factor nodes to variable nodes denoted $\mu_{f \rightarrow x}(x)$, and those that go from variable nodes to factor nodes denoted $\mu_{x \rightarrow f}(x)$. In each case, we see that messages passed along a link are always a function of the variable associated with the variable node that link connects to.

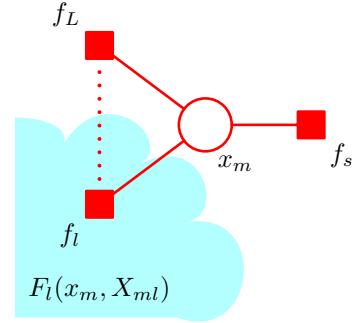
The result (8.66) says that to evaluate the message sent by a factor node to a variable node along the link connecting them, take the product of the incoming messages along all other links coming into the factor node, multiply by the factor associated with that node, and then marginalize over all of the variables associated with the incoming messages. This is illustrated in Figure 8.47. It is important to note that a factor node can send a message to a variable node once it has received incoming messages from all other neighbouring variable nodes.

Finally, we derive an expression for evaluating the messages from variable nodes to factor nodes, again by making use of the (sub-)graph factorization. From Figure 8.48, we see that term $G_m(x_m, X_{sm})$ associated with node x_m is given by a product of terms $F_l(x_m, X_{ml})$ each associated with one of the factor nodes f_l that is linked to node x_m (excluding node f_s), so that

$$G_m(x_m, X_{sm}) = \prod_{l \in \text{ne}(x_m) \setminus f_s} F_l(x_m, X_{ml}) \quad (8.68)$$

where the product is taken over all neighbours of node x_m except for node f_s . Note that each of the factors $F_l(x_m, X_{ml})$ represents a subtree of the original graph of precisely the same kind as introduced in (8.62). Substituting (8.68) into (8.67), we

Figure 8.48 Illustration of the evaluation of the message sent by a variable node to an adjacent factor node.



then obtain

$$\begin{aligned}\mu_{x_m \rightarrow f_s}(x_m) &= \prod_{l \in \text{ne}(x_m) \setminus f_s} \left[\sum_{X_{ml}} F_l(x_m, X_{ml}) \right] \\ &= \prod_{l \in \text{ne}(x_m) \setminus f_s} \mu_{f_l \rightarrow x_m}(x_m)\end{aligned}\quad (8.69)$$

where we have used the definition (8.64) of the messages passed from factor nodes to variable nodes. Thus to evaluate the message sent by a variable node to an adjacent factor node along the connecting link, we simply take the product of the incoming messages along all of the other links. Note that any variable node that has only two neighbours performs no computation but simply passes messages through unchanged. Also, we note that a variable node can send a message to a factor node once it has received incoming messages from all other neighbouring factor nodes.

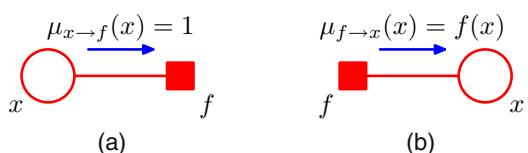
Recall that our goal is to calculate the marginal for variable node x , and that this marginal is given by the product of incoming messages along all of the links arriving at that node. Each of these messages can be computed recursively in terms of other messages. In order to start this recursion, we can view the node x as the root of the tree and begin at the leaf nodes. From the definition (8.69), we see that if a leaf node is a variable node, then the message that it sends along its one and only link is given by

$$\mu_{x \rightarrow f}(x) = 1 \quad (8.70)$$

as illustrated in Figure 8.49(a). Similarly, if the leaf node is a factor node, we see from (8.66) that the message sent should take the form

$$\mu_{f \rightarrow x}(x) = f(x) \quad (8.71)$$

Figure 8.49 The sum-product algorithm begins with messages sent by the leaf nodes, which depend on whether the leaf node is (a) a variable node, or (b) a factor node.



as illustrated in Figure 8.49(b).

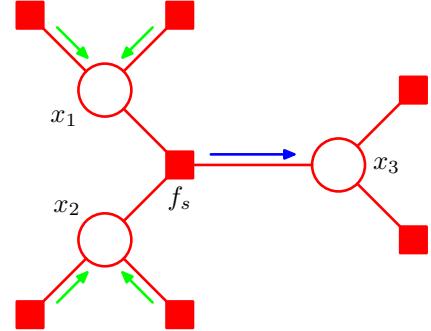
At this point, it is worth pausing to summarize the particular version of the sum-product algorithm obtained so far for evaluating the marginal $p(x)$. We start by viewing the variable node x as the root of the factor graph and initiating messages at the leaves of the graph using (8.70) and (8.71). The message passing steps (8.66) and (8.69) are then applied recursively until messages have been propagated along every link, and the root node has received messages from all of its neighbours. Each node can send a message towards the root once it has received messages from all of its other neighbours. Once the root node has received messages from all of its neighbours, the required marginal can be evaluated using (8.63). We shall illustrate this process shortly.

To see that each node will always receive enough messages to be able to send out a message, we can use a simple inductive argument as follows. Clearly, for a graph comprising a variable root node connected directly to several factor leaf nodes, the algorithm trivially involves sending messages of the form (8.71) directly from the leaves to the root. Now imagine building up a general graph by adding nodes one at a time, and suppose that for some particular graph we have a valid algorithm. When one more (variable or factor) node is added, it can be connected only by a single link because the overall graph must remain a tree, and so the new node will be a leaf node. It therefore sends a message to the node to which it is linked, which in turn will therefore receive all the messages it requires in order to send its own message towards the root, and so again we have a valid algorithm, thereby completing the proof.

Now suppose we wish to find the marginals for every variable node in the graph. This could be done by simply running the above algorithm afresh for each such node. However, this would be very wasteful as many of the required computations would be repeated. We can obtain a much more efficient procedure by ‘overlaying’ these multiple message passing algorithms to obtain the general sum-product algorithm as follows. Arbitrarily pick any (variable or factor) node and designate it as the root. Propagate messages from the leaves to the root as before. At this point, the root node will have received messages from all of its neighbours. It can therefore send out messages to all of its neighbours. These in turn will then have received messages from all of their neighbours and so can send out messages along the links going away from the root, and so on. In this way, messages are passed outwards from the root all the way to the leaves. By now, a message will have passed in both directions across every link in the graph, and every node will have received a message from all of its neighbours. Again a simple inductive argument can be used to verify the validity of this message passing protocol. Because every variable node will have received messages from all of its neighbours, we can readily calculate the marginal distribution for every variable in the graph. The number of messages that have to be computed is given by twice the number of links in the graph and so involves only twice the computation involved in finding a single marginal. By comparison, if we had run the sum-product algorithm separately for each node, the amount of computation would grow quadratically with the size of the graph. Note that this algorithm is in fact independent of which node was designated as the root,

Exercise 8.20

Figure 8.50 The sum-product algorithm can be viewed purely in terms of messages sent out by factor nodes to other factor nodes. In this example, the outgoing message shown by the blue arrow is obtained by taking the product of all the incoming messages shown by green arrows, multiplying by the factor f_s , and marginalizing over the variables x_1 and x_2 .



and indeed the notion of one node having a special status was introduced only as a convenient way to explain the message passing protocol.

Next suppose we wish to find the marginal distributions $p(\mathbf{x}_s)$ associated with the sets of variables belonging to each of the factors. By a similar argument to that used above, it is easy to see that the marginal associated with a factor is given by the product of messages arriving at the factor node and the local factor at that node

$$p(\mathbf{x}_s) = f_s(\mathbf{x}_s) \prod_{i \in \text{ne}(f_s)} \mu_{x_i \rightarrow f_s}(x_i) \quad (8.72)$$

in complete analogy with the marginals at the variable nodes. If the factors are parameterized functions and we wish to learn the values of the parameters using the EM algorithm, then these marginals are precisely the quantities we will need to calculate in the E step, as we shall see in detail when we discuss the hidden Markov model in Chapter 13.

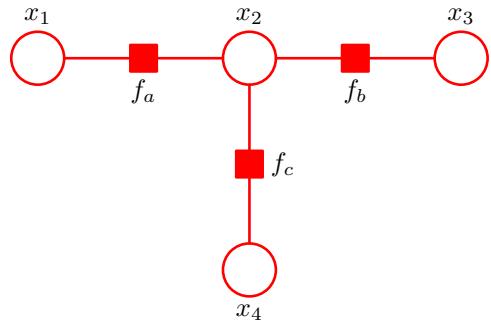
The message sent by a variable node to a factor node, as we have seen, is simply the product of the incoming messages on other links. We can if we wish view the sum-product algorithm in a slightly different form by eliminating messages from variable nodes to factor nodes and simply considering messages that are sent out by factor nodes. This is most easily seen by considering the example in Figure 8.50.

So far, we have rather neglected the issue of normalization. If the factor graph was derived from a directed graph, then the joint distribution is already correctly normalized, and so the marginals obtained by the sum-product algorithm will similarly be normalized correctly. However, if we started from an undirected graph, then in general there will be an unknown normalization coefficient $1/Z$. As with the simple chain example of Figure 8.38, this is easily handled by working with an unnormalized version $\tilde{p}(\mathbf{x})$ of the joint distribution, where $p(\mathbf{x}) = \tilde{p}(\mathbf{x})/Z$. We first run the sum-product algorithm to find the corresponding unnormalized marginals $\tilde{p}(x_i)$. The coefficient $1/Z$ is then easily obtained by normalizing any one of these marginals, and this is computationally efficient because the normalization is done over a single variable rather than over the entire set of variables as would be required to normalize $\tilde{p}(\mathbf{x})$ directly.

At this point, it may be helpful to consider a simple example to illustrate the operation of the sum-product algorithm. Figure 8.51 shows a simple 4-node factor

Exercise 8.21

Figure 8.51 A simple factor graph used to illustrate the sum-product algorithm.



graph whose unnormalized joint distribution is given by

$$\tilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4). \quad (8.73)$$

In order to apply the sum-product algorithm to this graph, let us designate node x_3 as the root, in which case there are two leaf nodes x_1 and x_4 . Starting with the leaf nodes, we then have the following sequence of six messages

$$\mu_{x_1 \rightarrow f_a}(x_1) = 1 \quad (8.74)$$

$$\mu_{f_a \rightarrow x_2}(x_2) = \sum_{x_1} f_a(x_1, x_2) \quad (8.75)$$

$$\mu_{x_4 \rightarrow f_c}(x_4) = 1 \quad (8.76)$$

$$\mu_{f_c \rightarrow x_2}(x_2) = \sum_{x_4} f_c(x_2, x_4) \quad (8.77)$$

$$\mu_{x_2 \rightarrow f_b}(x_2) = \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2) \quad (8.78)$$

$$\mu_{f_b \rightarrow x_3}(x_3) = \sum_{x_2} f_b(x_2, x_3) \mu_{x_2 \rightarrow f_b}. \quad (8.79)$$

The direction of flow of these messages is illustrated in Figure 8.52. Once this message propagation is complete, we can then propagate messages from the root node out to the leaf nodes, and these are given by

$$\mu_{x_3 \rightarrow f_b}(x_3) = 1 \quad (8.80)$$

$$\mu_{f_b \rightarrow x_2}(x_2) = \sum_{x_3} f_b(x_2, x_3) \quad (8.81)$$

$$\mu_{x_2 \rightarrow f_a}(x_2) = \mu_{f_b \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2) \quad (8.82)$$

$$\mu_{f_a \rightarrow x_1}(x_1) = \sum_{x_2} f_a(x_1, x_2) \mu_{x_2 \rightarrow f_a}(x_2) \quad (8.83)$$

$$\mu_{x_2 \rightarrow f_c}(x_2) = \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_b \rightarrow x_2}(x_2) \quad (8.84)$$

$$\mu_{f_c \rightarrow x_4}(x_4) = \sum_{x_2} f_c(x_2, x_4) \mu_{x_2 \rightarrow f_c}(x_2). \quad (8.85)$$

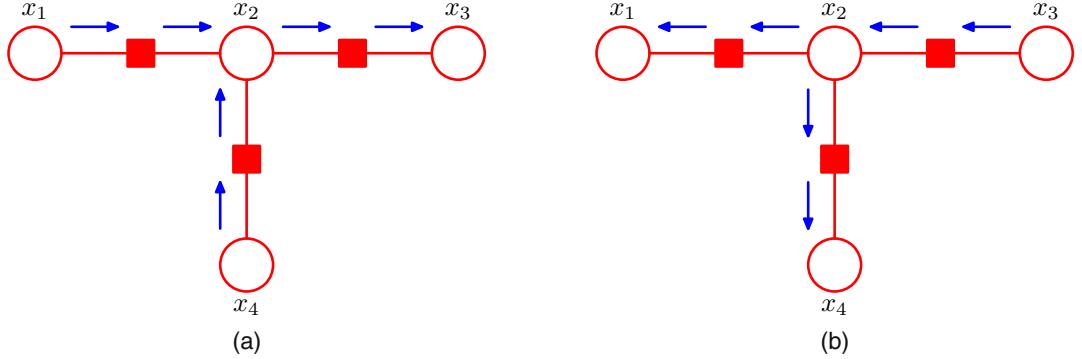


Figure 8.52 Flow of messages for the sum-product algorithm applied to the example graph in Figure 8.51. (a) From the leaf nodes x_1 and x_4 towards the root node x_3 . (b) From the root node towards the leaf nodes.

One message has now passed in each direction across each link, and we can now evaluate the marginals. As a simple check, let us verify that the marginal $p(x_2)$ is given by the correct expression. Using (8.63) and substituting for the messages using the above results, we have

$$\begin{aligned}
 \tilde{p}(x_2) &= \mu_{f_a \rightarrow x_2}(x_2)\mu_{f_b \rightarrow x_2}(x_2)\mu_{f_c \rightarrow x_2}(x_2) \\
 &= \left[\sum_{x_1} f_a(x_1, x_2) \right] \left[\sum_{x_3} f_b(x_2, x_3) \right] \left[\sum_{x_4} f_c(x_2, x_4) \right] \\
 &= \sum_{x_1} \sum_{x_2} \sum_{x_4} f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4) \\
 &= \sum_{x_1} \sum_{x_3} \sum_{x_4} \tilde{p}(\mathbf{x})
 \end{aligned} \tag{8.86}$$

as required.

So far, we have assumed that all of the variables in the graph are hidden. In most practical applications, a subset of the variables will be observed, and we wish to calculate posterior distributions conditioned on these observations. Observed nodes are easily handled within the sum-product algorithm as follows. Suppose we partition \mathbf{x} into hidden variables \mathbf{h} and observed variables \mathbf{v} , and that the observed value of \mathbf{v} is denoted $\hat{\mathbf{v}}$. Then we simply multiply the joint distribution $p(\mathbf{x})$ by $\prod_i I(v_i, \hat{v}_i)$, where $I(v, \hat{v}) = 1$ if $v = \hat{v}$ and $I(v, \hat{v}) = 0$ otherwise. This product corresponds to $p(\mathbf{h}, \mathbf{v} = \hat{\mathbf{v}})$ and hence is an unnormalized version of $p(\mathbf{h}|\mathbf{v} = \hat{\mathbf{v}})$. By running the sum-product algorithm, we can efficiently calculate the posterior marginals $p(h_i|\mathbf{v} = \hat{\mathbf{v}})$ up to a normalization coefficient whose value can be found efficiently using a local computation. Any summations over variables in \mathbf{v} then collapse into a single term.

We have assumed throughout this section that we are dealing with discrete variables. However, there is nothing specific to discrete variables either in the graphical framework or in the probabilistic construction of the sum-product algorithm. For

Table 8.1 Example of a joint distribution over two binary variables for which the maximum of the joint distribution occurs for different variable values compared to the maxima of the two marginals.

	$x = 0$	$x = 1$
$y = 0$	0.3	0.4
$y = 1$	0.3	0.0

continuous variables the summations are simply replaced by integrations. We shall give an example of the sum-product algorithm applied to a graph of linear-Gaussian variables when we consider linear dynamical systems.

8.4.5 The max-sum algorithm

The sum-product algorithm allows us to take a joint distribution $p(\mathbf{x})$ expressed as a factor graph and efficiently find marginals over the component variables. Two other common tasks are to find a setting of the variables that has the largest probability and to find the value of that probability. These can be addressed through a closely related algorithm called *max-sum*, which can be viewed as an application of *dynamic programming* in the context of graphical models (Cormen *et al.*, 2001).

A simple approach to finding latent variable values having high probability would be to run the sum-product algorithm to obtain the marginals $p(x_i)$ for every variable, and then, for each marginal in turn, to find the value x_i^* that maximizes that marginal. However, this would give the set of values that are *individually* the most probable. In practice, we typically wish to find the set of values that *jointly* have the largest probability, in other words the vector \mathbf{x}^{\max} that maximizes the joint distribution, so that

$$\mathbf{x}^{\max} = \arg \max_{\mathbf{x}} p(\mathbf{x}) \quad (8.87)$$

for which the corresponding value of the joint probability will be given by

$$p(\mathbf{x}^{\max}) = \max_{\mathbf{x}} p(\mathbf{x}). \quad (8.88)$$

In general, \mathbf{x}^{\max} is not the same as the set of x_i^* values, as we can easily show using a simple example. Consider the joint distribution $p(x, y)$ over two binary variables $x, y \in \{0, 1\}$ given in Table 8.1. The joint distribution is maximized by setting $x = 1$ and $y = 0$, corresponding the value 0.4. However, the marginal for $p(x)$, obtained by summing over both values of y , is given by $p(x = 0) = 0.6$ and $p(x = 1) = 0.4$, and similarly the marginal for y is given by $p(y = 0) = 0.7$ and $p(y = 1) = 0.3$, and so the marginals are maximized by $x = 0$ and $y = 0$, which corresponds to a value of 0.3 for the joint distribution. In fact, it is not difficult to construct examples for which the set of individually most probable values has probability zero under the joint distribution.

We therefore seek an efficient algorithm for finding the value of \mathbf{x} that maximizes the joint distribution $p(\mathbf{x})$ and that will allow us to obtain the value of the joint distribution at its maximum. To address the second of these problems, we shall simply write out the max operator in terms of its components

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_1} \dots \max_{x_M} p(\mathbf{x}) \quad (8.89)$$

Section 13.3

Exercise 8.27

where M is the total number of variables, and then substitute for $p(\mathbf{x})$ using its expansion in terms of a product of factors. In deriving the sum-product algorithm, we made use of the distributive law (8.53) for multiplication. Here we make use of the analogous law for the max operator

$$\max(ab, ac) = a \max(b, c) \quad (8.90)$$

which holds if $a \geq 0$ (as will always be the case for the factors in a graphical model). This allows us to exchange products with maximizations.

Consider first the simple example of a chain of nodes described by (8.49). The evaluation of the probability maximum can be written as

$$\begin{aligned} \max_{\mathbf{x}} p(\mathbf{x}) &= \frac{1}{Z} \max_{x_1} \cdots \max_{x_N} [\psi_{1,2}(x_1, x_2) \cdots \psi_{N-1,N}(x_{N-1}, x_N)] \\ &= \frac{1}{Z} \max_{x_1} \left[\psi_{1,2}(x_1, x_2) \left[\cdots \max_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right] \right]. \end{aligned}$$

As with the calculation of marginals, we see that exchanging the max and product operators results in a much more efficient computation, and one that is easily interpreted in terms of messages passed from node x_N backwards along the chain to node x_1 .

We can readily generalize this result to arbitrary tree-structured factor graphs by substituting the expression (8.59) for the factor graph expansion into (8.89) and again exchanging maximizations with products. The structure of this calculation is identical to that of the sum-product algorithm, and so we can simply translate those results into the present context. In particular, suppose that we designate a particular variable node as the ‘root’ of the graph. Then we start a set of messages propagating inwards from the leaves of the tree towards the root, with each node sending its message towards the root once it has received all incoming messages from its other neighbours. The final maximization is performed over the product of all messages arriving at the root node, and gives the maximum value for $p(\mathbf{x})$. This could be called the *max-product* algorithm and is identical to the sum-product algorithm except that summations are replaced by maximizations. Note that at this stage, messages have been sent from leaves to the root, but not in the other direction.

In practice, products of many small probabilities can lead to numerical underflow problems, and so it is convenient to work with the logarithm of the joint distribution. The logarithm is a monotonic function, so that if $a > b$ then $\ln a > \ln b$, and hence the max operator and the logarithm function can be interchanged, so that

$$\ln \left(\max_{\mathbf{x}} p(\mathbf{x}) \right) = \max_{\mathbf{x}} \ln p(\mathbf{x}). \quad (8.91)$$

The distributive property is preserved because

$$\max(a + b, a + c) = a + \max(b, c). \quad (8.92)$$

Thus taking the logarithm simply has the effect of replacing the products in the max-product algorithm with sums, and so we obtain the *max-sum* algorithm. From

the results (8.66) and (8.69) derived earlier for the sum-product algorithm, we can readily write down the max-sum algorithm in terms of message passing simply by replacing ‘sum’ with ‘max’ and replacing products with sums of logarithms to give

$$\mu_{f \rightarrow x}(x) = \max_{x_1, \dots, x_M} \left[\ln f(x, x_1, \dots, x_M) + \sum_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \rightarrow f}(x_m) \right] \quad (8.93)$$

$$\mu_{x \rightarrow f}(x) = \sum_{l \in \text{ne}(x) \setminus f} \mu_{f_l \rightarrow x}(x). \quad (8.94)$$

The initial messages sent by the leaf nodes are obtained by analogy with (8.70) and (8.71) and are given by

$$\mu_{x \rightarrow f}(x) = 0 \quad (8.95)$$

$$\mu_{f \rightarrow x}(x) = \ln f(x) \quad (8.96)$$

while at the root node the maximum probability can then be computed, by analogy with (8.63), using

$$p^{\max} = \max_x \left[\sum_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x) \right]. \quad (8.97)$$

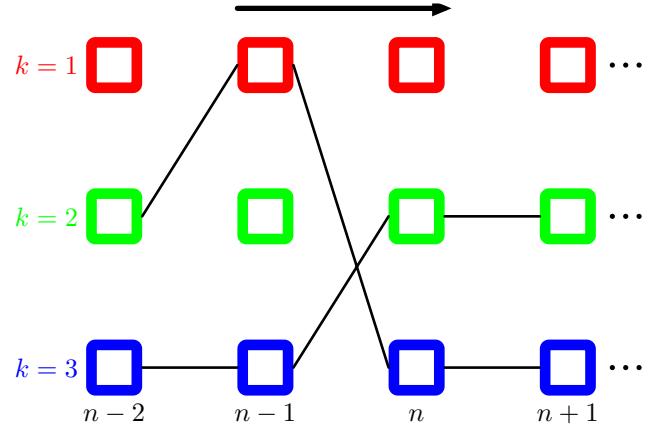
So far, we have seen how to find the maximum of the joint distribution by propagating messages from the leaves to an arbitrarily chosen root node. The result will be the same irrespective of which node is chosen as the root. Now we turn to the second problem of finding the configuration of the variables for which the joint distribution attains this maximum value. So far, we have sent messages from the leaves to the root. The process of evaluating (8.97) will also give the value x^{\max} for the most probable value of the root node variable, defined by

$$x^{\max} = \arg \max_x \left[\sum_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x) \right]. \quad (8.98)$$

At this point, we might be tempted simply to continue with the message passing algorithm and send messages from the root back out to the leaves, using (8.93) and (8.94), then apply (8.98) to all of the remaining variable nodes. However, because we are now maximizing rather than summing, it is possible that there may be multiple configurations of \mathbf{x} all of which give rise to the maximum value for $p(\mathbf{x})$. In such cases, this strategy can fail because it is possible for the individual variable values obtained by maximizing the product of messages at each node to belong to different maximizing configurations, giving an overall configuration that no longer corresponds to a maximum.

The problem can be resolved by adopting a rather different kind of message passing from the root node to the leaves. To see how this works, let us return once again to the simple chain example of N variables x_1, \dots, x_N each having K states,

Figure 8.53 A lattice, or trellis, diagram showing explicitly the K possible states (one per row of the diagram) for each of the variables x_n in the chain model. In this illustration $K = 3$. The arrow shows the direction of message passing in the max-product algorithm. For every state k of each variable x_n (corresponding to column n of the diagram) the function $\phi(x_n)$ defines a unique state at the previous variable, indicated by the black lines. The two paths through the lattice correspond to configurations that give the global maximum of the joint probability distribution, and either of these can be found by tracing back along the black lines in the opposite direction to the arrow.



corresponding to the graph shown in Figure 8.38. Suppose we take node x_N to be the root node. Then in the first phase, we propagate messages from the leaf node x_1 to the root node using

$$\begin{aligned}\mu_{x_n \rightarrow f_{n,n+1}}(x_n) &= \mu_{f_{n-1,n} \rightarrow x_n}(x_n) \\ \mu_{f_{n-1,n} \rightarrow x_n}(x_n) &= \max_{x_{n-1}} [\ln f_{n-1,n}(x_{n-1}, x_n) + \mu_{x_{n-1} \rightarrow f_{n-1,n}}(x_n)]\end{aligned}$$

which follow from applying (8.94) and (8.93) to this particular graph. The initial message sent from the leaf node is simply

$$\mu_{x_1 \rightarrow f_{1,2}}(x_1) = 0. \quad (8.99)$$

The most probable value for x_N is then given by

$$x_N^{\max} = \arg \max_{x_N} [\mu_{f_{N-1,N} \rightarrow x_N}(x_N)]. \quad (8.100)$$

Now we need to determine the states of the previous variables that correspond to the same maximizing configuration. This can be done by keeping track of which values of the variables gave rise to the maximum state of each variable, in other words by storing quantities given by

$$\phi(x_n) = \arg \max_{x_{n-1}} [\ln f_{n-1,n}(x_{n-1}, x_n) + \mu_{x_{n-1} \rightarrow f_{n-1,n}}(x_n)]. \quad (8.101)$$

To understand better what is happening, it is helpful to represent the chain of variables in terms of a *lattice* or *trellis* diagram as shown in Figure 8.53. Note that this is not a probabilistic graphical model because the nodes represent individual states of variables, while each variable corresponds to a column of such states in the diagram. For each state of a given variable, there is a unique state of the previous variable that maximizes the probability (ties are broken either systematically or at random), corresponding to the function $\phi(x_n)$ given by (8.101), and this is indicated

by the lines connecting the nodes. Once we know the most probable value of the final node x_N , we can then simply follow the link back to find the most probable state of node x_{N-1} and so on back to the initial node x_1 . This corresponds to propagating a message back down the chain using

$$x_{n-1}^{\max} = \phi(x_n^{\max}) \quad (8.102)$$

and is known as *back-tracking*. Note that there could be several values of x_{n-1} all of which give the maximum value in (8.101). Provided we chose one of these values when we do the back-tracking, we are assured of a globally consistent maximizing configuration.

In Figure 8.53, we have indicated two paths, each of which we shall suppose corresponds to a global maximum of the joint probability distribution. If $k = 2$ and $k = 3$ each represent possible values of x_N^{\max} , then starting from either state and tracing back along the black lines, which corresponds to iterating (8.102), we obtain a valid global maximum configuration. Note that if we had run a forward pass of max-sum message passing followed by a backward pass and then applied (8.98) at each node separately, we could end up selecting some states from one path and some from the other path, giving an overall configuration that is not a global maximizer. We see that it is necessary instead to keep track of the maximizing states during the forward pass using the functions $\phi(x_n)$ and then use back-tracking to find a consistent solution.

The extension to a general tree-structured factor graph should now be clear. If a message is sent from a factor node f to a variable node x , a maximization is performed over all other variable nodes x_1, \dots, x_M that are neighbours of that factor node, using (8.93). When we perform this maximization, we keep a record of which values of the variables x_1, \dots, x_M gave rise to the maximum. Then in the back-tracking step, having found x^{\max} , we can then use these stored values to assign consistent maximizing states $x_1^{\max}, \dots, x_M^{\max}$. The max-sum algorithm, with back-tracking, gives an exact maximizing configuration for the variables provided the factor graph is a tree. An important application of this technique is for finding the most probable sequence of hidden states in a hidden Markov model, in which case it is known as the *Viterbi* algorithm.

As with the sum-product algorithm, the inclusion of evidence in the form of observed variables is straightforward. The observed variables are clamped to their observed values, and the maximization is performed over the remaining hidden variables. This can be shown formally by including identity functions for the observed variables into the factor functions, as we did for the sum-product algorithm.

It is interesting to compare max-sum with the iterated conditional modes (ICM) algorithm described on page 389. Each step in ICM is computationally simpler because the ‘messages’ that are passed from one node to the next comprise a single value consisting of the new state of the node for which the conditional distribution is maximized. The max-sum algorithm is more complex because the messages are functions of node variables x and hence comprise a set of K values for each possible state of x . Unlike max-sum, however, ICM is not guaranteed to find a global maximum even for tree-structured graphs.

Section 13.2

8.4.6 Exact inference in general graphs

The sum-product and max-sum algorithms provide efficient and exact solutions to inference problems in tree-structured graphs. For many practical applications, however, we have to deal with graphs having loops.

The message passing framework can be generalized to arbitrary graph topologies, giving an exact inference procedure known as the *junction tree algorithm* (Lauritzen and Spiegelhalter, 1988; Jordan, 2007). Here we give a brief outline of the key steps involved. This is not intended to convey a detailed understanding of the algorithm, but rather to give a flavour of the various stages involved. If the starting point is a directed graph, it is first converted to an undirected graph by moralization, whereas if starting from an undirected graph this step is not required. Next the graph is *triangulated*, which involves finding chord-less cycles containing four or more nodes and adding extra links to eliminate such chord-less cycles. For instance, in the graph in Figure 8.36, the cycle $A-C-B-D-A$ is chord-less a link could be added between A and B or alternatively between C and D . Note that the joint distribution for the resulting triangulated graph is still defined by a product of the same potential functions, but these are now considered to be functions over expanded sets of variables. Next the triangulated graph is used to construct a new tree-structured undirected graph called a *join tree*, whose nodes correspond to the maximal cliques of the triangulated graph, and whose links connect pairs of cliques that have variables in common. The selection of which pairs of cliques to connect in this way is important and is done so as to give a *maximal spanning tree* defined as follows. Of all possible trees that link up the cliques, the one that is chosen is one for which the *weight* of the tree is largest, where the weight for a link is the number of nodes shared by the two cliques it connects, and the weight for the tree is the sum of the weights for the links. If the tree is condensed, so that any clique that is a subset of another clique is absorbed into the larger clique, this gives a *junction tree*. As a consequence of the triangulation step, the resulting tree satisfies the *running intersection property*, which means that if a variable is contained in two cliques, then it must also be contained in every clique on the path that connects them. This ensures that inference about variables will be consistent across the graph. Finally, a two-stage message passing algorithm, essentially equivalent to the sum-product algorithm, can now be applied to this junction tree in order to find marginals and conditionals. Although the junction tree algorithm sounds complicated, at its heart is the simple idea that we have used already of exploiting the factorization properties of the distribution to allow sums and products to be interchanged so that partial summations can be performed, thereby avoiding having to work directly with the joint distribution. The role of the junction tree is to provide a precise and efficient way to organize these computations. It is worth emphasizing that this is achieved using purely graphical operations!

The junction tree is exact for arbitrary graphs and is efficient in the sense that for a given graph there does not in general exist a computationally cheaper approach. Unfortunately, the algorithm must work with the joint distributions within each node (each of which corresponds to a clique of the triangulated graph) and so the computational cost of the algorithm is determined by the number of variables in the largest

clique and will grow exponentially with this number in the case of discrete variables. An important concept is the *treewidth* of a graph (Bodlaender, 1993), which is defined in terms of the number of variables in the largest clique. In fact, it is defined to be as one less than the size of the largest clique, to ensure that a tree has a treewidth of 1. Because there in general there can be multiple different junction trees that can be constructed from a given starting graph, the treewidth is defined by the junction tree for which the largest clique has the fewest variables. If the treewidth of the original graph is high, the junction tree algorithm becomes impractical.

8.4.7 Loopy belief propagation

For many problems of practical interest, it will not be feasible to use exact inference, and so we need to exploit effective approximation methods. An important class of such approximations, that can broadly be called *variational* methods, will be discussed in detail in Chapter 10. Complementing these deterministic approaches is a wide range of *sampling* methods, also called *Monte Carlo* methods, that are based on stochastic numerical sampling from distributions and that will be discussed at length in Chapter 11.

Here we consider one simple approach to approximate inference in graphs with loops, which builds directly on the previous discussion of exact inference in trees. The idea is simply to apply the sum-product algorithm even though there is no guarantee that it will yield good results. This approach is known as *loopy belief propagation* (Frey and MacKay, 1998) and is possible because the message passing rules (8.66) and (8.69) for the sum-product algorithm are purely local. However, because the graph now has cycles, information can flow many times around the graph. For some models, the algorithm will converge, whereas for others it will not.

In order to apply this approach, we need to define a *message passing schedule*. Let us assume that one message is passed at a time on any given link and in any given direction. Each message sent from a node replaces any previous message sent in the same direction across the same link and will itself be a function only of the most recent messages received by that node at previous steps of the algorithm.

We have seen that a message can only be sent across a link from a node when all other messages have been received by that node across its other links. Because there are loops in the graph, this raises the problem of how to initiate the message passing algorithm. To resolve this, we suppose that an initial message given by the unit function has been passed across every link in each direction. Every node is then in a position to send a message.

There are now many possible ways to organize the message passing schedule. For example, the *flooding schedule* simultaneously passes a message across every link in both directions at each time step, whereas schedules that pass one message at a time are called *serial schedules*.

Following Kschischang *et al.* (2001), we will say that a (variable or factor) node a has a message *pending* on its link to a node b if node a has received any message on any of its other links since the last time it sent a message to b . Thus, when a node receives a message on one of its links, this creates pending messages on all of its other links. Only pending messages need to be transmitted because

other messages would simply duplicate the previous message on the same link. For graphs that have a tree structure, any schedule that sends only pending messages will eventually terminate once a message has passed in each direction across every link. At this point, there are no pending messages, and the product of the received messages at every variable give the exact marginal. In graphs having loops, however, the algorithm may never terminate because there might always be pending messages, although in practice it is generally found to converge within a reasonable time for most applications. Once the algorithm has converged, or once it has been stopped if convergence is not observed, the (approximate) local marginals can be computed using the product of the most recently received incoming messages to each variable node or factor node on every link.

In some applications, the loopy belief propagation algorithm can give poor results, whereas in other applications it has proven to be very effective. In particular, state-of-the-art algorithms for decoding certain kinds of error-correcting codes are equivalent to loopy belief propagation (Gallager, 1963; Berrou *et al.*, 1993; McEliece *et al.*, 1998; MacKay and Neal, 1999; Frey, 1998).

8.4.8 Learning the graph structure

In our discussion of inference in graphical models, we have assumed that the structure of the graph is known and fixed. However, there is also interest in going beyond the inference problem and learning the graph structure itself from data (Friedman and Koller, 2003). This requires that we define a space of possible structures as well as a measure that can be used to score each structure.

From a Bayesian viewpoint, we would ideally like to compute a posterior distribution over graph structures and to make predictions by averaging with respect to this distribution. If we have a prior $p(m)$ over graphs indexed by m , then the posterior distribution is given by

$$p(m|\mathcal{D}) \propto p(m)p(\mathcal{D}|m) \quad (8.103)$$

where \mathcal{D} is the observed data set. The model evidence $p(\mathcal{D}|m)$ then provides the score for each model. However, evaluation of the evidence involves marginalization over the latent variables and presents a challenging computational problem for many models.

Exploring the space of structures can also be problematic. Because the number of different graph structures grows exponentially with the number of nodes, it is often necessary to resort to heuristics to find good candidates.

Exercises

- 8.1** (*) **www** By marginalizing out the variables in order, show that the representation (8.5) for the joint distribution of a directed graph is correctly normalized, provided each of the conditional distributions is normalized.
- 8.2** (*) **www** Show that the property of there being no directed cycles in a directed graph follows from the statement that there exists an ordered numbering of the nodes such that for each node there are no links going to a lower-numbered node.

Table 8.2 The joint distribution over three binary variables.

a	b	c	$p(a, b, c)$
0	0	0	0.192
0	0	1	0.144
0	1	0	0.048
0	1	1	0.216
1	0	0	0.192
1	0	1	0.064
1	1	0	0.048
1	1	1	0.096

8.3 (**) Consider three binary variables $a, b, c \in \{0, 1\}$ having the joint distribution given in Table 8.2. Show by direct evaluation that this distribution has the property that a and b are marginally dependent, so that $p(a, b) \neq p(a)p(b)$, but that they become independent when conditioned on c , so that $p(a, b|c) = p(a|c)p(b|c)$ for both $c = 0$ and $c = 1$.

8.4 (**) Evaluate the distributions $p(a)$, $p(b|c)$, and $p(c|a)$ corresponding to the joint distribution given in Table 8.2. Hence show by direct evaluation that $p(a, b, c) = p(a)p(c|a)p(b|c)$. Draw the corresponding directed graph.

8.5 (*) **www** Draw a directed probabilistic graphical model corresponding to the relevance vector machine described by (7.79) and (7.80).

8.6 (*) For the model shown in Figure 8.13, we have seen that the number of parameters required to specify the conditional distribution $p(y|x_1, \dots, x_M)$, where $x_i \in \{0, 1\}$, could be reduced from 2^M to $M + 1$ by making use of the logistic sigmoid representation (8.10). An alternative representation (Pearl, 1988) is given by

$$p(y = 1|x_1, \dots, x_M) = 1 - (1 - \mu_0) \prod_{i=1}^M (1 - \mu_i)^{x_i} \quad (8.104)$$

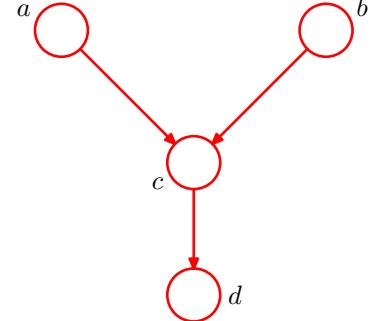
where the parameters μ_i represent the probabilities $p(x_i = 1)$, and μ_0 is an additional parameter satisfying $0 \leq \mu_0 \leq 1$. The conditional distribution (8.104) is known as the *noisy-OR*. Show that this can be interpreted as a ‘soft’ (probabilistic) form of the logical OR function (i.e., the function that gives $y = 1$ whenever at least one of the $x_i = 1$). Discuss the interpretation of μ_0 .

8.7 (**) Using the recursion relations (8.15) and (8.16), show that the mean and covariance of the joint distribution for the graph shown in Figure 8.14 are given by (8.17) and (8.18), respectively.

8.8 (*) **www** Show that $a \perp\!\!\!\perp b, c | d$ implies $a \perp\!\!\!\perp b | d$.

8.9 (*) **www** Using the d-separation criterion, show that the conditional distribution for a node x in a directed graph, conditioned on all of the nodes in the Markov blanket, is independent of the remaining variables in the graph.

Figure 8.54 Example of a graphical model used to explore the conditional independence properties of the head-to-head path $a-c-b$ when a descendant of c , namely the node d , is observed.



- 8.10** (*) Consider the directed graph shown in Figure 8.54 in which none of the variables is observed. Show that $a \perp\!\!\!\perp b | \emptyset$. Suppose we now observe the variable d . Show that in general $a \not\perp\!\!\!\perp b | d$.
- 8.11** (**) Consider the example of the car fuel system shown in Figure 8.21, and suppose that instead of observing the state of the fuel gauge G directly, the gauge is seen by the driver D who reports to us the reading on the gauge. This report is either that the gauge shows full $D = 1$ or that it shows empty $D = 0$. Our driver is a bit unreliable, as expressed through the following probabilities

$$p(D = 1|G = 1) = 0.9 \quad (8.105)$$

$$p(D = 0|G = 0) = 0.9. \quad (8.106)$$

Suppose that the driver tells us that the fuel gauge shows empty, in other words that we observe $D = 0$. Evaluate the probability that the tank is empty given only this observation. Similarly, evaluate the corresponding probability given also the observation that the battery is flat, and note that this second probability is lower. Discuss the intuition behind this result, and relate the result to Figure 8.54.

- 8.12** (*) **www** Show that there are $2^{M(M-1)/2}$ distinct undirected graphs over a set of M distinct random variables. Draw the 8 possibilities for the case of $M = 3$.
- 8.13** (*) Consider the use of iterated conditional modes (ICM) to minimize the energy function given by (8.42). Write down an expression for the difference in the values of the energy associated with the two states of a particular variable x_j , with all other variables held fixed, and show that it depends only on quantities that are local to x_j in the graph.
- 8.14** (*) Consider a particular case of the energy function given by (8.42) in which the coefficients $\beta = h = 0$. Show that the most probable configuration of the latent variables is given by $x_i = y_i$ for all i .
- 8.15** (**) **www** Show that the joint distribution $p(x_{n-1}, x_n)$ for two neighbouring nodes in the graph shown in Figure 8.38 is given by an expression of the form (8.58).

- 8.16** (**) Consider the inference problem of evaluating $p(\mathbf{x}_n | \mathbf{x}_N)$ for the graph shown in Figure 8.38, for all nodes $n \in \{1, \dots, N-1\}$. Show that the message passing algorithm discussed in Section 8.4.1 can be used to solve this efficiently, and discuss which messages are modified and in what way.
- 8.17** (**) Consider a graph of the form shown in Figure 8.38 having $N = 5$ nodes, in which nodes x_3 and x_5 are observed. Use d-separation to show that $x_2 \perp\!\!\!\perp x_5 | x_3$. Show that if the message passing algorithm of Section 8.4.1 is applied to the evaluation of $p(x_2 | x_3, x_5)$, the result will be independent of the value of x_5 .
- 8.18** (**) **www** Show that a distribution represented by a directed tree can trivially be written as an equivalent distribution over the corresponding undirected tree. Also show that a distribution expressed as an undirected tree can, by suitable normalization of the clique potentials, be written as a directed tree. Calculate the number of distinct directed trees that can be constructed from a given undirected tree.
- 8.19** (**) Apply the sum-product algorithm derived in Section 8.4.4 to the chain-of-nodes model discussed in Section 8.4.1 and show that the results (8.54), (8.55), and (8.57) are recovered as a special case.
- 8.20** (*) **www** Consider the message passing protocol for the sum-product algorithm on a tree-structured factor graph in which messages are first propagated from the leaves to an arbitrarily chosen root node and then from the root node out to the leaves. Use proof by induction to show that the messages can be passed in such an order that at every step, each node that must send a message has received all of the incoming messages necessary to construct its outgoing messages.
- 8.21** (**) **www** Show that the marginal distributions $p(\mathbf{x}_s)$ over the sets of variables \mathbf{x}_s associated with each of the factors $f_x(\mathbf{x}_s)$ in a factor graph can be found by first running the sum-product message passing algorithm and then evaluating the required marginals using (8.72).
- 8.22** (*) Consider a tree-structured factor graph, in which a given subset of the variable nodes form a connected subgraph (i.e., any variable node of the subset is connected to at least one of the other variable nodes via a single factor node). Show how the sum-product algorithm can be used to compute the marginal distribution over that subset.
- 8.23** (**) **www** In Section 8.4.4, we showed that the marginal distribution $p(x_i)$ for a variable node x_i in a factor graph is given by the product of the messages arriving at this node from neighbouring factor nodes in the form (8.63). Show that the marginal $p(x_i)$ can also be written as the product of the incoming message along any one of the links with the outgoing message along the same link.
- 8.24** (**) Show that the marginal distribution for the variables \mathbf{x}_s in a factor $f_s(\mathbf{x}_s)$ in a tree-structured factor graph, after running the sum-product message passing algorithm, can be written as the product of the message arriving at the factor node along all its links, times the local factor $f(\mathbf{x}_s)$, in the form (8.72).

- 8.25** (**) In (8.86), we verified that the sum-product algorithm run on the graph in Figure 8.51 with node x_3 designated as the root node gives the correct marginal for x_2 . Show that the correct marginals are obtained also for x_1 and x_3 . Similarly, show that the use of the result (8.72) after running the sum-product algorithm on this graph gives the correct joint distribution for x_1, x_2 .
- 8.26** (*) Consider a tree-structured factor graph over discrete variables, and suppose we wish to evaluate the joint distribution $p(x_a, x_b)$ associated with two variables x_a and x_b that do not belong to a common factor. Define a procedure for using the sum-product algorithm to evaluate this joint distribution in which one of the variables is successively clamped to each of its allowed values.
- 8.27** (**) Consider two discrete variables x and y each having three possible states, for example $x, y \in \{0, 1, 2\}$. Construct a joint distribution $p(x, y)$ over these variables having the property that the value \hat{x} that maximizes the marginal $p(x)$, along with the value \hat{y} that maximizes the marginal $p(y)$, together have probability zero under the joint distribution, so that $p(\hat{x}, \hat{y}) = 0$.
- 8.28** (**) **www** The concept of a *pending* message in the sum-product algorithm for a factor graph was defined in Section 8.4.7. Show that if the graph has one or more cycles, there will always be at least one pending message irrespective of how long the algorithm runs.
- 8.29** (**) **www** Show that if the sum-product algorithm is run on a factor graph with a tree structure (no loops), then after a finite number of messages have been sent, there will be no pending messages.

9

Mixture Models and EM

If we define a joint distribution over observed and latent variables, the corresponding distribution of the observed variables alone is obtained by marginalization. This allows relatively complex marginal distributions over observed variables to be expressed in terms of more tractable joint distributions over the expanded space of observed and latent variables. The introduction of latent variables thereby allows complicated distributions to be formed from simpler components. In this chapter, we shall see that mixture distributions, such as the Gaussian mixture discussed in Section 2.3.9, can be interpreted in terms of discrete latent variables. Continuous latent variables will form the subject of Chapter 12.

As well as providing a framework for building more complex probability distributions, mixture models can also be used to cluster data. We therefore begin our discussion of mixture distributions by considering the problem of finding clusters in a set of data points, which we approach first using a nonprobabilistic technique called the K -means algorithm (Lloyd, 1982). Then we introduce the latent variable

Section 9.2

Section 9.3

Section 9.4

view of mixture distributions in which the discrete latent variables can be interpreted as defining assignments of data points to specific components of the mixture. A general technique for finding maximum likelihood estimators in latent variable models is the expectation-maximization (EM) algorithm. We first of all use the Gaussian mixture distribution to motivate the EM algorithm in a fairly informal way, and then we give a more careful treatment based on the latent variable viewpoint. We shall see that the K -means algorithm corresponds to a particular nonprobabilistic limit of EM applied to mixtures of Gaussians. Finally, we discuss EM in some generality.

Gaussian mixture models are widely used in data mining, pattern recognition, machine learning, and statistical analysis. In many applications, their parameters are determined by maximum likelihood, typically using the EM algorithm. However, as we shall see there are some significant limitations to the maximum likelihood approach, and in Chapter 10 we shall show that an elegant Bayesian treatment can be given using the framework of variational inference. This requires little additional computation compared with EM, and it resolves the principal difficulties of maximum likelihood while also allowing the number of components in the mixture to be inferred automatically from the data.

9.1. K-means Clustering

We begin by considering the problem of identifying groups, or clusters, of data points in a multidimensional space. Suppose we have a data set $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ consisting of N observations of a random D -dimensional Euclidean variable \mathbf{x} . Our goal is to partition the data set into some number K of clusters, where we shall suppose for the moment that the value of K is given. Intuitively, we might think of a cluster as comprising a group of data points whose inter-point distances are small compared with the distances to points outside of the cluster. We can formalize this notion by first introducing a set of D -dimensional vectors $\boldsymbol{\mu}_k$, where $k = 1, \dots, K$, in which $\boldsymbol{\mu}_k$ is a prototype associated with the k^{th} cluster. As we shall see shortly, we can think of the $\boldsymbol{\mu}_k$ as representing the centres of the clusters. Our goal is then to find an assignment of data points to clusters, as well as a set of vectors $\{\boldsymbol{\mu}_k\}$, such that the sum of the squares of the distances of each data point to its closest vector $\boldsymbol{\mu}_k$, is a minimum.

It is convenient at this point to define some notation to describe the assignment of data points to clusters. For each data point \mathbf{x}_n , we introduce a corresponding set of binary indicator variables $r_{nk} \in \{0, 1\}$, where $k = 1, \dots, K$ describing which of the K clusters the data point \mathbf{x}_n is assigned to, so that if data point \mathbf{x}_n is assigned to cluster k then $r_{nk} = 1$, and $r_{nj} = 0$ for $j \neq k$. This is known as the 1-of- K coding scheme. We can then define an objective function, sometimes called a *distortion measure*, given by

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 \quad (9.1)$$

which represents the sum of the squares of the distances of each data point to its

assigned vector μ_k . Our goal is to find values for the $\{r_{nk}\}$ and the $\{\mu_k\}$ so as to minimize J . We can do this through an iterative procedure in which each iteration involves two successive steps corresponding to successive optimizations with respect to the r_{nk} and the μ_k . First we choose some initial values for the μ_k . Then in the first phase we minimize J with respect to the r_{nk} , keeping the μ_k fixed. In the second phase we minimize J with respect to the μ_k , keeping r_{nk} fixed. This two-stage optimization is then repeated until convergence. We shall see that these two stages of updating r_{nk} and updating μ_k correspond respectively to the E (expectation) and M (maximization) steps of the EM algorithm, and to emphasize this we shall use the terms E step and M step in the context of the K -means algorithm.

Consider first the determination of the r_{nk} . Because J in (9.1) is a linear function of r_{nk} , this optimization can be performed easily to give a closed form solution. The terms involving different n are independent and so we can optimize for each n separately by choosing r_{nk} to be 1 for whichever value of k gives the minimum value of $\|\mathbf{x}_n - \mu_k\|^2$. In other words, we simply assign the n^{th} data point to the closest cluster centre. More formally, this can be expressed as

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg \min_j \|\mathbf{x}_n - \mu_j\|^2 \\ 0 & \text{otherwise.} \end{cases} \quad (9.2)$$

Now consider the optimization of the μ_k with the r_{nk} held fixed. The objective function J is a quadratic function of μ_k , and it can be minimized by setting its derivative with respect to μ_k to zero giving

$$2 \sum_{n=1}^N r_{nk} (\mathbf{x}_n - \mu_k) = 0 \quad (9.3)$$

which we can easily solve for μ_k to give

$$\mu_k = \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}}. \quad (9.4)$$

The denominator in this expression is equal to the number of points assigned to cluster k , and so this result has a simple interpretation, namely set μ_k equal to the mean of all of the data points \mathbf{x}_n assigned to cluster k . For this reason, the procedure is known as the *K-means* algorithm.

The two phases of re-assigning data points to clusters and re-computing the cluster means are repeated in turn until there is no further change in the assignments (or until some maximum number of iterations is exceeded). Because each phase reduces the value of the objective function J , convergence of the algorithm is assured. However, it may converge to a local rather than global minimum of J . The convergence properties of the *K-means* algorithm were studied by MacQueen (1967).

The *K-means* algorithm is illustrated using the Old Faithful data set in Figure 9.1. For the purposes of this example, we have made a linear re-scaling of the data, known as *standardizing*, such that each of the variables has zero mean and unit standard deviation. For this example, we have chosen $K = 2$, and so in this

Section 9.4

Exercise 9.1

Appendix A

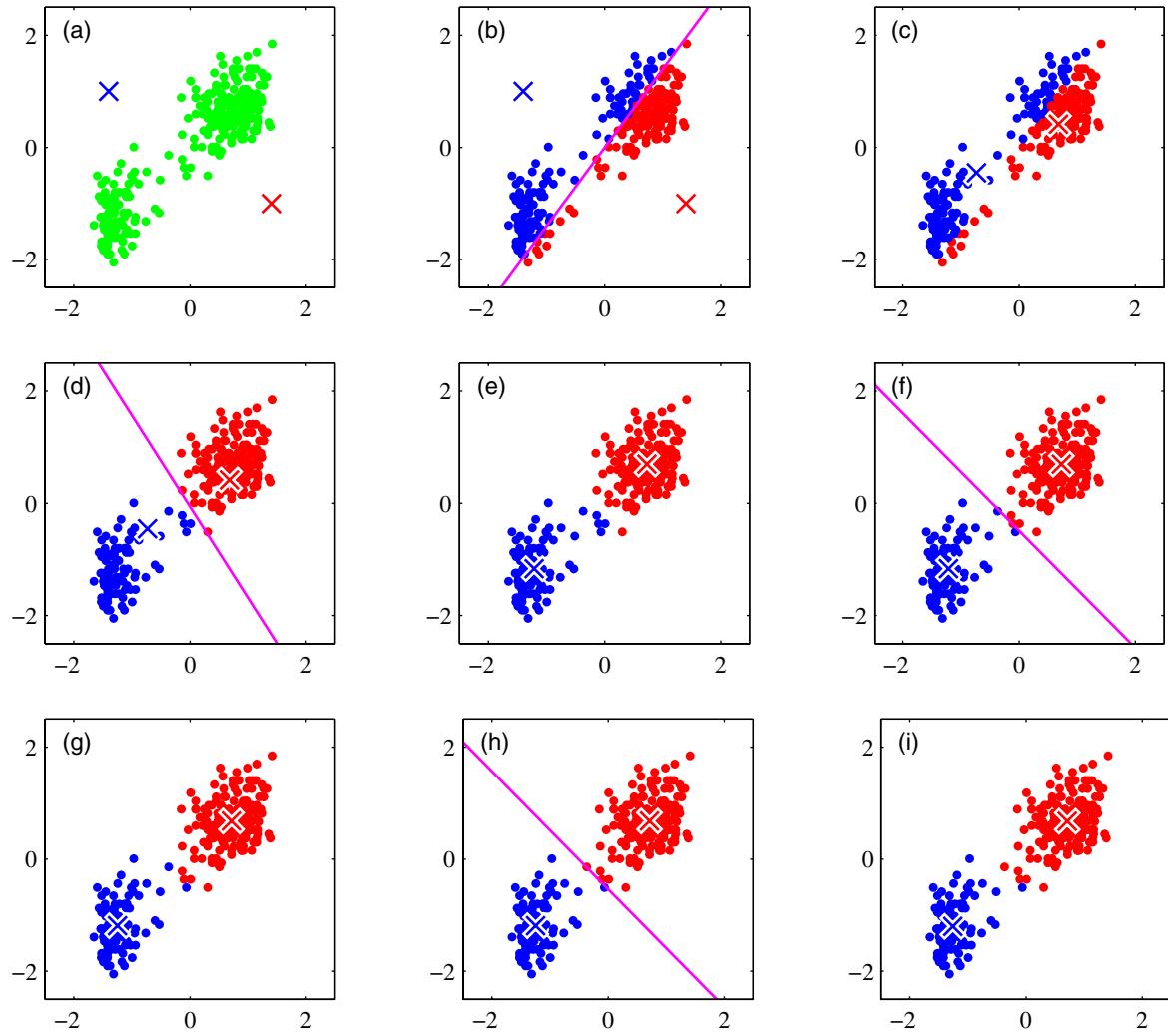
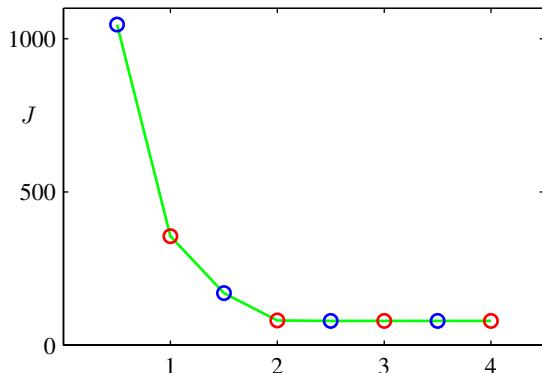


Figure 9.1 Illustration of the K -means algorithm using the re-scaled Old Faithful data set. (a) Green points denote the data set in a two-dimensional Euclidean space. The initial choices for centres μ_1 and μ_2 are shown by the red and blue crosses, respectively. (b) In the initial E step, each data point is assigned either to the red cluster or to the blue cluster, according to which cluster centre is nearer. This is equivalent to classifying the points according to which side of the perpendicular bisector of the two cluster centres, shown by the magenta line, they lie on. (c) In the subsequent M step, each cluster centre is re-computed to be the mean of the points assigned to the corresponding cluster. (d)–(i) show successive E and M steps through to final convergence of the algorithm.

Figure 9.2 Plot of the cost function J given by (9.1) after each E step (blue points) and M step (red points) of the K -means algorithm for the example shown in Figure 9.1. The algorithm has converged after the third M step, and the final EM cycle produces no changes in either the assignments or the prototype vectors.



case, the assignment of each data point to the nearest cluster centre is equivalent to a classification of the data points according to which side they lie of the perpendicular bisector of the two cluster centres. A plot of the cost function J given by (9.1) for the Old Faithful example is shown in Figure 9.2.

Note that we have deliberately chosen poor initial values for the cluster centres so that the algorithm takes several steps before convergence. In practice, a better initialization procedure would be to choose the cluster centres μ_k to be equal to a random subset of K data points. It is also worth noting that the K -means algorithm itself is often used to initialize the parameters in a Gaussian mixture model before applying the EM algorithm.

A direct implementation of the K -means algorithm as discussed here can be relatively slow, because in each E step it is necessary to compute the Euclidean distance between every prototype vector and every data point. Various schemes have been proposed for speeding up the K -means algorithm, some of which are based on precomputing a data structure such as a tree such that nearby points are in the same subtree (Ramasubramanian and Paliwal, 1990; Moore, 2000). Other approaches make use of the triangle inequality for distances, thereby avoiding unnecessary distance calculations (Hodgson, 1998; Elkan, 2003).

So far, we have considered a batch version of K -means in which the whole data set is used together to update the prototype vectors. We can also derive an on-line stochastic algorithm (MacQueen, 1967) by applying the Robbins-Monro procedure to the problem of finding the roots of the regression function given by the derivatives of J in (9.1) with respect to μ_k . This leads to a sequential update in which, for each data point \mathbf{x}_n in turn, we update the nearest prototype μ_k using

$$\mu_k^{\text{new}} = \mu_k^{\text{old}} + \eta_n(\mathbf{x}_n - \mu_k^{\text{old}}) \quad (9.5)$$

where η_n is the learning rate parameter, which is typically made to decrease monotonically as more data points are considered.

The K -means algorithm is based on the use of squared Euclidean distance as the measure of dissimilarity between a data point and a prototype vector. Not only does this limit the type of data variables that can be considered (it would be inappropriate for cases where some or all of the variables represent categorical labels for instance),

Section 9.2.2

Section 2.3.5

Exercise 9.2

Section 2.3.7

but it can also make the determination of the cluster means nonrobust to outliers. We can generalize the K -means algorithm by introducing a more general dissimilarity measure $\mathcal{V}(\mathbf{x}, \mathbf{x}')$ between two vectors \mathbf{x} and \mathbf{x}' and then minimizing the following distortion measure

$$\tilde{J} = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \mathcal{V}(\mathbf{x}_n, \boldsymbol{\mu}_k) \quad (9.6)$$

which gives the K -medoids algorithm. The E step again involves, for given cluster prototypes $\boldsymbol{\mu}_k$, assigning each data point to the cluster for which the dissimilarity to the corresponding prototype is smallest. The computational cost of this is $O(KN)$, as is the case for the standard K -means algorithm. For a general choice of dissimilarity measure, the M step is potentially more complex than for K -means, and so it is common to restrict each cluster prototype to be equal to one of the data vectors assigned to that cluster, as this allows the algorithm to be implemented for any choice of dissimilarity measure $\mathcal{V}(\cdot, \cdot)$ so long as it can be readily evaluated. Thus the M step involves, for each cluster k , a discrete search over the N_k points assigned to that cluster, which requires $O(N_k^2)$ evaluations of $\mathcal{V}(\cdot, \cdot)$.

One notable feature of the K -means algorithm is that at each iteration, every data point is assigned uniquely to one, and only one, of the clusters. Whereas some data points will be much closer to a particular centre $\boldsymbol{\mu}_k$ than to any other centre, there may be other data points that lie roughly midway between cluster centres. In the latter case, it is not clear that the hard assignment to the nearest cluster is the most appropriate. We shall see in the next section that by adopting a probabilistic approach, we obtain ‘soft’ assignments of data points to clusters in a way that reflects the level of uncertainty over the most appropriate assignment. This probabilistic formulation brings with it numerous benefits.

9.1.1 Image segmentation and compression

As an illustration of the application of the K -means algorithm, we consider the related problems of image segmentation and image compression. The goal of segmentation is to partition an image into regions each of which has a reasonably homogeneous visual appearance or which corresponds to objects or parts of objects (Forsyth and Ponce, 2003). Each pixel in an image is a point in a 3-dimensional space comprising the intensities of the red, blue, and green channels, and our segmentation algorithm simply treats each pixel in the image as a separate data point. Note that strictly this space is not Euclidean because the channel intensities are bounded by the interval $[0, 1]$. Nevertheless, we can apply the K -means algorithm without difficulty. We illustrate the result of running K -means to convergence, for any particular value of K , by re-drawing the image replacing each pixel vector with the $\{R, G, B\}$ intensity triplet given by the centre $\boldsymbol{\mu}_k$ to which that pixel has been assigned. Results for various values of K are shown in Figure 9.3. We see that for a given value of K , the algorithm is representing the image using a palette of only K colours. It should be emphasized that this use of K -means is not a particularly sophisticated approach to image segmentation, not least because it takes no account of the spatial proximity of different pixels. The image segmentation problem is in general extremely difficult

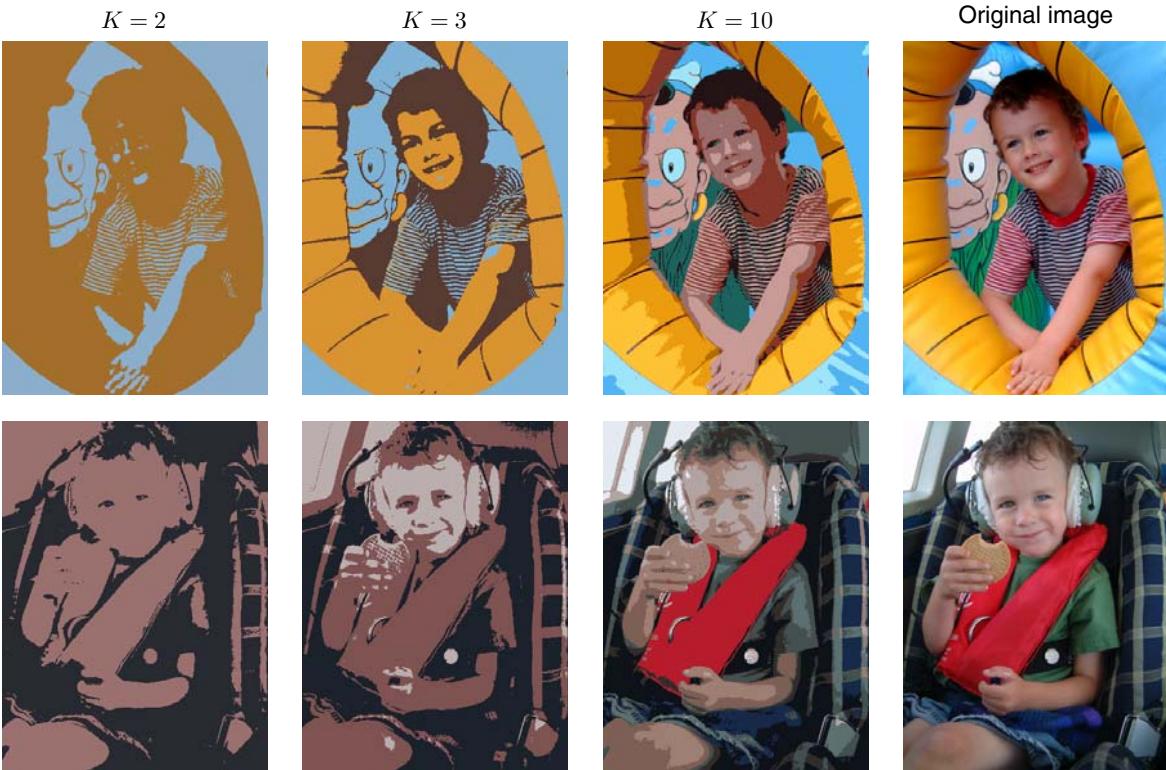


Figure 9.3 Two examples of the application of the K -means clustering algorithm to image segmentation showing the initial images together with their K -means segmentations obtained using various values of K . This also illustrates the use of vector quantization for data compression, in which smaller values of K give higher compression at the expense of poorer image quality.

and remains the subject of active research and is introduced here simply to illustrate the behaviour of the K -means algorithm.

We can also use the result of a clustering algorithm to perform data compression. It is important to distinguish between *lossless data compression*, in which the goal is to be able to reconstruct the original data exactly from the compressed representation, and *lossy data compression*, in which we accept some errors in the reconstruction in return for higher levels of compression than can be achieved in the lossless case. We can apply the K -means algorithm to the problem of lossy data compression as follows. For each of the N data points, we store only the identity k of the cluster to which it is assigned. We also store the values of the K cluster centres μ_k , which typically requires significantly less data, provided we choose $K \ll N$. Each data point is then approximated by its nearest centre μ_k . New data points can similarly be compressed by first finding the nearest μ_k and then storing the label k instead of the original data vector. This framework is often called *vector quantization*, and the vectors μ_k are called *code-book vectors*.

The image segmentation problem discussed above also provides an illustration of the use of clustering for data compression. Suppose the original image has N pixels comprising $\{R, G, B\}$ values each of which is stored with 8 bits of precision. Then to transmit the whole image directly would cost $24N$ bits. Now suppose we first run K -means on the image data, and then instead of transmitting the original pixel intensity vectors we transmit the identity of the nearest vector μ_k . Because there are K such vectors, this requires $\log_2 K$ bits per pixel. We must also transmit the K code book vectors μ_k , which requires $24K$ bits, and so the total number of bits required to transmit the image is $24K + N \log_2 K$ (rounding up to the nearest integer). The original image shown in Figure 9.3 has $240 \times 180 = 43,200$ pixels and so requires $24 \times 43,200 = 1,036,800$ bits to transmit directly. By comparison, the compressed images require 43,248 bits ($K = 2$), 86,472 bits ($K = 3$), and 173,040 bits ($K = 10$), respectively, to transmit. These represent compression ratios compared to the original image of 4.2%, 8.3%, and 16.7%, respectively. We see that there is a trade-off between degree of compression and image quality. Note that our aim in this example is to illustrate the K -means algorithm. If we had been aiming to produce a good image compressor, then it would be more fruitful to consider small blocks of adjacent pixels, for instance 5×5 , and thereby exploit the correlations that exist in natural images between nearby pixels.

9.2. Mixtures of Gaussians

In Section 2.3.9 we motivated the Gaussian mixture model as a simple linear superposition of Gaussian components, aimed at providing a richer class of density models than the single Gaussian. We now turn to a formulation of Gaussian mixtures in terms of discrete *latent* variables. This will provide us with a deeper insight into this important distribution, and will also serve to motivate the expectation-maximization algorithm.

Recall from (2.188) that the Gaussian mixture distribution can be written as a linear superposition of Gaussians in the form

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \quad (9.7)$$

Let us introduce a K -dimensional binary random variable \mathbf{z} having a 1-of- K representation in which a particular element z_k is equal to 1 and all other elements are equal to 0. The values of z_k therefore satisfy $z_k \in \{0, 1\}$ and $\sum_k z_k = 1$, and we see that there are K possible states for the vector \mathbf{z} according to which element is nonzero. We shall define the joint distribution $p(\mathbf{x}, \mathbf{z})$ in terms of a marginal distribution $p(\mathbf{z})$ and a conditional distribution $p(\mathbf{x} | \mathbf{z})$, corresponding to the graphical model in Figure 9.4. The marginal distribution over \mathbf{z} is specified in terms of the mixing coefficients π_k , such that

$$p(z_k = 1) = \pi_k$$

Figure 9.4 Graphical representation of a mixture model, in which the joint distribution is expressed in the form $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$.



where the parameters $\{\pi_k\}$ must satisfy

$$0 \leq \pi_k \leq 1 \quad (9.8)$$

together with

$$\sum_{k=1}^K \pi_k = 1 \quad (9.9)$$

in order to be valid probabilities. Because \mathbf{z} uses a 1-of- K representation, we can also write this distribution in the form

$$p(\mathbf{z}) = \prod_{k=1}^K \pi_k^{z_k}. \quad (9.10)$$

Similarly, the conditional distribution of \mathbf{x} given a particular value for \mathbf{z} is a Gaussian

$$p(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

which can also be written in the form

$$p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^K \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}. \quad (9.11)$$

The joint distribution is given by $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$, and the marginal distribution of \mathbf{x} is then obtained by summing the joint distribution over all possible states of \mathbf{z} to give

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad (9.12)$$

where we have made use of (9.10) and (9.11). Thus the marginal distribution of \mathbf{x} is a Gaussian mixture of the form (9.7). If we have several observations $\mathbf{x}_1, \dots, \mathbf{x}_n$, then, because we have represented the marginal distribution in the form $p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z})$, it follows that for every observed data point \mathbf{x}_n there is a corresponding latent variable \mathbf{z}_n .

We have therefore found an equivalent formulation of the Gaussian mixture involving an explicit latent variable. It might seem that we have not gained much by doing so. However, we are now able to work with the joint distribution $p(\mathbf{x}, \mathbf{z})$

Exercise 9.3

instead of the marginal distribution $p(\mathbf{x})$, and this will lead to significant simplifications, most notably through the introduction of the expectation-maximization (EM) algorithm.

Another quantity that will play an important role is the conditional probability of \mathbf{z} given \mathbf{x} . We shall use $\gamma(z_k)$ to denote $p(z_k = 1|\mathbf{x})$, whose value can be found using Bayes' theorem

$$\begin{aligned}\gamma(z_k) \equiv p(z_k = 1|\mathbf{x}) &= \frac{p(z_k = 1)p(\mathbf{x}|z_k = 1)}{\sum_{j=1}^K p(z_j = 1)p(\mathbf{x}|z_j = 1)} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}.\end{aligned}\quad (9.13)$$

We shall view π_k as the prior probability of $z_k = 1$, and the quantity $\gamma(z_k)$ as the corresponding posterior probability once we have observed \mathbf{x} . As we shall see later, $\gamma(z_k)$ can also be viewed as the *responsibility* that component k takes for ‘explaining’ the observation \mathbf{x} .

Section 8.1.2

We can use the technique of ancestral sampling to generate random samples distributed according to the Gaussian mixture model. To do this, we first generate a value for \mathbf{z} , which we denote $\hat{\mathbf{z}}$, from the marginal distribution $p(\mathbf{z})$ and then generate a value for \mathbf{x} from the conditional distribution $p(\mathbf{x}|\hat{\mathbf{z}})$. Techniques for sampling from standard distributions are discussed in Chapter 11. We can depict samples from the joint distribution $p(\mathbf{x}, \mathbf{z})$ by plotting points at the corresponding values of \mathbf{x} and then colouring them according to the value of \mathbf{z} , in other words according to which Gaussian component was responsible for generating them, as shown in Figure 9.5(a). Similarly samples from the marginal distribution $p(\mathbf{x})$ are obtained by taking the samples from the joint distribution and ignoring the values of \mathbf{z} . These are illustrated in Figure 9.5(b) by plotting the \mathbf{x} values without any coloured labels.

We can also use this synthetic data set to illustrate the ‘responsibilities’ by evaluating, for every data point, the posterior probability for each component in the mixture distribution from which this data set was generated. In particular, we can represent the value of the responsibilities $\gamma(z_{nk})$ associated with data point \mathbf{x}_n by plotting the corresponding point using proportions of red, blue, and green ink given by $\gamma(z_{nk})$ for $k = 1, 2, 3$, respectively, as shown in Figure 9.5(c). So, for instance, a data point for which $\gamma(z_{n1}) = 1$ will be coloured red, whereas one for which $\gamma(z_{n2}) = \gamma(z_{n3}) = 0.5$ will be coloured with equal proportions of blue and green ink and so will appear cyan. This should be compared with Figure 9.5(a) in which the data points were labelled using the true identity of the component from which they were generated.

9.2.1 Maximum likelihood

Suppose we have a data set of observations $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, and we wish to model this data using a mixture of Gaussians. We can represent this data set as an $N \times D$

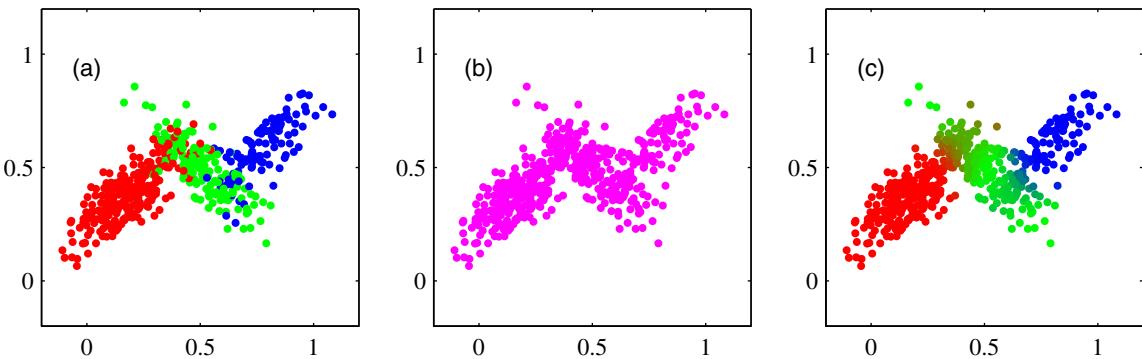


Figure 9.5 Example of 500 points drawn from the mixture of 3 Gaussians shown in Figure 2.23. (a) Samples from the joint distribution $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$ in which the three states of \mathbf{z} , corresponding to the three components of the mixture, are depicted in red, green, and blue, and (b) the corresponding samples from the marginal distribution $p(\mathbf{x})$, which is obtained by simply ignoring the values of \mathbf{z} and just plotting the \mathbf{x} values. The data set in (a) is said to be *complete*, whereas that in (b) is *incomplete*. (c) The same samples in which the colours represent the value of the responsibilities $\gamma(z_{nk})$ associated with data point \mathbf{x}_n , obtained by plotting the corresponding point using proportions of red, blue, and green ink given by $\gamma(z_{nk})$ for $k = 1, 2, 3$, respectively

matrix \mathbf{X} in which the n^{th} row is given by \mathbf{x}_n^T . Similarly, the corresponding latent variables will be denoted by an $N \times K$ matrix \mathbf{Z} with rows \mathbf{z}_n^T . If we assume that the data points are drawn independently from the distribution, then we can express the Gaussian mixture model for this i.i.d. data set using the graphical representation shown in Figure 9.6. From (9.7) the log of the likelihood function is given by

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}. \quad (9.14)$$

Before discussing how to maximize this function, it is worth emphasizing that there is a significant problem associated with the maximum likelihood framework applied to Gaussian mixture models, due to the presence of singularities. For simplicity, consider a Gaussian mixture whose components have covariance matrices given by $\boldsymbol{\Sigma}_k = \sigma_k^2 \mathbf{I}$, where \mathbf{I} is the unit matrix, although the conclusions will hold for general covariance matrices. Suppose that one of the components of the mixture model, let us say the j^{th} component, has its mean $\boldsymbol{\mu}_j$ exactly equal to one of the data

Figure 9.6 Graphical representation of a Gaussian mixture model for a set of N i.i.d. data points $\{\mathbf{x}_n\}$, with corresponding latent points $\{\mathbf{z}_n\}$, where $n = 1, \dots, N$.

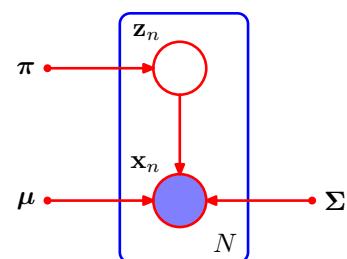
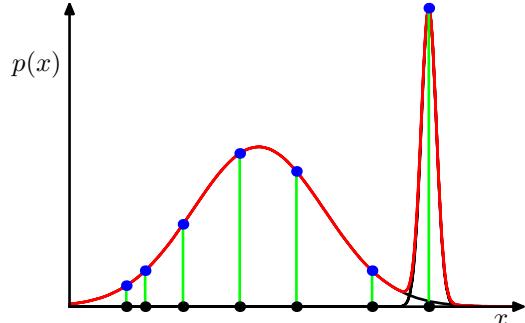


Figure 9.7 Illustration of how singularities in the likelihood function arise with mixtures of Gaussians. This should be compared with the case of a single Gaussian shown in Figure 1.14 for which no singularities arise.



points so that $\mu_j = \mathbf{x}_n$ for some value of n . This data point will then contribute a term in the likelihood function of the form

$$\mathcal{N}(\mathbf{x}_n | \mathbf{x}_n, \sigma_j^2 \mathbf{I}) = \frac{1}{(2\pi)^{1/2}} \frac{1}{\sigma_j}. \quad (9.15)$$

If we consider the limit $\sigma_j \rightarrow 0$, then we see that this term goes to infinity and so the log likelihood function will also go to infinity. Thus the maximization of the log likelihood function is not a well posed problem because such singularities will always be present and will occur whenever one of the Gaussian components ‘collapses’ onto a specific data point. Recall that this problem did not arise in the case of a single Gaussian distribution. To understand the difference, note that if a single Gaussian collapses onto a data point it will contribute multiplicative factors to the likelihood function arising from the other data points and these factors will go to zero exponentially fast, giving an overall likelihood that goes to zero rather than infinity. However, once we have (at least) two components in the mixture, one of the components can have a finite variance and therefore assign finite probability to all of the data points while the other component can shrink onto one specific data point and thereby contribute an ever increasing additive value to the log likelihood. This is illustrated in Figure 9.7. These singularities provide another example of the severe over-fitting that can occur in a maximum likelihood approach. We shall see that this difficulty does not occur if we adopt a Bayesian approach. For the moment, however, we simply note that in applying maximum likelihood to Gaussian mixture models we must take steps to avoid finding such pathological solutions and instead seek local maxima of the likelihood function that are well behaved. We can hope to avoid the singularities by using suitable heuristics, for instance by detecting when a Gaussian component is collapsing and resetting its mean to a randomly chosen value while also resetting its covariance to some large value, and then continuing with the optimization.

A further issue in finding maximum likelihood solutions arises from the fact that for any given maximum likelihood solution, a K -component mixture will have a total of $K!$ equivalent solutions corresponding to the $K!$ ways of assigning K sets of parameters to K components. In other words, for any given (nondegenerate) point in the space of parameter values there will be a further $K! - 1$ additional points all of which give rise to exactly the same distribution. This problem is known as

Section 10.1

identifiability (Casella and Berger, 2002) and is an important issue when we wish to interpret the parameter values discovered by a model. Identifiability will also arise when we discuss models having continuous latent variables in Chapter 12. However, for the purposes of finding a good density model, it is irrelevant because any of the equivalent solutions is as good as any other.

Maximizing the log likelihood function (9.14) for a Gaussian mixture model turns out to be a more complex problem than for the case of a single Gaussian. The difficulty arises from the presence of the summation over k that appears inside the logarithm in (9.14), so that the logarithm function no longer acts directly on the Gaussian. If we set the derivatives of the log likelihood to zero, we will no longer obtain a closed form solution, as we shall see shortly.

One approach is to apply gradient-based optimization techniques (Fletcher, 1987; Nocedal and Wright, 1999; Bishop and Nabney, 2008). Although gradient-based techniques are feasible, and indeed will play an important role when we discuss mixture density networks in Chapter 5, we now consider an alternative approach known as the EM algorithm which has broad applicability and which will lay the foundations for a discussion of variational inference techniques in Chapter 10.

9.2.2 EM for Gaussian mixtures

An elegant and powerful method for finding maximum likelihood solutions for models with latent variables is called the *expectation-maximization* algorithm, or *EM* algorithm (Dempster *et al.*, 1977; McLachlan and Krishnan, 1997). Later we shall give a general treatment of EM, and we shall also show how EM can be generalized to obtain the variational inference framework. Initially, we shall motivate the EM algorithm by giving a relatively informal treatment in the context of the Gaussian mixture model. We emphasize, however, that EM has broad applicability, and indeed it will be encountered in the context of a variety of different models in this book.

Let us begin by writing down the conditions that must be satisfied at a maximum of the likelihood function. Setting the derivatives of $\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ in (9.14) with respect to the means $\boldsymbol{\mu}_k$ of the Gaussian components to zero, we obtain

$$0 = - \sum_{n=1}^N \underbrace{\frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}} \boldsymbol{\Sigma}_k (\mathbf{x}_n - \boldsymbol{\mu}_k) \gamma(z_{nk}) \quad (9.16)$$

where we have made use of the form (2.43) for the Gaussian distribution. Note that the posterior probabilities, or responsibilities, given by (9.13) appear naturally on the right-hand side. Multiplying by $\boldsymbol{\Sigma}_k^{-1}$ (which we assume to be nonsingular) and rearranging we obtain

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n \quad (9.17)$$

where we have defined

$$N_k = \sum_{n=1}^N \gamma(z_{nk}). \quad (9.18)$$

Section 10.1

We can interpret N_k as the effective number of points assigned to cluster k . Note carefully the form of this solution. We see that the mean μ_k for the k^{th} Gaussian component is obtained by taking a weighted mean of all of the points in the data set, in which the weighting factor for data point \mathbf{x}_n is given by the posterior probability $\gamma(z_{nk})$ that component k was responsible for generating \mathbf{x}_n .

If we set the derivative of $\ln p(\mathbf{X}|\pi, \mu, \Sigma)$ with respect to Σ_k to zero, and follow a similar line of reasoning, making use of the result for the maximum likelihood solution for the covariance matrix of a single Gaussian, we obtain

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk})(\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^T \quad (9.19)$$

which has the same form as the corresponding result for a single Gaussian fitted to the data set, but again with each data point weighted by the corresponding posterior probability and with the denominator given by the effective number of points associated with the corresponding component.

Finally, we maximize $\ln p(\mathbf{X}|\pi, \mu, \Sigma)$ with respect to the mixing coefficients π_k . Here we must take account of the constraint (9.9), which requires the mixing coefficients to sum to one. This can be achieved using a Lagrange multiplier and maximizing the following quantity

$$\ln p(\mathbf{X}|\pi, \mu, \Sigma) + \lambda \left(\sum_{k=1}^K \pi_k - 1 \right) \quad (9.20)$$

which gives

$$0 = \sum_{n=1}^N \frac{\mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_n | \mu_j, \Sigma_j)} + \lambda \quad (9.21)$$

where again we see the appearance of the responsibilities. If we now multiply both sides by π_k and sum over k making use of the constraint (9.9), we find $\lambda = -N$. Using this to eliminate λ and rearranging we obtain

$$\pi_k = \frac{N_k}{N} \quad (9.22)$$

so that the mixing coefficient for the k^{th} component is given by the average responsibility which that component takes for explaining the data points.

It is worth emphasizing that the results (9.17), (9.19), and (9.22) do not constitute a closed-form solution for the parameters of the mixture model because the responsibilities $\gamma(z_{nk})$ depend on those parameters in a complex way through (9.13). However, these results do suggest a simple iterative scheme for finding a solution to the maximum likelihood problem, which as we shall see turns out to be an instance of the EM algorithm for the particular case of the Gaussian mixture model. We first choose some initial values for the means, covariances, and mixing coefficients. Then we alternate between the following two updates that we shall call the E step

Section 2.3.4

Appendix E

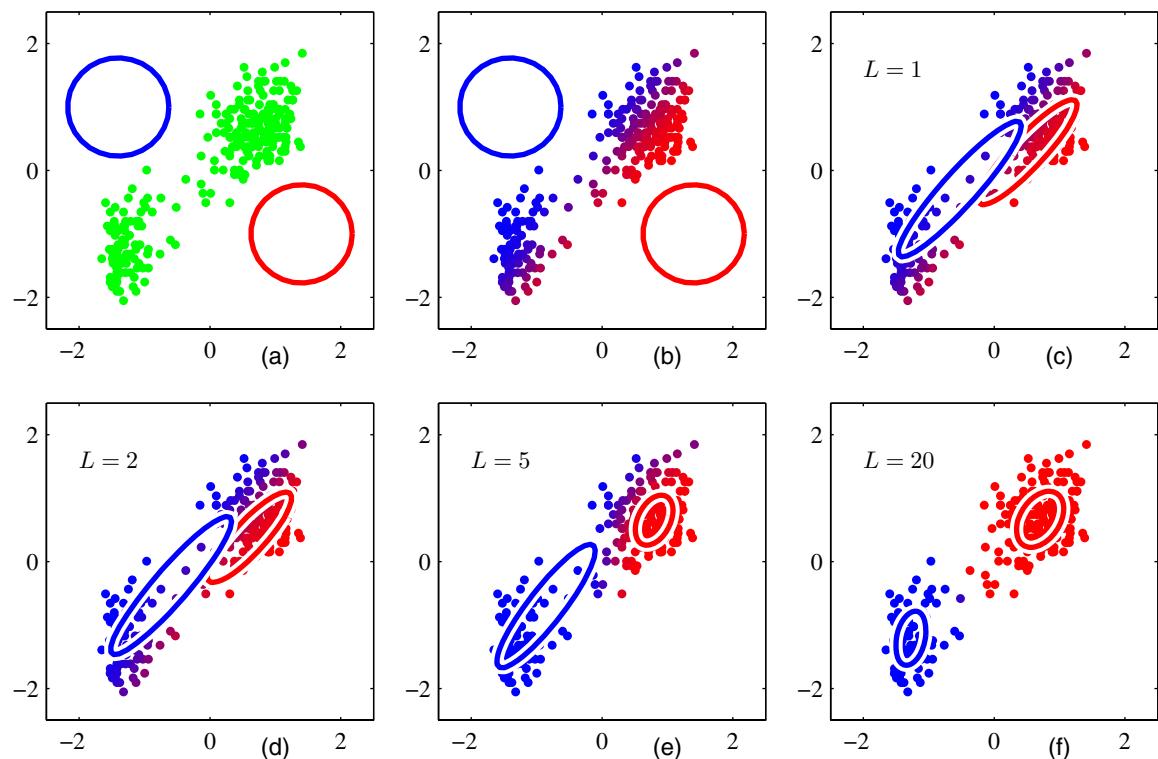


Figure 9.8 Illustration of the EM algorithm using the Old Faithful set as used for the illustration of the K -means algorithm in Figure 9.1. See the text for details.

and the M step, for reasons that will become apparent shortly. In the *expectation* step, or E step, we use the current values for the parameters to evaluate the posterior probabilities, or responsibilities, given by (9.13). We then use these probabilities in the *maximization* step, or M step, to re-estimate the means, covariances, and mixing coefficients using the results (9.17), (9.19), and (9.22). Note that in so doing we first evaluate the new means using (9.17) and then use these new values to find the covariances using (9.19), in keeping with the corresponding result for a single Gaussian distribution. We shall show that each update to the parameters resulting from an E step followed by an M step is guaranteed to increase the log likelihood function. In practice, the algorithm is deemed to have converged when the change in the log likelihood function, or alternatively in the parameters, falls below some threshold. We illustrate the EM algorithm for a mixture of two Gaussians applied to the rescaled Old Faithful data set in Figure 9.8. Here a mixture of two Gaussians is used, with centres initialized using the same values as for the K -means algorithm in Figure 9.1, and with precision matrices initialized to be proportional to the unit matrix. Plot (a) shows the data points in green, together with the initial configuration of the mixture model in which the one standard-deviation contours for the two

Section 9.4

Gaussian components are shown as blue and red circles. Plot (b) shows the result of the initial E step, in which each data point is depicted using a proportion of blue ink equal to the posterior probability of having been generated from the blue component, and a corresponding proportion of red ink given by the posterior probability of having been generated by the red component. Thus, points that have a significant probability for belonging to either cluster appear purple. The situation after the first M step is shown in plot (c), in which the mean of the blue Gaussian has moved to the mean of the data set, weighted by the probabilities of each data point belonging to the blue cluster, in other words it has moved to the centre of mass of the blue ink. Similarly, the covariance of the blue Gaussian is set equal to the covariance of the blue ink. Analogous results hold for the red component. Plots (d), (e), and (f) show the results after 2, 5, and 20 complete cycles of EM, respectively. In plot (f) the algorithm is close to convergence.

Note that the EM algorithm takes many more iterations to reach (approximate) convergence compared with the K -means algorithm, and that each cycle requires significantly more computation. It is therefore common to run the K -means algorithm in order to find a suitable initialization for a Gaussian mixture model that is subsequently adapted using EM. The covariance matrices can conveniently be initialized to the sample covariances of the clusters found by the K -means algorithm, and the mixing coefficients can be set to the fractions of data points assigned to the respective clusters. As with gradient-based approaches for maximizing the log likelihood, techniques must be employed to avoid singularities of the likelihood function in which a Gaussian component collapses onto a particular data point. It should be emphasized that there will generally be multiple local maxima of the log likelihood function, and that EM is not guaranteed to find the largest of these maxima. Because the EM algorithm for Gaussian mixtures plays such an important role, we summarize it below.

EM for Gaussian Mixtures

Given a Gaussian mixture model, the goal is to maximize the likelihood function with respect to the parameters (comprising the means and covariances of the components and the mixing coefficients).

1. Initialize the means μ_k , covariances Σ_k and mixing coefficients π_k , and evaluate the initial value of the log likelihood.
2. **E step.** Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \quad (9.23)$$

3. **M step.** Re-estimate the parameters using the current responsibilities

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n \quad (9.24)$$

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})^T \quad (9.25)$$

$$\pi_k^{\text{new}} = \frac{N_k}{N} \quad (9.26)$$

where

$$N_k = \sum_{n=1}^N \gamma(z_{nk}). \quad (9.27)$$

4. Evaluate the log likelihood

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \quad (9.28)$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to step 2.

9.3. An Alternative View of EM

In this section, we present a complementary view of the EM algorithm that recognizes the key role played by latent variables. We discuss this approach first of all in an abstract setting, and then for illustration we consider once again the case of Gaussian mixtures.

The goal of the EM algorithm is to find maximum likelihood solutions for models having latent variables. We denote the set of all observed data by \mathbf{X} , in which the n^{th} row represents \mathbf{x}_n^T , and similarly we denote the set of all latent variables by \mathbf{Z} , with a corresponding row \mathbf{z}_n^T . The set of all model parameters is denoted by $\boldsymbol{\theta}$, and so the log likelihood function is given by

$$\ln p(\mathbf{X}|\boldsymbol{\theta}) = \ln \left\{ \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) \right\}. \quad (9.29)$$

Note that our discussion will apply equally well to continuous latent variables simply by replacing the sum over \mathbf{Z} with an integral.

A key observation is that the summation over the latent variables appears inside the logarithm. Even if the joint distribution $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ belongs to the exponential

family, the marginal distribution $p(\mathbf{X}|\boldsymbol{\theta})$ typically does not as a result of this summation. The presence of the sum prevents the logarithm from acting directly on the joint distribution, resulting in complicated expressions for the maximum likelihood solution.

Now suppose that, for each observation in \mathbf{X} , we were told the corresponding value of the latent variable \mathbf{Z} . We shall call $\{\mathbf{X}, \mathbf{Z}\}$ the *complete* data set, and we shall refer to the actual observed data \mathbf{X} as *incomplete*, as illustrated in Figure 9.5. The likelihood function for the complete data set simply takes the form $\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$, and we shall suppose that maximization of this complete-data log likelihood function is straightforward.

In practice, however, we are not given the complete data set $\{\mathbf{X}, \mathbf{Z}\}$, but only the incomplete data \mathbf{X} . Our state of knowledge of the values of the latent variables in \mathbf{Z} is given only by the posterior distribution $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})$. Because we cannot use the complete-data log likelihood, we consider instead its expected value under the posterior distribution of the latent variable, which corresponds (as we shall see) to the E step of the EM algorithm. In the subsequent M step, we maximize this expectation. If the current estimate for the parameters is denoted $\boldsymbol{\theta}^{\text{old}}$, then a pair of successive E and M steps gives rise to a revised estimate $\boldsymbol{\theta}^{\text{new}}$. The algorithm is initialized by choosing some starting value for the parameters $\boldsymbol{\theta}_0$. The use of the expectation may seem somewhat arbitrary. However, we shall see the motivation for this choice when we give a deeper treatment of EM in Section 9.4.

In the E step, we use the current parameter values $\boldsymbol{\theta}^{\text{old}}$ to find the posterior distribution of the latent variables given by $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$. We then use this posterior distribution to find the expectation of the complete-data log likelihood evaluated for some general parameter value $\boldsymbol{\theta}$. This expectation, denoted $\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$, is given by

$$\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}). \quad (9.30)$$

In the M step, we determine the revised parameter estimate $\boldsymbol{\theta}^{\text{new}}$ by maximizing this function

$$\boldsymbol{\theta}^{\text{new}} = \arg \max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}). \quad (9.31)$$

Note that in the definition of $\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$, the logarithm acts directly on the joint distribution $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$, and so the corresponding M-step maximization will, by supposition, be tractable.

The general EM algorithm is summarized below. It has the property, as we shall show later, that each cycle of EM will increase the incomplete-data log likelihood (unless it is already at a local maximum).

The General EM Algorithm

Given a joint distribution $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ over observed variables \mathbf{X} and latent variables \mathbf{Z} , governed by parameters $\boldsymbol{\theta}$, the goal is to maximize the likelihood function $p(\mathbf{X}|\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$.

1. Choose an initial setting for the parameters $\boldsymbol{\theta}^{\text{old}}$.

2. **E step** Evaluate $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$.

3. **M step** Evaluate $\boldsymbol{\theta}^{\text{new}}$ given by

$$\boldsymbol{\theta}^{\text{new}} = \arg \max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) \quad (9.32)$$

where

$$\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}). \quad (9.33)$$

4. Check for convergence of either the log likelihood or the parameter values. If the convergence criterion is not satisfied, then let

$$\boldsymbol{\theta}^{\text{old}} \leftarrow \boldsymbol{\theta}^{\text{new}} \quad (9.34)$$

and return to step 2.

Exercise 9.4

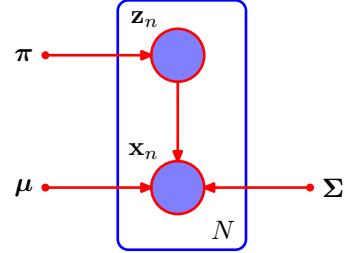
The EM algorithm can also be used to find MAP (maximum posterior) solutions for models in which a prior $p(\boldsymbol{\theta})$ is defined over the parameters. In this case the E step remains the same as in the maximum likelihood case, whereas in the M step the quantity to be maximized is given by $\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) + \ln p(\boldsymbol{\theta})$. Suitable choices for the prior will remove the singularities of the kind illustrated in Figure 9.7.

Here we have considered the use of the EM algorithm to maximize a likelihood function when there are discrete latent variables. However, it can also be applied when the unobserved variables correspond to missing values in the data set. The distribution of the observed values is obtained by taking the joint distribution of all the variables and then marginalizing over the missing ones. EM can then be used to maximize the corresponding likelihood function. We shall show an example of the application of this technique in the context of principal component analysis in Figure 12.11. This will be a valid procedure if the data values are *missing at random*, meaning that the mechanism causing values to be missing does not depend on the unobserved values. In many situations this will not be the case, for instance if a sensor fails to return a value whenever the quantity it is measuring exceeds some threshold.

9.3.1 Gaussian mixtures revisited

We now consider the application of this latent variable view of EM to the specific case of a Gaussian mixture model. Recall that our goal is to maximize the log likelihood function (9.14), which is computed using the observed data set \mathbf{X} , and we saw that this was more difficult than for the case of a single Gaussian distribution due to the presence of the summation over k that occurs inside the logarithm. Suppose then that in addition to the observed data set \mathbf{X} , we were also given the values of the corresponding discrete variables \mathbf{Z} . Recall that Figure 9.5(a) shows a ‘complete’ data set (i.e., one that includes labels showing which component generated each data point) while Figure 9.5(b) shows the corresponding ‘incomplete’ data set. The graphical model for the complete data is shown in Figure 9.9.

Figure 9.9 This shows the same graph as in Figure 9.6 except that we now suppose that the discrete variables \mathbf{z}_n are observed, as well as the data variables \mathbf{x}_n .



Now consider the problem of maximizing the likelihood for the complete data set $\{\mathbf{X}, \mathbf{Z}\}$. From (9.10) and (9.11), this likelihood function takes the form

$$p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \prod_{n=1}^N \prod_{k=1}^K \pi_k^{z_{nk}} \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}} \quad (9.35)$$

where z_{nk} denotes the k^{th} component of \mathbf{z}_n . Taking the logarithm, we obtain

$$\ln p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^N \sum_{k=1}^K z_{nk} \{ \ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \}. \quad (9.36)$$

Comparison with the log likelihood function (9.14) for the incomplete data shows that the summation over k and the logarithm have been interchanged. The logarithm now acts directly on the Gaussian distribution, which itself is a member of the exponential family. Not surprisingly, this leads to a much simpler solution to the maximum likelihood problem, as we now show. Consider first the maximization with respect to the means and covariances. Because \mathbf{z}_n is a K -dimensional vector with all elements equal to 0 except for a single element having the value 1, the complete-data log likelihood function is simply a sum of K independent contributions, one for each mixture component. Thus the maximization with respect to a mean or a covariance is exactly as for a single Gaussian, except that it involves only the subset of data points that are ‘assigned’ to that component. For the maximization with respect to the mixing coefficients, we note that these are coupled for different values of k by virtue of the summation constraint (9.9). Again, this can be enforced using a Lagrange multiplier as before, and leads to the result

$$\pi_k = \frac{1}{N} \sum_{n=1}^N z_{nk} \quad (9.37)$$

so that the mixing coefficients are equal to the fractions of data points assigned to the corresponding components.

Thus we see that the complete-data log likelihood function can be maximized trivially in closed form. In practice, however, we do not have values for the latent variables so, as discussed earlier, we consider the expectation, with respect to the posterior distribution of the latent variables, of the complete-data log likelihood.

Using (9.10) and (9.11) together with Bayes' theorem, we see that this posterior distribution takes the form

$$p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) \propto \prod_{n=1}^N \prod_{k=1}^K [\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)]^{z_{nk}}. \quad (9.38)$$

and hence factorizes over n so that under the posterior distribution the $\{\mathbf{z}_n\}$ are independent. This is easily verified by inspection of the directed graph in Figure 9.6 and making use of the d-separation criterion. The expected value of the indicator variable z_{nk} under this posterior distribution is then given by

$$\begin{aligned} \mathbb{E}[z_{nk}] &= \frac{\sum z_{nk} [\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)]^{z_{nk}}}{\sum_{z_{nj}} [\pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)]^{z_{nj}}} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} = \gamma(z_{nk}) \end{aligned} \quad (9.39)$$

which is just the responsibility of component k for data point \mathbf{x}_n . The expected value of the complete-data log likelihood function is therefore given by

$$\mathbb{E}_{\mathbf{Z}}[\ln p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})] = \sum_{n=1}^N \sum_{k=1}^K \gamma(z_{nk}) \{\ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\}. \quad (9.40)$$

We can now proceed as follows. First we choose some initial values for the parameters $\boldsymbol{\mu}^{\text{old}}$, $\boldsymbol{\Sigma}^{\text{old}}$ and $\boldsymbol{\pi}^{\text{old}}$, and use these to evaluate the responsibilities (the E step). We then keep the responsibilities fixed and maximize (9.40) with respect to $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and π_k (the M step). This leads to closed form solutions for $\boldsymbol{\mu}^{\text{new}}$, $\boldsymbol{\Sigma}^{\text{new}}$ and $\boldsymbol{\pi}^{\text{new}}$ given by (9.17), (9.19), and (9.22) as before. This is precisely the EM algorithm for Gaussian mixtures as derived earlier. We shall gain more insight into the role of the expected complete-data log likelihood function when we give a proof of convergence of the EM algorithm in Section 9.4.

9.3.2 Relation to K-means

Comparison of the K -means algorithm with the EM algorithm for Gaussian mixtures shows that there is a close similarity. Whereas the K -means algorithm performs a *hard* assignment of data points to clusters, in which each data point is associated uniquely with one cluster, the EM algorithm makes a *soft* assignment based on the posterior probabilities. In fact, we can derive the K -means algorithm as a particular limit of EM for Gaussian mixtures as follows.

Consider a Gaussian mixture model in which the covariance matrices of the mixture components are given by $\epsilon \mathbf{I}$, where ϵ is a variance parameter that is shared

by all of the components, and \mathbf{I} is the identity matrix, so that

$$p(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k) = \frac{1}{(2\pi\epsilon)^{1/2}} \exp \left\{ -\frac{1}{2\epsilon} \|\mathbf{x} - \boldsymbol{\mu}_k\|^2 \right\}. \quad (9.41)$$

We now consider the EM algorithm for a mixture of K Gaussians of this form in which we treat ϵ as a fixed constant, instead of a parameter to be re-estimated. From (9.13) the posterior probabilities, or responsibilities, for a particular data point \mathbf{x}_n , are given by

$$\gamma(z_{nk}) = \frac{\pi_k \exp \{-\|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 / 2\epsilon\}}{\sum_j \pi_j \exp \{-\|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 / 2\epsilon\}}. \quad (9.42)$$

If we consider the limit $\epsilon \rightarrow 0$, we see that in the denominator the term for which $\|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2$ is smallest will go to zero most slowly, and hence the responsibilities $\gamma(z_{nk})$ for the data point \mathbf{x}_n all go to zero except for term j , for which the responsibility $\gamma(z_{nj})$ will go to unity. Note that this holds independently of the values of the π_k so long as none of the π_k is zero. Thus, in this limit, we obtain a hard assignment of data points to clusters, just as in the K -means algorithm, so that $\gamma(z_{nk}) \rightarrow r_{nk}$ where r_{nk} is defined by (9.2). Each data point is thereby assigned to the cluster having the closest mean.

The EM re-estimation equation for the $\boldsymbol{\mu}_k$, given by (9.17), then reduces to the K -means result (9.4). Note that the re-estimation formula for the mixing coefficients (9.22) simply re-sets the value of π_k to be equal to the fraction of data points assigned to cluster k , although these parameters no longer play an active role in the algorithm.

Finally, in the limit $\epsilon \rightarrow 0$ the expected complete-data log likelihood, given by (9.40), becomes

$$\mathbb{E}_{\mathbf{Z}}[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \Sigma, \boldsymbol{\pi})] \rightarrow -\frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 + \text{const.} \quad (9.43)$$

Thus we see that in this limit, maximizing the expected complete-data log likelihood is equivalent to minimizing the distortion measure J for the K -means algorithm given by (9.1).

Note that the K -means algorithm does not estimate the covariances of the clusters but only the cluster means. A hard-assignment version of the Gaussian mixture model with general covariance matrices, known as the *elliptical K-means* algorithm, has been considered by Sung and Poggio (1994).

9.3.3 Mixtures of Bernoulli distributions

So far in this chapter, we have focussed on distributions over continuous variables described by mixtures of Gaussians. As a further example of mixture modelling, and to illustrate the EM algorithm in a different context, we now discuss mixtures of discrete binary variables described by Bernoulli distributions. This model is also known as *latent class analysis* (Lazarsfeld and Henry, 1968; McLachlan and Peel, 2000). As well as being of practical importance in its own right, our discussion of Bernoulli mixtures will also lay the foundation for a consideration of hidden Markov models over discrete variables.

Consider a set of D binary variables x_i , where $i = 1, \dots, D$, each of which is governed by a Bernoulli distribution with parameter μ_i , so that

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{i=1}^D \mu_i^{x_i} (1 - \mu_i)^{(1-x_i)} \quad (9.44)$$

where $\mathbf{x} = (x_1, \dots, x_D)^\top$ and $\boldsymbol{\mu} = (\mu_1, \dots, \mu_D)^\top$. We see that the individual variables x_i are independent, given $\boldsymbol{\mu}$. The mean and covariance of this distribution are easily seen to be

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu} \quad (9.45)$$

$$\text{cov}[\mathbf{x}] = \text{diag}\{\mu_i(1 - \mu_i)\}. \quad (9.46)$$

Now let us consider a finite mixture of these distributions given by

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{k=1}^K \pi_k p(\mathbf{x}|\boldsymbol{\mu}_k) \quad (9.47)$$

where $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K\}$, $\boldsymbol{\pi} = \{\pi_1, \dots, \pi_K\}$, and

$$p(\mathbf{x}|\boldsymbol{\mu}_k) = \prod_{i=1}^D \mu_{ki}^{x_i} (1 - \mu_{ki})^{(1-x_i)}. \quad (9.48)$$

Exercise 9.12

The mean and covariance of this mixture distribution are given by

$$\mathbb{E}[\mathbf{x}] = \sum_{k=1}^K \pi_k \boldsymbol{\mu}_k \quad (9.49)$$

$$\text{cov}[\mathbf{x}] = \sum_{k=1}^K \pi_k \left\{ \boldsymbol{\Sigma}_k + \boldsymbol{\mu}_k \boldsymbol{\mu}_k^\top \right\} - \mathbb{E}[\mathbf{x}] \mathbb{E}[\mathbf{x}]^\top \quad (9.50)$$

where $\boldsymbol{\Sigma}_k = \text{diag}\{\mu_{ki}(1 - \mu_{ki})\}$. Because the covariance matrix $\text{cov}[\mathbf{x}]$ is no longer diagonal, the mixture distribution can capture correlations between the variables, unlike a single Bernoulli distribution.

If we are given a data set $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ then the log likelihood function for this model is given by

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k) \right\}. \quad (9.51)$$

Again we see the appearance of the summation inside the logarithm, so that the maximum likelihood solution no longer has closed form.

We now derive the EM algorithm for maximizing the likelihood function for the mixture of Bernoulli distributions. To do this, we first introduce an explicit latent

variable \mathbf{z} associated with each instance of \mathbf{x} . As in the case of the Gaussian mixture, $\mathbf{z} = (z_1, \dots, z_K)^T$ is a binary K -dimensional variable having a single component equal to 1, with all other components equal to 0. We can then write the conditional distribution of \mathbf{x} , given the latent variable, as

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu}) = \prod_{k=1}^K p(\mathbf{x}|\boldsymbol{\mu}_k)^{z_k} \quad (9.52)$$

while the prior distribution for the latent variables is the same as for the mixture of Gaussians model, so that

$$p(\mathbf{z}|\boldsymbol{\pi}) = \prod_{k=1}^K \pi_k^{z_k}. \quad (9.53)$$

If we form the product of $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu})$ and $p(\mathbf{z}|\boldsymbol{\pi})$ and then marginalize over \mathbf{z} , then we recover (9.47).

In order to derive the EM algorithm, we first write down the complete-data log likelihood function, which is given by

$$\begin{aligned} \ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \boldsymbol{\pi}) &= \sum_{n=1}^N \sum_{k=1}^K z_{nk} \left\{ \ln \pi_k \right. \\ &\quad \left. + \sum_{i=1}^D [x_{ni} \ln \mu_{ki} + (1 - x_{ni}) \ln(1 - \mu_{ki})] \right\} \end{aligned} \quad (9.54)$$

where $\mathbf{X} = \{\mathbf{x}_n\}$ and $\mathbf{Z} = \{\mathbf{z}_n\}$. Next we take the expectation of the complete-data log likelihood with respect to the posterior distribution of the latent variables to give

$$\begin{aligned} \mathbb{E}_{\mathbf{Z}}[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \boldsymbol{\pi})] &= \sum_{n=1}^N \sum_{k=1}^K \gamma(z_{nk}) \left\{ \ln \pi_k \right. \\ &\quad \left. + \sum_{i=1}^D [x_{ni} \ln \mu_{ki} + (1 - x_{ni}) \ln(1 - \mu_{ki})] \right\} \end{aligned} \quad (9.55)$$

where $\gamma(z_{nk}) = \mathbb{E}[z_{nk}]$ is the posterior probability, or responsibility, of component k given data point \mathbf{x}_n . In the E step, these responsibilities are evaluated using Bayes' theorem, which takes the form

$$\begin{aligned} \gamma(z_{nk}) &= \mathbb{E}[z_{nk}] = \frac{\sum_{j=1}^K z_{nj} [\pi_j p(\mathbf{x}_n|\boldsymbol{\mu}_j)]^{z_{nj}}}{\sum_{j=1}^K [\pi_j p(\mathbf{x}_n|\boldsymbol{\mu}_j)]^{z_{nj}}} \\ &= \frac{\pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k)}{\sum_{j=1}^K \pi_j p(\mathbf{x}_n|\boldsymbol{\mu}_j)}. \end{aligned} \quad (9.56)$$

Exercise 9.14

If we consider the sum over n in (9.55), we see that the responsibilities enter only through two terms, which can be written as

$$N_k = \sum_{n=1}^N \gamma(z_{nk}) \quad (9.57)$$

$$\bar{\mathbf{x}}_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n \quad (9.58)$$

where N_k is the effective number of data points associated with component k . In the M step, we maximize the expected complete-data log likelihood with respect to the parameters μ_k and π . If we set the derivative of (9.55) with respect to μ_k equal to zero and rearrange the terms, we obtain

$$\mu_k = \bar{\mathbf{x}}_k. \quad (9.59)$$

We see that this sets the mean of component k equal to a weighted mean of the data, with weighting coefficients given by the responsibilities that component k takes for data points. For the maximization with respect to π_k , we need to introduce a Lagrange multiplier to enforce the constraint $\sum_k \pi_k = 1$. Following analogous steps to those used for the mixture of Gaussians, we then obtain

$$\pi_k = \frac{N_k}{N} \quad (9.60)$$

which represents the intuitively reasonable result that the mixing coefficient for component k is given by the effective fraction of points in the data set explained by that component.

Note that in contrast to the mixture of Gaussians, there are no singularities in which the likelihood function goes to infinity. This can be seen by noting that the likelihood function is bounded above because $0 \leq p(\mathbf{x}_n | \mu_k) \leq 1$. There exist singularities at which the likelihood function goes to zero, but these will not be found by EM provided it is not initialized to a pathological starting point, because the EM algorithm always increases the value of the likelihood function, until a local maximum is found. We illustrate the Bernoulli mixture model in Figure 9.10 by using it to model handwritten digits. Here the digit images have been turned into binary vectors by setting all elements whose values exceed 0.5 to 1 and setting the remaining elements to 0. We now fit a data set of $N = 600$ such digits, comprising the digits ‘2’, ‘3’, and ‘4’, with a mixture of $K = 3$ Bernoulli distributions by running 10 iterations of the EM algorithm. The mixing coefficients were initialized to $\pi_k = 1/K$, and the parameters μ_{kj} were set to random values chosen uniformly in the range $(0.25, 0.75)$ and then normalized to satisfy the constraint that $\sum_j \mu_{kj} = 1$. We see that a mixture of 3 Bernoulli distributions is able to find the three clusters in the data set corresponding to the different digits.

The conjugate prior for the parameters of a Bernoulli distribution is given by the beta distribution, and we have seen that a beta prior is equivalent to introducing

Exercise 9.15

Exercise 9.16

Exercise 9.17

Section 9.4

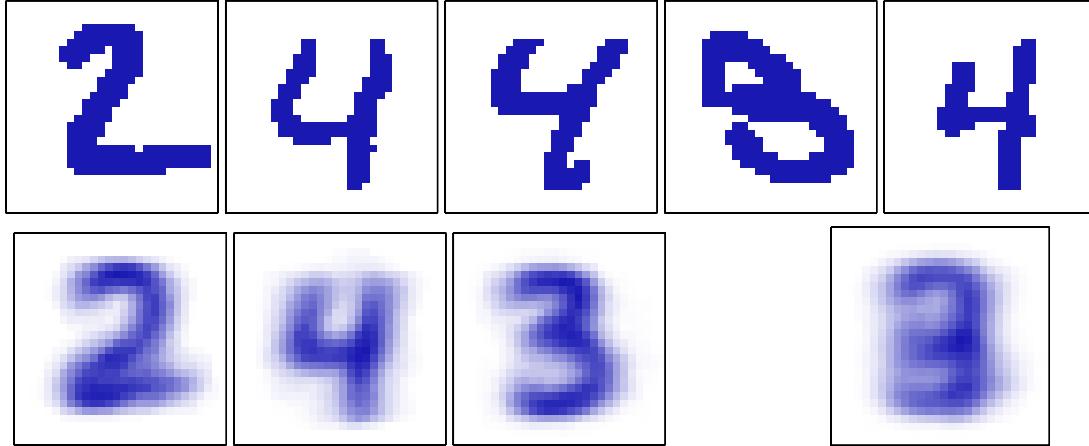


Figure 9.10 Illustration of the Bernoulli mixture model in which the top row shows examples from the digits data set after converting the pixel values from grey scale to binary using a threshold of 0.5. On the bottom row the first three images show the parameters μ_{ki} for each of the three components in the mixture model. As a comparison, we also fit the same data set using a single multivariate Bernoulli distribution, again using maximum likelihood. This amounts to simply averaging the counts in each pixel and is shown by the right-most image on the bottom row.

Section 2.1.1

Exercise 9.18

Exercise 9.19

additional effective observations of \mathbf{x} . We can similarly introduce priors into the Bernoulli mixture model, and use EM to maximize the posterior probability distributions.

It is straightforward to extend the analysis of Bernoulli mixtures to the case of multinomial binary variables having $M > 2$ states by making use of the discrete distribution (2.26). Again, we can introduce Dirichlet priors over the model parameters if desired.

9.3.4 EM for Bayesian linear regression

As a third example of the application of EM, we return to the evidence approximation for Bayesian linear regression. In Section 3.5.2, we obtained the re-estimation equations for the hyperparameters α and β by evaluation of the evidence and then setting the derivatives of the resulting expression to zero. We now turn to an alternative approach for finding α and β based on the EM algorithm. Recall that our goal is to maximize the evidence function $p(\mathbf{t}|\alpha, \beta)$ given by (3.77) with respect to α and β . Because the parameter vector \mathbf{w} is marginalized out, we can regard it as a latent variable, and hence we can optimize this marginal likelihood function using EM. In the E step, we compute the posterior distribution of \mathbf{w} given the current setting of the parameters α and β and then use this to find the expected complete-data log likelihood. In the M step, we maximize this quantity with respect to α and β . We have already derived the posterior distribution of \mathbf{w} because this is given by (3.49). The complete-data log likelihood function is then given by

$$\ln p(\mathbf{t}, \mathbf{w}|\alpha, \beta) = \ln p(\mathbf{t}|\mathbf{w}, \beta) + \ln p(\mathbf{w}|\alpha) \quad (9.61)$$

where the likelihood $p(\mathbf{t}|\mathbf{w}, \beta)$ and the prior $p(\mathbf{w}|\alpha)$ are given by (3.10) and (3.52), respectively, and $y(\mathbf{x}, \mathbf{w})$ is given by (3.3). Taking the expectation with respect to the posterior distribution of \mathbf{w} then gives

$$\begin{aligned}\mathbb{E} [\ln p(\mathbf{t}, \mathbf{w}|\alpha, \beta)] &= \frac{M}{2} \ln \left(\frac{\alpha}{2\pi} \right) - \frac{\alpha}{2} \mathbb{E} [\mathbf{w}^T \mathbf{w}] + \frac{N}{2} \ln \left(\frac{\beta}{2\pi} \right) \\ &\quad - \frac{\beta}{2} \sum_{n=1}^N \mathbb{E} [(t_n - \mathbf{w}^T \boldsymbol{\phi}_n)^2].\end{aligned}\quad (9.62)$$

Setting the derivatives with respect to α to zero, we obtain the M step re-estimation equation

$$\alpha = \frac{M}{\mathbb{E} [\mathbf{w}^T \mathbf{w}]} = \frac{M}{\mathbf{m}_N^T \mathbf{m}_N + \text{Tr}(\mathbf{S}_N)}.\quad (9.63)$$

Exercise 9.20

An analogous result holds for β .

Note that this re-estimation equation takes a slightly different form from the corresponding result (3.92) derived by direct evaluation of the evidence function. However, they each involve computation and inversion (or eigen decomposition) of an $M \times M$ matrix and hence will have comparable computational cost per iteration.

These two approaches to determining α should of course converge to the same result (assuming they find the same local maximum of the evidence function). This can be verified by first noting that the quantity γ is defined by

$$\gamma = M - \alpha \sum_{i=1}^M \frac{1}{\lambda_i + \alpha} = M - \alpha \text{Tr}(\mathbf{S}_N).\quad (9.64)$$

At a stationary point of the evidence function, the re-estimation equation (3.92) will be self-consistently satisfied, and hence we can substitute for γ to give

$$\alpha \mathbf{m}_N^T \mathbf{m}_N = \gamma = M - \alpha \text{Tr}(\mathbf{S}_N)\quad (9.65)$$

and solving for α we obtain (9.63), which is precisely the EM re-estimation equation.

As a final example, we consider a closely related model, namely the relevance vector machine for regression discussed in Section 7.2.1. There we used direct maximization of the marginal likelihood to derive re-estimation equations for the hyperparameters α and β . Here we consider an alternative approach in which we view the weight vector \mathbf{w} as a latent variable and apply the EM algorithm. The E step involves finding the posterior distribution over the weights, and this is given by (7.81). In the M step we maximize the expected complete-data log likelihood, which is defined by

$$\mathbb{E}_{\mathbf{w}} [\ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) p(\mathbf{w}|\alpha)]\quad (9.66)$$

where the expectation is taken with respect to the posterior distribution computed using the ‘old’ parameter values. To compute the new parameter values we maximize with respect to α and β to give

Exercise 9.22

$$\alpha_i^{\text{new}} = \frac{1}{m_i^2 + \Sigma_{ii}} \quad (9.67)$$

$$(\beta^{\text{new}})^{-1} = \frac{\|\mathbf{t} - \Phi\mathbf{m}_N\|^2 + \beta^{-1} \sum_i \gamma_i}{N} \quad (9.68)$$

These re-estimation equations are formally equivalent to those obtained by direct maximization.

Exercise 9.23

9.4. The EM Algorithm in General

The *expectation maximization* algorithm, or EM algorithm, is a general technique for finding maximum likelihood solutions for probabilistic models having latent variables (Dempster *et al.*, 1977; McLachlan and Krishnan, 1997). Here we give a very general treatment of the EM algorithm and in the process provide a proof that the EM algorithm derived heuristically in Sections 9.2 and 9.3 for Gaussian mixtures does indeed maximize the likelihood function (Csiszár and Tusnády, 1984; Hathaway, 1986; Neal and Hinton, 1999). Our discussion will also form the basis for the derivation of the variational inference framework.

Consider a probabilistic model in which we collectively denote all of the observed variables by \mathbf{X} and all of the hidden variables by \mathbf{Z} . The joint distribution $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ is governed by a set of parameters denoted $\boldsymbol{\theta}$. Our goal is to maximize the likelihood function that is given by

$$p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}). \quad (9.69)$$

Here we are assuming \mathbf{Z} is discrete, although the discussion is identical if \mathbf{Z} comprises continuous variables or a combination of discrete and continuous variables, with summation replaced by integration as appropriate.

We shall suppose that direct optimization of $p(\mathbf{X}|\boldsymbol{\theta})$ is difficult, but that optimization of the complete-data likelihood function $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ is significantly easier. Next we introduce a distribution $q(\mathbf{Z})$ defined over the latent variables, and we observe that, for any choice of $q(\mathbf{Z})$, the following decomposition holds

$$\ln p(\mathbf{X}|\boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + \text{KL}(q||p) \quad (9.70)$$

where we have defined

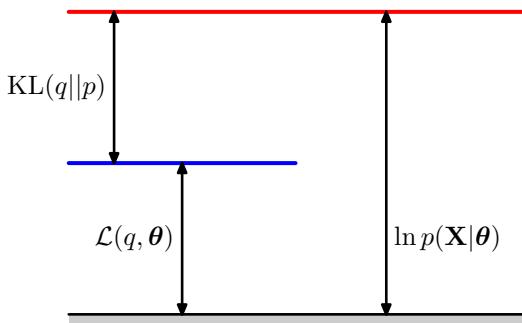
$$\mathcal{L}(q, \boldsymbol{\theta}) = \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})}{q(\mathbf{Z})} \right\} \quad (9.71)$$

$$\text{KL}(q||p) = - \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})}{q(\mathbf{Z})} \right\}. \quad (9.72)$$

Note that $\mathcal{L}(q, \boldsymbol{\theta})$ is a functional (see Appendix D for a discussion of functionals) of the distribution $q(\mathbf{Z})$, and a function of the parameters $\boldsymbol{\theta}$. It is worth studying

Section 10.1

Figure 9.11 Illustration of the decomposition given by (9.70), which holds for any choice of distribution $q(\mathbf{Z})$. Because the Kullback-Leibler divergence satisfies $\text{KL}(q||p) \geq 0$, we see that the quantity $\mathcal{L}(q, \theta)$ is a lower bound on the log likelihood function $\ln p(\mathbf{X}|\theta)$.



carefully the forms of the expressions (9.71) and (9.72), and in particular noting that they differ in sign and also that $\mathcal{L}(q, \theta)$ contains the joint distribution of \mathbf{X} and \mathbf{Z} while $\text{KL}(q||p)$ contains the conditional distribution of \mathbf{Z} given \mathbf{X} . To verify the decomposition (9.70), we first make use of the product rule of probability to give

$$\ln p(\mathbf{X}, \mathbf{Z}|\theta) = \ln p(\mathbf{Z}|\mathbf{X}, \theta) + \ln p(\mathbf{X}|\theta) \quad (9.73)$$

which we then substitute into the expression for $\mathcal{L}(q, \theta)$. This gives rise to two terms, one of which cancels $\text{KL}(q||p)$ while the other gives the required log likelihood $\ln p(\mathbf{X}|\theta)$ after noting that $q(\mathbf{Z})$ is a normalized distribution that sums to 1.

From (9.72), we see that $\text{KL}(q||p)$ is the Kullback-Leibler divergence between $q(\mathbf{Z})$ and the posterior distribution $p(\mathbf{Z}|\mathbf{X}, \theta)$. Recall that the Kullback-Leibler divergence satisfies $\text{KL}(q||p) \geq 0$, with equality if, and only if, $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \theta)$. It therefore follows from (9.70) that $\mathcal{L}(q, \theta) \leq \ln p(\mathbf{X}|\theta)$, in other words that $\mathcal{L}(q, \theta)$ is a lower bound on $\ln p(\mathbf{X}|\theta)$. The decomposition (9.70) is illustrated in Figure 9.11.

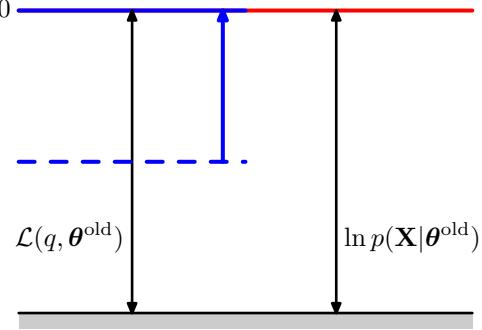
The EM algorithm is a two-stage iterative optimization technique for finding maximum likelihood solutions. We can use the decomposition (9.70) to define the EM algorithm and to demonstrate that it does indeed maximize the log likelihood. Suppose that the current value of the parameter vector is θ^{old} . In the E step, the lower bound $\mathcal{L}(q, \theta^{\text{old}})$ is maximized with respect to $q(\mathbf{Z})$ while holding θ^{old} fixed. The solution to this maximization problem is easily seen by noting that the value of $\ln p(\mathbf{X}|\theta^{\text{old}})$ does not depend on $q(\mathbf{Z})$ and so the largest value of $\mathcal{L}(q, \theta^{\text{old}})$ will occur when the Kullback-Leibler divergence vanishes, in other words when $q(\mathbf{Z})$ is equal to the posterior distribution $p(\mathbf{Z}|\mathbf{X}, \theta^{\text{old}})$. In this case, the lower bound will equal the log likelihood, as illustrated in Figure 9.12.

In the subsequent M step, the distribution $q(\mathbf{Z})$ is held fixed and the lower bound $\mathcal{L}(q, \theta)$ is maximized with respect to θ to give some new value θ^{new} . This will cause the lower bound \mathcal{L} to increase (unless it is already at a maximum), which will necessarily cause the corresponding log likelihood function to increase. Because the distribution q is determined using the old parameter values rather than the new values and is held fixed during the M step, it will not equal the new posterior distribution $p(\mathbf{Z}|\mathbf{X}, \theta^{\text{new}})$, and hence there will be a nonzero KL divergence. The increase in the log likelihood function is therefore greater than the increase in the lower bound, as

Exercise 9.24

Section 1.6.1

Figure 9.12 Illustration of the E step of the EM algorithm. The q distribution is set equal to the posterior distribution for the current parameter values θ^{old} , causing the lower bound to move up to the same value as the log likelihood function, with the KL divergence vanishing.



shown in Figure 9.13. If we substitute $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$ into (9.71), we see that, after the E step, the lower bound takes the form

$$\begin{aligned}\mathcal{L}(q, \boldsymbol{\theta}) &= \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) - \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \\ &= \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) + \text{const}\end{aligned}\quad (9.74)$$

where the constant is simply the negative entropy of the q distribution and is therefore independent of $\boldsymbol{\theta}$. Thus in the M step, the quantity that is being maximized is the expectation of the complete-data log likelihood, as we saw earlier in the case of mixtures of Gaussians. Note that the variable $\boldsymbol{\theta}$ over which we are optimizing appears only inside the logarithm. If the joint distribution $p(\mathbf{Z}, \mathbf{X}|\boldsymbol{\theta})$ comprises a member of the exponential family, or a product of such members, then we see that the logarithm will cancel the exponential and lead to an M step that will be typically much simpler than the maximization of the corresponding incomplete-data log likelihood function $p(\mathbf{X}|\boldsymbol{\theta})$.

The operation of the EM algorithm can also be viewed in the space of parameters, as illustrated schematically in Figure 9.14. Here the red curve depicts the (in-

Figure 9.13 Illustration of the M step of the EM algorithm. The distribution $q(\mathbf{Z})$ is held fixed and the lower bound $\mathcal{L}(q, \boldsymbol{\theta})$ is maximized with respect to the parameter vector $\boldsymbol{\theta}$ to give a revised value $\boldsymbol{\theta}^{\text{new}}$. Because the KL divergence is nonnegative, this causes the log likelihood $\ln p(\mathbf{X}|\boldsymbol{\theta})$ to increase by at least as much as the lower bound does.

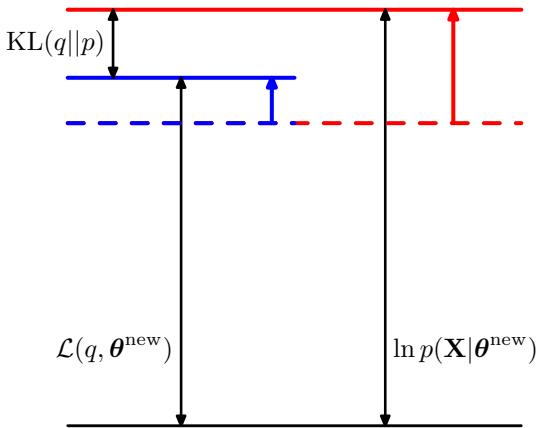
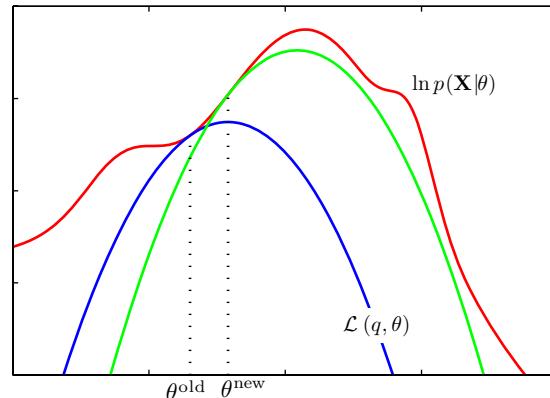


Figure 9.14 The EM algorithm involves alternately computing a lower bound on the log likelihood for the current parameter values and then maximizing this bound to obtain the new parameter values. See the text for a full discussion.



complete data) log likelihood function whose value we wish to maximize. We start with some initial parameter value θ^{old} , and in the first E step we evaluate the posterior distribution over latent variables, which gives rise to a lower bound $\mathcal{L}(\theta, \theta^{(\text{old})})$ whose value equals the log likelihood at $\theta^{(\text{old})}$, as shown by the blue curve. Note that the bound makes a tangential contact with the log likelihood at $\theta^{(\text{old})}$, so that both curves have the same gradient. This bound is a convex function having a unique maximum (for mixture components from the exponential family). In the M step, the bound is maximized giving the value $\theta^{(\text{new})}$, which gives a larger value of log likelihood than $\theta^{(\text{old})}$. The subsequent E step then constructs a bound that is tangential at $\theta^{(\text{new})}$ as shown by the green curve.

For the particular case of an independent, identically distributed data set, \mathbf{X} will comprise N data points $\{\mathbf{x}_n\}$ while \mathbf{Z} will comprise N corresponding latent variables $\{\mathbf{z}_n\}$, where $n = 1, \dots, N$. From the independence assumption, we have $p(\mathbf{X}, \mathbf{Z}) = \prod_n p(\mathbf{x}_n, \mathbf{z}_n)$ and, by marginalizing over the $\{\mathbf{z}_n\}$ we have $p(\mathbf{X}) = \prod_n p(\mathbf{x}_n)$. Using the sum and product rules, we see that the posterior probability that is evaluated in the E step takes the form

$$p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}) = \frac{p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})}{\sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})} = \frac{\prod_{n=1}^N p(\mathbf{x}_n, \mathbf{z}_n|\boldsymbol{\theta})}{\sum_{\mathbf{Z}} \prod_{n=1}^N p(\mathbf{x}_n, \mathbf{z}_n|\boldsymbol{\theta})} = \prod_{n=1}^N p(\mathbf{z}_n|\mathbf{x}_n, \boldsymbol{\theta}) \quad (9.75)$$

and so the posterior distribution also factorizes with respect to n . In the case of the Gaussian mixture model this simply says that the responsibility that each of the mixture components takes for a particular data point \mathbf{x}_n depends only on the value of \mathbf{x}_n and on the parameters $\boldsymbol{\theta}$ of the mixture components, not on the values of the other data points.

We have seen that both the E and the M steps of the EM algorithm are increasing the value of a well-defined bound on the log likelihood function and that the

Exercise 9.25

complete EM cycle will change the model parameters in such a way as to cause the log likelihood to increase (unless it is already at a maximum, in which case the parameters remain unchanged).

We can also use the EM algorithm to maximize the posterior distribution $p(\boldsymbol{\theta}|\mathbf{X})$ for models in which we have introduced a prior $p(\boldsymbol{\theta})$ over the parameters. To see this, we note that as a function of $\boldsymbol{\theta}$, we have $p(\boldsymbol{\theta}|\mathbf{X}) = p(\boldsymbol{\theta}, \mathbf{X})/p(\mathbf{X})$ and so

$$\ln p(\boldsymbol{\theta}|\mathbf{X}) = \ln p(\boldsymbol{\theta}, \mathbf{X}) - \ln p(\mathbf{X}). \quad (9.76)$$

Making use of the decomposition (9.70), we have

$$\begin{aligned} \ln p(\boldsymbol{\theta}|\mathbf{X}) &= \mathcal{L}(q, \boldsymbol{\theta}) + \text{KL}(q\|p) + \ln p(\boldsymbol{\theta}) - \ln p(\mathbf{X}) \\ &\geq \mathcal{L}(q, \boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}) - \ln p(\mathbf{X}). \end{aligned} \quad (9.77)$$

where $\ln p(\mathbf{X})$ is a constant. We can again optimize the right-hand side alternately with respect to q and $\boldsymbol{\theta}$. The optimization with respect to q gives rise to the same E-step equations as for the standard EM algorithm, because q only appears in $\mathcal{L}(q, \boldsymbol{\theta})$. The M-step equations are modified through the introduction of the prior term $\ln p(\boldsymbol{\theta})$, which typically requires only a small modification to the standard maximum likelihood M-step equations.

The EM algorithm breaks down the potentially difficult problem of maximizing the likelihood function into two stages, the E step and the M step, each of which will often prove simpler to implement. Nevertheless, for complex models it may be the case that either the E step or the M step, or indeed both, remain intractable. This leads to two possible extensions of the EM algorithm, as follows.

The *generalized EM*, or *GEM*, algorithm addresses the problem of an intractable M step. Instead of aiming to maximize $\mathcal{L}(q, \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, it seeks instead to change the parameters in such a way as to increase its value. Again, because $\mathcal{L}(q, \boldsymbol{\theta})$ is a lower bound on the log likelihood function, each complete EM cycle of the GEM algorithm is guaranteed to increase the value of the log likelihood (unless the parameters already correspond to a local maximum). One way to exploit the GEM approach would be to use one of the nonlinear optimization strategies, such as the conjugate gradients algorithm, during the M step. Another form of GEM algorithm, known as the *expectation conditional maximization*, or ECM, algorithm, involves making several constrained optimizations within each M step (Meng and Rubin, 1993). For instance, the parameters might be partitioned into groups, and the M step is broken down into multiple steps each of which involves optimizing one of the subset with the remainder held fixed.

We can similarly generalize the E step of the EM algorithm by performing a partial, rather than complete, optimization of $\mathcal{L}(q, \boldsymbol{\theta})$ with respect to $q(\mathbf{Z})$ (Neal and Hinton, 1999). As we have seen, for any given value of $\boldsymbol{\theta}$ there is a unique maximum of $\mathcal{L}(q, \boldsymbol{\theta})$ with respect to $q(\mathbf{Z})$ that corresponds to the posterior distribution $q_{\boldsymbol{\theta}}(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})$ and that for this choice of $q(\mathbf{Z})$ the bound $\mathcal{L}(q, \boldsymbol{\theta})$ is equal to the log likelihood function $\ln p(\mathbf{X}|\boldsymbol{\theta})$. It follows that any algorithm that converges to the global maximum of $\mathcal{L}(q, \boldsymbol{\theta})$ will find a value of $\boldsymbol{\theta}$ that is also a global maximum of the log likelihood $\ln p(\mathbf{X}|\boldsymbol{\theta})$. Provided $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ is a continuous function of $\boldsymbol{\theta}$

then, by continuity, any local maximum of $\mathcal{L}(q, \theta)$ will also be a local maximum of $\ln p(\mathbf{X}|\theta)$.

Consider the case of N independent data points $\mathbf{x}_1, \dots, \mathbf{x}_N$ with corresponding latent variables $\mathbf{z}_1, \dots, \mathbf{z}_N$. The joint distribution $p(\mathbf{X}, \mathbf{Z}|\theta)$ factorizes over the data points, and this structure can be exploited in an incremental form of EM in which at each EM cycle only one data point is processed at a time. In the E step, instead of recomputing the responsibilities for all of the data points, we just re-evaluate the responsibilities for one data point. It might appear that the subsequent M step would require computation involving the responsibilities for all of the data points. However, if the mixture components are members of the exponential family, then the responsibilities enter only through simple sufficient statistics, and these can be updated efficiently. Consider, for instance, the case of a Gaussian mixture, and suppose we perform an update for data point m in which the corresponding old and new values of the responsibilities are denoted $\gamma^{\text{old}}(z_{mk})$ and $\gamma^{\text{new}}(z_{mk})$. In the M step, the required sufficient statistics can be updated incrementally. For instance, for the means the sufficient statistics are defined by (9.17) and (9.18) from which we obtain

$$\boldsymbol{\mu}_k^{\text{new}} = \boldsymbol{\mu}_k^{\text{old}} + \left(\frac{\gamma^{\text{new}}(z_{mk}) - \gamma^{\text{old}}(z_{mk})}{N_k^{\text{new}}} \right) (\mathbf{x}_m - \boldsymbol{\mu}_k^{\text{old}}) \quad (9.78)$$

together with

$$N_k^{\text{new}} = N_k^{\text{old}} + \gamma^{\text{new}}(z_{mk}) - \gamma^{\text{old}}(z_{mk}). \quad (9.79)$$

The corresponding results for the covariances and the mixing coefficients are analogous.

Thus both the E step and the M step take fixed time that is independent of the total number of data points. Because the parameters are revised after each data point, rather than waiting until after the whole data set is processed, this incremental version can converge faster than the batch version. Each E or M step in this incremental algorithm is increasing the value of $\mathcal{L}(q, \theta)$ and, as we have shown above, if the algorithm converges to a local (or global) maximum of $\mathcal{L}(q, \theta)$, this will correspond to a local (or global) maximum of the log likelihood function $\ln p(\mathbf{X}|\theta)$.

Exercises

- 9.1** (*) **www** Consider the K -means algorithm discussed in Section 9.1. Show that as a consequence of there being a finite number of possible assignments for the set of discrete indicator variables r_{nk} , and that for each such assignment there is a unique optimum for the $\{\boldsymbol{\mu}_k\}$, the K -means algorithm must converge after a finite number of iterations.
- 9.2** (*) Apply the Robbins-Monro sequential estimation procedure described in Section 2.3.5 to the problem of finding the roots of the regression function given by the derivatives of J in (9.1) with respect to $\boldsymbol{\mu}_k$. Show that this leads to a stochastic K -means algorithm in which, for each data point \mathbf{x}_n , the nearest prototype $\boldsymbol{\mu}_k$ is updated using (9.5).

9.3 (*) **www** Consider a Gaussian mixture model in which the marginal distribution $p(\mathbf{z})$ for the latent variable is given by (9.10), and the conditional distribution $p(\mathbf{x}|\mathbf{z})$ for the observed variable is given by (9.11). Show that the marginal distribution $p(\mathbf{x})$, obtained by summing $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$ over all possible values of \mathbf{z} , is a Gaussian mixture of the form (9.7).

9.4 (*) Suppose we wish to use the EM algorithm to maximize the posterior distribution over parameters $p(\boldsymbol{\theta}|\mathbf{X})$ for a model containing latent variables, where \mathbf{X} is the observed data set. Show that the E step remains the same as in the maximum likelihood case, whereas in the M step the quantity to be maximized is given by $\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) + \ln p(\boldsymbol{\theta})$ where $\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ is defined by (9.30).

9.5 (*) Consider the directed graph for a Gaussian mixture model shown in Figure 9.6. By making use of the d-separation criterion discussed in Section 8.2, show that the posterior distribution of the latent variables factorizes with respect to the different data points so that

$$p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \prod_{n=1}^N p(\mathbf{z}_n|\mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}). \quad (9.80)$$

9.6 (**) Consider a special case of a Gaussian mixture model in which the covariance matrices $\boldsymbol{\Sigma}_k$ of the components are all constrained to have a common value $\boldsymbol{\Sigma}$. Derive the EM equations for maximizing the likelihood function under such a model.

9.7 (*) **www** Verify that maximization of the complete-data log likelihood (9.36) for a Gaussian mixture model leads to the result that the means and covariances of each component are fitted independently to the corresponding group of data points, and the mixing coefficients are given by the fractions of points in each group.

9.8 (*) **www** Show that if we maximize (9.40) with respect to $\boldsymbol{\mu}_k$ while keeping the responsibilities $\gamma(z_{nk})$ fixed, we obtain the closed form solution given by (9.17).

9.9 (*) Show that if we maximize (9.40) with respect to $\boldsymbol{\Sigma}_k$ and π_k while keeping the responsibilities $\gamma(z_{nk})$ fixed, we obtain the closed form solutions given by (9.19) and (9.22).

9.10 (**) Consider a density model given by a mixture distribution

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k p(\mathbf{x}|k) \quad (9.81)$$

and suppose that we partition the vector \mathbf{x} into two parts so that $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$. Show that the conditional density $p(\mathbf{x}_b|\mathbf{x}_a)$ is itself a mixture distribution and find expressions for the mixing coefficients and for the component densities.

- 9.11** (*) In Section 9.3.2, we obtained a relationship between K means and EM for Gaussian mixtures by considering a mixture model in which all components have covariance $\epsilon\mathbf{I}$. Show that in the limit $\epsilon \rightarrow 0$, maximizing the expected complete-data log likelihood for this model, given by (9.40), is equivalent to minimizing the distortion measure J for the K -means algorithm given by (9.1).

- 9.12** (*) **www** Consider a mixture distribution of the form

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k p(\mathbf{x}|k) \quad (9.82)$$

where the elements of \mathbf{x} could be discrete or continuous or a combination of these. Denote the mean and covariance of $p(\mathbf{x}|k)$ by μ_k and Σ_k , respectively. Show that the mean and covariance of the mixture distribution are given by (9.49) and (9.50).

- 9.13** (**) Using the re-estimation equations for the EM algorithm, show that a mixture of Bernoulli distributions, with its parameters set to values corresponding to a maximum of the likelihood function, has the property that

$$\mathbb{E}[\mathbf{x}] = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \equiv \bar{\mathbf{x}}. \quad (9.83)$$

Hence show that if the parameters of this model are initialized such that all components have the same mean $\mu_k = \hat{\mu}$ for $k = 1, \dots, K$, then the EM algorithm will converge after one iteration, for any choice of the initial mixing coefficients, and that this solution has the property $\mu_k = \bar{\mathbf{x}}$. Note that this represents a degenerate case of the mixture model in which all of the components are identical, and in practice we try to avoid such solutions by using an appropriate initialization.

- 9.14** (*) Consider the joint distribution of latent and observed variables for the Bernoulli distribution obtained by forming the product of $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu})$ given by (9.52) and $p(\mathbf{z}|\boldsymbol{\pi})$ given by (9.53). Show that if we marginalize this joint distribution with respect to \mathbf{z} , then we obtain (9.47).

- 9.15** (*) **www** Show that if we maximize the expected complete-data log likelihood function (9.55) for a mixture of Bernoulli distributions with respect to $\boldsymbol{\mu}_k$, we obtain the M step equation (9.59).

- 9.16** (*) Show that if we maximize the expected complete-data log likelihood function (9.55) for a mixture of Bernoulli distributions with respect to the mixing coefficients π_k , using a Lagrange multiplier to enforce the summation constraint, we obtain the M step equation (9.60).

- 9.17** (*) **www** Show that as a consequence of the constraint $0 \leq p(\mathbf{x}_n|\boldsymbol{\mu}_k) \leq 1$ for the discrete variable \mathbf{x}_n , the incomplete-data log likelihood function for a mixture of Bernoulli distributions is bounded above, and hence that there are no singularities for which the likelihood goes to infinity.

9.18 (**) Consider a Bernoulli mixture model as discussed in Section 9.3.3, together with a prior distribution $p(\boldsymbol{\mu}_k | a_k, b_k)$ over each of the parameter vectors $\boldsymbol{\mu}_k$ given by the beta distribution (2.13), and a Dirichlet prior $p(\boldsymbol{\pi} | \boldsymbol{\alpha})$ given by (2.38). Derive the EM algorithm for maximizing the posterior probability $p(\boldsymbol{\mu}, \boldsymbol{\pi} | \mathbf{X})$.

9.19 (**) Consider a D -dimensional variable \mathbf{x} each of whose components i is itself a multinomial variable of degree M so that \mathbf{x} is a binary vector with components x_{ij} where $i = 1, \dots, D$ and $j = 1, \dots, M$, subject to the constraint that $\sum_j x_{ij} = 1$ for all i . Suppose that the distribution of these variables is described by a mixture of the discrete multinomial distributions considered in Section 2.2 so that

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k p(\mathbf{x} | \boldsymbol{\mu}_k) \quad (9.84)$$

where

$$p(\mathbf{x} | \boldsymbol{\mu}_k) = \prod_{i=1}^D \prod_{j=1}^M \mu_{kij}^{x_{ij}}. \quad (9.85)$$

The parameters μ_{kij} represent the probabilities $p(x_{ij} = 1 | \boldsymbol{\mu}_k)$ and must satisfy $0 \leq \mu_{kij} \leq 1$ together with the constraint $\sum_j \mu_{kij} = 1$ for all values of k and i . Given an observed data set $\{\mathbf{x}_n\}$, where $n = 1, \dots, N$, derive the E and M step equations of the EM algorithm for optimizing the mixing coefficients π_k and the component parameters μ_{kij} of this distribution by maximum likelihood.

9.20 (*) **www** Show that maximization of the expected complete-data log likelihood function (9.62) for the Bayesian linear regression model leads to the M step re-estimation result (9.63) for $\boldsymbol{\alpha}$.

9.21 (**) Using the evidence framework of Section 3.5, derive the M-step re-estimation equations for the parameter β in the Bayesian linear regression model, analogous to the result (9.63) for $\boldsymbol{\alpha}$.

9.22 (**) By maximization of the expected complete-data log likelihood defined by (9.66), derive the M step equations (9.67) and (9.68) for re-estimating the hyperparameters of the relevance vector machine for regression.

9.23 (**) **www** In Section 7.2.1 we used direct maximization of the marginal likelihood to derive the re-estimation equations (7.87) and (7.88) for finding values of the hyperparameters $\boldsymbol{\alpha}$ and β for the regression RVM. Similarly, in Section 9.3.4 we used the EM algorithm to maximize the same marginal likelihood, giving the re-estimation equations (9.67) and (9.68). Show that these two sets of re-estimation equations are formally equivalent.

9.24 (*) Verify the relation (9.70) in which $\mathcal{L}(q, \theta)$ and $\text{KL}(q \| p)$ are defined by (9.71) and (9.72), respectively.

- 9.25** (*) **www** Show that the lower bound $\mathcal{L}(q, \theta)$ given by (9.71), with $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \theta^{(\text{old})})$, has the same gradient with respect to θ as the log likelihood function $\ln p(\mathbf{X}|\theta)$ at the point $\theta = \theta^{(\text{old})}$.
- 9.26** (*) **www** Consider the incremental form of the EM algorithm for a mixture of Gaussians, in which the responsibilities are recomputed only for a specific data point \mathbf{x}_m . Starting from the M-step formulae (9.17) and (9.18), derive the results (9.78) and (9.79) for updating the component means.
- 9.27** (**) Derive M-step formulae for updating the covariance matrices and mixing coefficients in a Gaussian mixture model when the responsibilities are updated incrementally, analogous to the result (9.78) for updating the means.

10

Approximate Inference



A central task in the application of probabilistic models is the evaluation of the posterior distribution $p(\mathbf{Z}|\mathbf{X})$ of the latent variables \mathbf{Z} given the observed (visible) data variables \mathbf{X} , and the evaluation of expectations computed with respect to this distribution. The model might also contain some deterministic parameters, which we will leave implicit for the moment, or it may be a fully Bayesian model in which any unknown parameters are given prior distributions and are absorbed into the set of latent variables denoted by the vector \mathbf{Z} . For instance, in the EM algorithm we need to evaluate the expectation of the complete-data log likelihood with respect to the posterior distribution of the latent variables. For many models of practical interest, it will be infeasible to evaluate the posterior distribution or indeed to compute expectations with respect to this distribution. This could be because the dimensionality of the latent space is too high to work with directly or because the posterior distribution has a highly complex form for which expectations are not analytically tractable. In the case of continuous variables, the required integrations may not have closed-form

analytical solutions, while the dimensionality of the space and the complexity of the integrand may prohibit numerical integration. For discrete variables, the marginalizations involve summing over all possible configurations of the hidden variables, and though this is always possible in principle, we often find in practice that there may be exponentially many hidden states so that exact calculation is prohibitively expensive.

In such situations, we need to resort to approximation schemes, and these fall broadly into two classes, according to whether they rely on stochastic or deterministic approximations. Stochastic techniques such as Markov chain Monte Carlo, described in Chapter 11, have enabled the widespread use of Bayesian methods across many domains. They generally have the property that given infinite computational resource, they can generate exact results, and the approximation arises from the use of a finite amount of processor time. In practice, sampling methods can be computationally demanding, often limiting their use to small-scale problems. Also, it can be difficult to know whether a sampling scheme is generating independent samples from the required distribution.

In this chapter, we introduce a range of deterministic approximation schemes, some of which scale well to large applications. These are based on analytical approximations to the posterior distribution, for example by assuming that it factorizes in a particular way or that it has a specific parametric form such as a Gaussian. As such, they can never generate exact results, and so their strengths and weaknesses are complementary to those of sampling methods.

In Section 4.4, we discussed the Laplace approximation, which is based on a local Gaussian approximation to a mode (i.e., a maximum) of the distribution. Here we turn to a family of approximation techniques called *variational inference* or *variational Bayes*, which use more global criteria and which have been widely applied. We conclude with a brief introduction to an alternative variational framework known as *expectation propagation*.

10.1. Variational Inference

Variational methods have their origins in the 18th century with the work of Euler, Lagrange, and others on the *calculus of variations*. Standard calculus is concerned with finding derivatives of functions. We can think of a function as a mapping that takes the value of a variable as the input and returns the value of the function as the output. The derivative of the function then describes how the output value varies as we make infinitesimal changes to the input value. Similarly, we can define a *functional* as a mapping that takes a function as the input and that returns the value of the functional as the output. An example would be the entropy $H[p]$, which takes a probability distribution $p(x)$ as the input and returns the quantity

$$H[p] = \int p(x) \ln p(x) dx \quad (10.1)$$

as the output. We can introduce the concept of a *functional derivative*, which expresses how the value of the functional changes in response to infinitesimal changes to the input function (Feynman *et al.*, 1964). The rules for the calculus of variations mirror those of standard calculus and are discussed in Appendix D. Many problems can be expressed in terms of an optimization problem in which the quantity being optimized is a functional. The solution is obtained by exploring all possible input functions to find the one that maximizes, or minimizes, the functional. Variational methods have broad applicability and include such areas as finite element methods (Kapur, 1989) and maximum entropy (Schwarz, 1988).

Although there is nothing intrinsically approximate about variational methods, they do naturally lend themselves to finding approximate solutions. This is done by restricting the range of functions over which the optimization is performed, for instance by considering only quadratic functions or by considering functions composed of a linear combination of fixed basis functions in which only the coefficients of the linear combination can vary. In the case of applications to probabilistic inference, the restriction may for example take the form of factorization assumptions (Jordan *et al.*, 1999; Jaakkola, 2001).

Now let us consider in more detail how the concept of variational optimization can be applied to the inference problem. Suppose we have a fully Bayesian model in which all parameters are given prior distributions. The model may also have latent variables as well as parameters, and we shall denote the set of all latent variables and parameters by \mathbf{Z} . Similarly, we denote the set of all observed variables by \mathbf{X} . For example, we might have a set of N independent, identically distributed data, for which $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ and $\mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_N\}$. Our probabilistic model specifies the joint distribution $p(\mathbf{X}, \mathbf{Z})$, and our goal is to find an approximation for the posterior distribution $p(\mathbf{Z}|\mathbf{X})$ as well as for the model evidence $p(\mathbf{X})$. As in our discussion of EM, we can decompose the log marginal probability using

$$\ln p(\mathbf{X}) = \mathcal{L}(q) + \text{KL}(q||p) \quad (10.2)$$

where we have defined

$$\mathcal{L}(q) = \int q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{X}, \mathbf{Z})}{q(\mathbf{Z})} \right\} d\mathbf{Z} \quad (10.3)$$

$$\text{KL}(q||p) = - \int q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{Z}|\mathbf{X})}{q(\mathbf{Z})} \right\} d\mathbf{Z}. \quad (10.4)$$

This differs from our discussion of EM only in that the parameter vector θ no longer appears, because the parameters are now stochastic variables and are absorbed into \mathbf{Z} . Since in this chapter we will mainly be interested in continuous variables we have used integrations rather than summations in formulating this decomposition. However, the analysis goes through unchanged if some or all of the variables are discrete simply by replacing the integrations with summations as required. As before, we can maximize the lower bound $\mathcal{L}(q)$ by optimization with respect to the distribution $q(\mathbf{Z})$, which is equivalent to minimizing the KL divergence. If we allow any possible choice for $q(\mathbf{Z})$, then the maximum of the lower bound occurs when the KL divergence vanishes, which occurs when $q(\mathbf{Z})$ equals the posterior distribution $p(\mathbf{Z}|\mathbf{X})$.

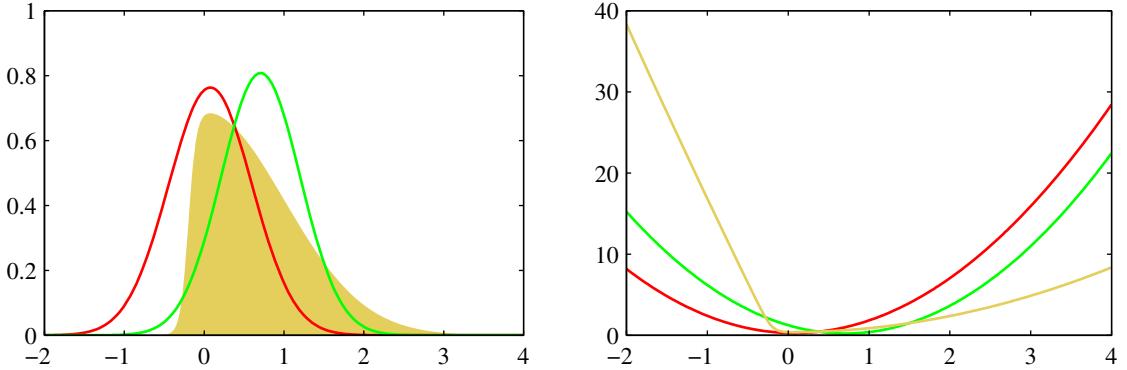


Figure 10.1 Illustration of the variational approximation for the example considered earlier in Figure 4.14. The left-hand plot shows the original distribution (yellow) along with the Laplace (red) and variational (green) approximations, and the right-hand plot shows the negative logarithms of the corresponding curves.

However, we shall suppose the model is such that working with the true posterior distribution is intractable.

We therefore consider instead a restricted family of distributions $q(\mathbf{Z})$ and then seek the member of this family for which the KL divergence is minimized. Our goal is to restrict the family sufficiently that they comprise only tractable distributions, while at the same time allowing the family to be sufficiently rich and flexible that it can provide a good approximation to the true posterior distribution. It is important to emphasize that the restriction is imposed purely to achieve tractability, and that subject to this requirement we should use as rich a family of approximating distributions as possible. In particular, there is no ‘over-fitting’ associated with highly flexible distributions. Using more flexible approximations simply allows us to approach the true posterior distribution more closely.

One way to restrict the family of approximating distributions is to use a parametric distribution $q(\mathbf{Z}|\boldsymbol{\omega})$ governed by a set of parameters $\boldsymbol{\omega}$. The lower bound $\mathcal{L}(q)$ then becomes a function of $\boldsymbol{\omega}$, and we can exploit standard nonlinear optimization techniques to determine the optimal values for the parameters. An example of this approach, in which the variational distribution is a Gaussian and we have optimized with respect to its mean and variance, is shown in Figure 10.1.

10.1.1 Factorized distributions

Here we consider an alternative way in which to restrict the family of distributions $q(\mathbf{Z})$. Suppose we partition the elements of \mathbf{Z} into disjoint groups that we denote by \mathbf{Z}_i where $i = 1, \dots, M$. We then assume that the q distribution factorizes with respect to these groups, so that

$$q(\mathbf{Z}) = \prod_{i=1}^M q_i(\mathbf{Z}_i). \quad (10.5)$$

It should be emphasized that we are making no further assumptions about the distribution. In particular, we place no restriction on the functional forms of the individual factors $q_i(\mathbf{Z}_i)$. This factorized form of variational inference corresponds to an approximation framework developed in physics called *mean field theory* (Parisi, 1988).

Amongst all distributions $q(\mathbf{Z})$ having the form (10.5), we now seek that distribution for which the lower bound $\mathcal{L}(q)$ is largest. We therefore wish to make a free form (variational) optimization of $\mathcal{L}(q)$ with respect to all of the distributions $q_i(\mathbf{Z}_i)$, which we do by optimizing with respect to each of the factors in turn. To achieve this, we first substitute (10.5) into (10.3) and then dissect out the dependence on one of the factors $q_j(\mathbf{Z}_j)$. Denoting $q_j(\mathbf{Z}_j)$ by simply q_j to keep the notation uncluttered, we then obtain

$$\begin{aligned}\mathcal{L}(q) &= \int \prod_i q_i \left\{ \ln p(\mathbf{X}, \mathbf{Z}) - \sum_i \ln q_i \right\} d\mathbf{Z} \\ &= \int q_j \left\{ \int \ln p(\mathbf{X}, \mathbf{Z}) \prod_{i \neq j} q_i d\mathbf{Z}_i \right\} d\mathbf{Z}_j - \int q_j \ln q_j d\mathbf{Z}_j + \text{const} \\ &= \int q_j \ln \tilde{p}(\mathbf{X}, \mathbf{Z}_j) d\mathbf{Z}_j - \int q_j \ln q_j d\mathbf{Z}_j + \text{const}\end{aligned}\quad (10.6)$$

where we have defined a new distribution $\tilde{p}(\mathbf{X}, \mathbf{Z}_j)$ by the relation

$$\ln \tilde{p}(\mathbf{X}, \mathbf{Z}_j) = \mathbb{E}_{i \neq j} [\ln p(\mathbf{X}, \mathbf{Z})] + \text{const}. \quad (10.7)$$

Here the notation $\mathbb{E}_{i \neq j} [\dots]$ denotes an expectation with respect to the q distributions over all variables \mathbf{z}_i for $i \neq j$, so that

$$\mathbb{E}_{i \neq j} [\ln p(\mathbf{X}, \mathbf{Z})] = \int \ln p(\mathbf{X}, \mathbf{Z}) \prod_{i \neq j} q_i d\mathbf{Z}_i. \quad (10.8)$$

Now suppose we keep the $\{q_{i \neq j}\}$ fixed and maximize $\mathcal{L}(q)$ in (10.6) with respect to all possible forms for the distribution $q_j(\mathbf{Z}_j)$. This is easily done by recognizing that (10.6) is a negative Kullback-Leibler divergence between $q_j(\mathbf{Z}_j)$ and $\tilde{p}(\mathbf{X}, \mathbf{Z}_j)$. Thus maximizing (10.6) is equivalent to minimizing the Kullback-Leibler



Leonhard Euler
1707–1783

Euler was a Swiss mathematician and physicist who worked in St. Petersburg and Berlin and who is widely considered to be one of the greatest mathematicians of all time. He is certainly the most prolific, and his collected works fill 75 volumes. Amongst his many

contributions, he formulated the modern theory of the function, he developed (together with Lagrange) the calculus of variations, and he discovered the formula $e^{i\pi} = -1$, which relates four of the most important numbers in mathematics. During the last 17 years of his life, he was almost totally blind, and yet he produced nearly half of his results during this period.

divergence, and the minimum occurs when $q_j(\mathbf{Z}_j) = \tilde{p}(\mathbf{X}, \mathbf{Z}_j)$. Thus we obtain a general expression for the optimal solution $q_j^*(\mathbf{Z}_j)$ given by

$$\ln q_j^*(\mathbf{Z}_j) = \mathbb{E}_{i \neq j} [\ln p(\mathbf{X}, \mathbf{Z})] + \text{const.} \quad (10.9)$$

It is worth taking a few moments to study the form of this solution as it provides the basis for applications of variational methods. It says that the log of the optimal solution for factor q_j is obtained simply by considering the log of the joint distribution over all hidden and visible variables and then taking the expectation with respect to all of the other factors $\{q_i\}$ for $i \neq j$.

The additive constant in (10.9) is set by normalizing the distribution $q_j^*(\mathbf{Z}_j)$. Thus if we take the exponential of both sides and normalize, we have

$$q_j^*(\mathbf{Z}_j) = \frac{\exp(\mathbb{E}_{i \neq j} [\ln p(\mathbf{X}, \mathbf{Z})])}{\int \exp(\mathbb{E}_{i \neq j} [\ln p(\mathbf{X}, \mathbf{Z})]) d\mathbf{Z}_j}.$$

In practice, we shall find it more convenient to work with the form (10.9) and then reinstate the normalization constant (where required) by inspection. This will become clear from subsequent examples.

The set of equations given by (10.9) for $j = 1, \dots, M$ represent a set of consistency conditions for the maximum of the lower bound subject to the factorization constraint. However, they do not represent an explicit solution because the expression on the right-hand side of (10.9) for the optimum $q_j^*(\mathbf{Z}_j)$ depends on expectations computed with respect to the other factors $q_i(\mathbf{Z}_i)$ for $i \neq j$. We will therefore seek a consistent solution by first initializing all of the factors $q_i(\mathbf{Z}_i)$ appropriately and then cycling through the factors and replacing each in turn with a revised estimate given by the right-hand side of (10.9) evaluated using the current estimates for all of the other factors. Convergence is guaranteed because bound is convex with respect to each of the factors $q_i(\mathbf{Z}_i)$ (Boyd and Vandenberghe, 2004).

10.1.2 Properties of factorized approximations

Our approach to variational inference is based on a factorized approximation to the true posterior distribution. Let us consider for a moment the problem of approximating a general distribution by a factorized distribution. To begin with, we discuss the problem of approximating a Gaussian distribution using a factorized Gaussian, which will provide useful insight into the types of inaccuracy introduced in using factorized approximations. Consider a Gaussian distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$ over two correlated variables $\mathbf{z} = (z_1, z_2)$ in which the mean and precision have elements

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \quad \boldsymbol{\Lambda} = \begin{pmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{pmatrix} \quad (10.10)$$

and $\Lambda_{21} = \Lambda_{12}$ due to the symmetry of the precision matrix. Now suppose we wish to approximate this distribution using a factorized Gaussian of the form $q(\mathbf{z}) = q_1(z_1)q_2(z_2)$. We first apply the general result (10.9) to find an expression for the

optimal factor $q_1^*(z_1)$. In doing so it is useful to note that on the right-hand side we only need to retain those terms that have some functional dependence on z_1 because all other terms can be absorbed into the normalization constant. Thus we have

$$\begin{aligned}\ln q_1^*(z_1) &= \mathbb{E}_{z_2}[\ln p(\mathbf{z})] + \text{const} \\ &= \mathbb{E}_{z_2} \left[-\frac{1}{2}(z_1 - \mu_1)^2 \Lambda_{11} - (z_1 - \mu_1) \Lambda_{12} (z_2 - \mu_2) \right] + \text{const} \\ &= -\frac{1}{2} z_1^2 \Lambda_{11} + z_1 \mu_1 \Lambda_{11} - z_1 \Lambda_{12} (\mathbb{E}[z_2] - \mu_2) + \text{const}. \quad (10.11)\end{aligned}$$

Next we observe that the right-hand side of this expression is a quadratic function of z_1 , and so we can identify $q^*(z_1)$ as a Gaussian distribution. It is worth emphasizing that we did not assume that $q(z_i)$ is Gaussian, but rather we derived this result by variational optimization of the KL divergence over all possible distributions $q(z_i)$. Note also that we do not need to consider the additive constant in (10.9) explicitly because it represents the normalization constant that can be found at the end by inspection if required. Using the technique of completing the square, we can identify the mean and precision of this Gaussian, giving

$$q^*(z_1) = \mathcal{N}(z_1 | m_1, \Lambda_{11}^{-1}) \quad (10.12)$$

where

$$m_1 = \mu_1 - \Lambda_{11}^{-1} \Lambda_{12} (\mathbb{E}[z_2] - \mu_2). \quad (10.13)$$

By symmetry, $q_2^*(z_2)$ is also Gaussian and can be written as

$$q_2^*(z_2) = \mathcal{N}(z_2 | m_2, \Lambda_{22}^{-1}) \quad (10.14)$$

in which

$$m_2 = \mu_2 - \Lambda_{22}^{-1} \Lambda_{21} (\mathbb{E}[z_1] - \mu_1). \quad (10.15)$$

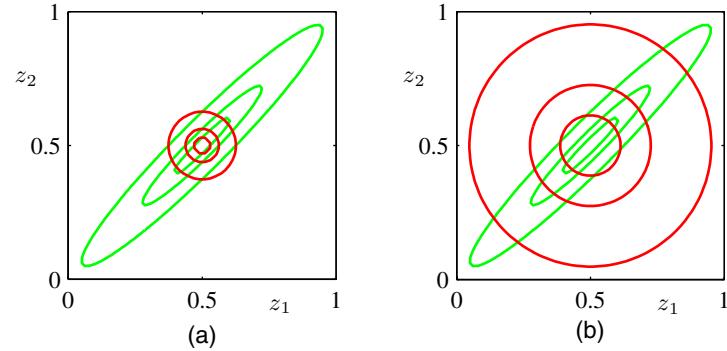
Note that these solutions are coupled, so that $q^*(z_1)$ depends on expectations computed with respect to $q^*(z_2)$ and vice versa. In general, we address this by treating the variational solutions as re-estimation equations and cycling through the variables in turn updating them until some convergence criterion is satisfied. We shall see an example of this shortly. Here, however, we note that the problem is sufficiently simple that a closed form solution can be found. In particular, because $\mathbb{E}[z_1] = m_1$ and $\mathbb{E}[z_2] = m_2$, we see that the two equations are satisfied if we take $\mathbb{E}[z_1] = \mu_1$ and $\mathbb{E}[z_2] = \mu_2$, and it is easily shown that this is the only solution provided the distribution is nonsingular. This result is illustrated in Figure 10.2(a). We see that the mean is correctly captured but that the variance of $q(\mathbf{z})$ is controlled by the direction of smallest variance of $p(\mathbf{z})$, and that the variance along the orthogonal direction is significantly under-estimated. It is a general result that a factorized variational approximation tends to give approximations to the posterior distribution that are too compact.

By way of comparison, suppose instead that we had been minimizing the reverse Kullback-Leibler divergence $\text{KL}(p||q)$. As we shall see, this form of KL divergence

Section 2.3.1

Exercise 10.2

Figure 10.2 Comparison of the two alternative forms for the Kullback-Leibler divergence. The green contours corresponding to 1, 2, and 3 standard deviations for a correlated Gaussian distribution $p(\mathbf{z})$ over two variables z_1 and z_2 , and the red contours represent the corresponding levels for an approximating distribution $q(\mathbf{z})$ over the same variables given by the product of two independent univariate Gaussian distributions whose parameters are obtained by minimization of (a) the Kullback-Leibler divergence $\text{KL}(q||p)$, and (b) the reverse Kullback-Leibler divergence $\text{KL}(p||q)$.



Section 10.7

is used in an alternative approximate inference framework called *expectation propagation*. We therefore consider the general problem of minimizing $\text{KL}(p||q)$ when $q(\mathbf{Z})$ is a factorized approximation of the form (10.5). The KL divergence can then be written in the form

$$\text{KL}(p||q) = - \int p(\mathbf{Z}) \left[\sum_{i=1}^M \ln q_i(\mathbf{Z}_i) \right] d\mathbf{Z} + \text{const} \quad (10.16)$$

Exercise 10.3

where the constant term is simply the entropy of $p(\mathbf{Z})$ and so does not depend on $q(\mathbf{Z})$. We can now optimize with respect to each of the factors $q_j(\mathbf{Z}_j)$, which is easily done using a Lagrange multiplier to give

$$q_j^*(\mathbf{Z}_j) = \int p(\mathbf{Z}) \prod_{i \neq j} d\mathbf{Z}_i = p(\mathbf{Z}_j). \quad (10.17)$$

In this case, we find that the optimal solution for $q_j(\mathbf{Z}_j)$ is just given by the corresponding marginal distribution of $p(\mathbf{Z})$. Note that this is a closed-form solution and so does not require iteration.

To apply this result to the illustrative example of a Gaussian distribution $p(\mathbf{z})$ over a vector \mathbf{z} we can use (2.98), which gives the result shown in Figure 10.2(b). We see that once again the mean of the approximation is correct, but that it places significant probability mass in regions of variable space that have very low probability.

The difference between these two results can be understood by noting that there is a large positive contribution to the Kullback-Leibler divergence

$$\text{KL}(q||p) = - \int q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{Z})}{q(\mathbf{Z})} \right\} d\mathbf{Z} \quad (10.18)$$

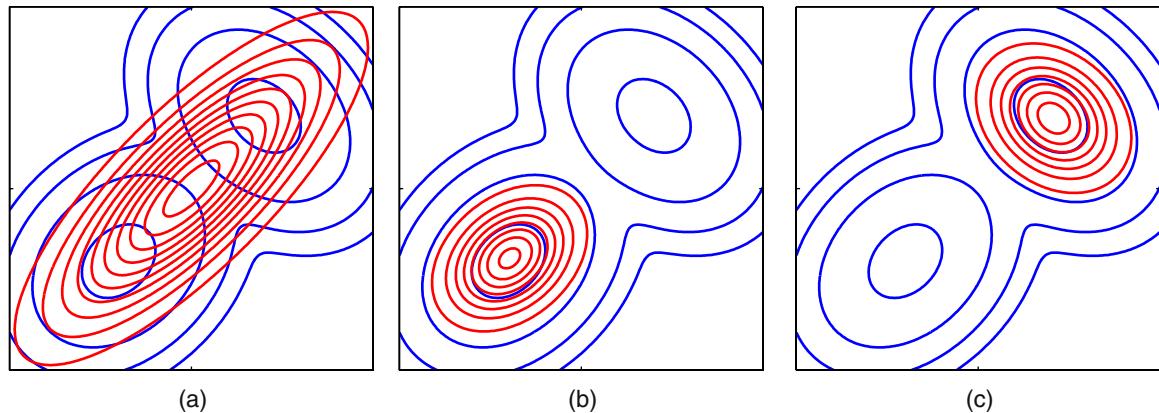


Figure 10.3 Another comparison of the two alternative forms for the Kullback-Leibler divergence. (a) The blue contours show a bimodal distribution $p(\mathbf{Z})$ given by a mixture of two Gaussians, and the red contours correspond to the single Gaussian distribution $q(\mathbf{Z})$ that best approximates $p(\mathbf{Z})$ in the sense of minimizing the Kullback-Leibler divergence $\text{KL}(p\|q)$. (b) As in (a) but now the red contours correspond to a Gaussian distribution $q(\mathbf{Z})$ found by numerical minimization of the Kullback-Leibler divergence $\text{KL}(q\|p)$. (c) As in (b) but showing a different local minimum of the Kullback-Leibler divergence.

from regions of \mathbf{Z} space in which $p(\mathbf{Z})$ is near zero unless $q(\mathbf{Z})$ is also close to zero. Thus minimizing this form of KL divergence leads to distributions $q(\mathbf{Z})$ that avoid regions in which $p(\mathbf{Z})$ is small. Conversely, the Kullback-Leibler divergence $\text{KL}(p\|q)$ is minimized by distributions $q(\mathbf{Z})$ that are nonzero in regions where $p(\mathbf{Z})$ is nonzero.

We can gain further insight into the different behaviour of the two KL divergences if we consider approximating a multimodal distribution by a unimodal one, as illustrated in Figure 10.3. In practical applications, the true posterior distribution will often be multimodal, with most of the posterior mass concentrated in some number of relatively small regions of parameter space. These multiple modes may arise through nonidentifiability in the latent space or through complex nonlinear dependence on the parameters. Both types of multimodality were encountered in Chapter 9 in the context of Gaussian mixtures, where they manifested themselves as multiple maxima in the likelihood function, and a variational treatment based on the minimization of $\text{KL}(q\|p)$ will tend to find one of these modes. By contrast, if we were to minimize $\text{KL}(p\|q)$, the resulting approximations would average across all of the modes and, in the context of the mixture model, would lead to poor predictive distributions (because the average of two good parameter values is typically itself not a good parameter value). It is possible to make use of $\text{KL}(p\|q)$ to define a useful inference procedure, but this requires a rather different approach to the one discussed here, and will be considered in detail when we discuss expectation propagation.

The two forms of Kullback-Leibler divergence are members of the *alpha family*

of divergences (Ali and Silvey, 1966; Amari, 1985; Minka, 2005) defined by

$$D_\alpha(p\|q) = \frac{4}{1-\alpha^2} \left(1 - \int p(x)^{(1+\alpha)/2} q(x)^{(1-\alpha)/2} dx \right) \quad (10.19)$$

where $-\infty < \alpha < \infty$ is a continuous parameter. The Kullback-Leibler divergence $\text{KL}(p\|q)$ corresponds to the limit $\alpha \rightarrow 1$, whereas $\text{KL}(q\|p)$ corresponds to the limit $\alpha \rightarrow -1$. For all values of α we have $D_\alpha(p\|q) \geq 0$, with equality if, and only if, $p(x) = q(x)$. Suppose $p(x)$ is a fixed distribution, and we minimize $D_\alpha(p\|q)$ with respect to some set of distributions $q(x)$. Then for $\alpha \leq -1$ the divergence is *zero forcing*, so that any values of x for which $p(x) = 0$ will have $q(x) = 0$, and typically $q(x)$ will under-estimate the support of $p(x)$ and will tend to seek the mode with the largest mass. Conversely for $\alpha \geq 1$ the divergence is *zero-avoiding*, so that values of x for which $p(x) > 0$ will have $q(x) > 0$, and typically $q(x)$ will stretch to cover all of $p(x)$, and will over-estimate the support of $p(x)$. When $\alpha = 0$ we obtain a symmetric divergence that is linearly related to the *Hellinger distance* given by

$$D_H(p\|q) = \int (p(x)^{1/2} - q(x)^{1/2}) dx. \quad (10.20)$$

The square root of the Hellinger distance is a valid distance metric.

10.1.3 Example: The univariate Gaussian

We now illustrate the factorized variational approximation using a Gaussian distribution over a single variable x (MacKay, 2003). Our goal is to infer the posterior distribution for the mean μ and precision τ , given a data set $\mathcal{D} = \{x_1, \dots, x_N\}$ of observed values of x which are assumed to be drawn independently from the Gaussian. The likelihood function is given by

$$p(\mathcal{D}|\mu, \tau) = \left(\frac{\tau}{2\pi} \right)^{N/2} \exp \left\{ -\frac{\tau}{2} \sum_{n=1}^N (x_n - \mu)^2 \right\}. \quad (10.21)$$

We now introduce conjugate prior distributions for μ and τ given by

$$p(\mu|\tau) = \mathcal{N}(\mu|\mu_0, (\lambda_0\tau)^{-1}) \quad (10.22)$$

$$p(\tau) = \text{Gam}(\tau|a_0, b_0) \quad (10.23)$$

where $\text{Gam}(\tau|a_0, b_0)$ is the gamma distribution defined by (2.146). Together these distributions constitute a Gaussian-Gamma conjugate prior distribution.

For this simple problem the posterior distribution can be found exactly, and again takes the form of a Gaussian-gamma distribution. However, for tutorial purposes we will consider a factorized variational approximation to the posterior distribution given by

$$q(\mu, \tau) = q_\mu(\mu)q_\tau(\tau). \quad (10.24)$$

Note that the true posterior distribution does not factorize in this way. The optimum factors $q_\mu(\mu)$ and $q_\tau(\tau)$ can be obtained from the general result (10.9) as follows. For $q_\mu(\mu)$ we have

$$\begin{aligned}\ln q_\mu^*(\mu) &= \mathbb{E}_\tau [\ln p(\mathcal{D}|\mu, \tau) + \ln p(\mu|\tau)] + \text{const} \\ &= -\frac{\mathbb{E}[\tau]}{2} \left\{ \lambda_0(\mu - \mu_0)^2 + \sum_{n=1}^N (x_n - \mu)^2 \right\} + \text{const.}\end{aligned}\quad (10.25)$$

Completing the square over μ we see that $q_\mu(\mu)$ is a Gaussian $\mathcal{N}(\mu|\mu_N, \lambda_N^{-1})$ with mean and precision given by

$$\mu_N = \frac{\lambda_0 \mu_0 + N \bar{x}}{\lambda_0 + N} \quad (10.26)$$

$$\lambda_N = (\lambda_0 + N) \mathbb{E}[\tau]. \quad (10.27)$$

Note that for $N \rightarrow \infty$ this gives the maximum likelihood result in which $\mu_N = \bar{x}$ and the precision is infinite.

Similarly, the optimal solution for the factor $q_\tau(\tau)$ is given by

$$\begin{aligned}\ln q_\tau^*(\tau) &= \mathbb{E}_\mu [\ln p(\mathcal{D}|\mu, \tau) + \ln p(\mu|\tau)] + \ln p(\tau) + \text{const} \\ &= (a_0 - 1) \ln \tau - b_0 \tau + \frac{N}{2} \ln \tau \\ &\quad - \frac{\tau}{2} \mathbb{E}_\mu \left[\sum_{n=1}^N (x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2 \right] + \text{const}\end{aligned}\quad (10.28)$$

and hence $q_\tau(\tau)$ is a gamma distribution $\text{Gam}(\tau|a_N, b_N)$ with parameters

$$a_N = a_0 + \frac{N}{2} \quad (10.29)$$

$$b_N = b_0 + \frac{1}{2} \mathbb{E}_\mu \left[\sum_{n=1}^N (x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2 \right]. \quad (10.30)$$

Again this exhibits the expected behaviour when $N \rightarrow \infty$.

It should be emphasized that we did not assume these specific functional forms for the optimal distributions $q_\mu(\mu)$ and $q_\tau(\tau)$. They arose naturally from the structure of the likelihood function and the corresponding conjugate priors.

Thus we have expressions for the optimal distributions $q_\mu(\mu)$ and $q_\tau(\tau)$ each of which depends on moments evaluated with respect to the other distribution. One approach to finding a solution is therefore to make an initial guess for, say, the moment $\mathbb{E}[\tau]$ and use this to re-compute the distribution $q_\mu(\mu)$. Given this revised distribution we can then extract the required moments $\mathbb{E}[\mu]$ and $\mathbb{E}[\mu^2]$, and use these to recompute the distribution $q_\tau(\tau)$, and so on. Since the space of hidden variables for this example is only two dimensional, we can illustrate the variational approximation to the posterior distribution by plotting contours of both the true posterior and the factorized approximation, as illustrated in Figure 10.4.

Exercise 10.7

Section 10.4.1

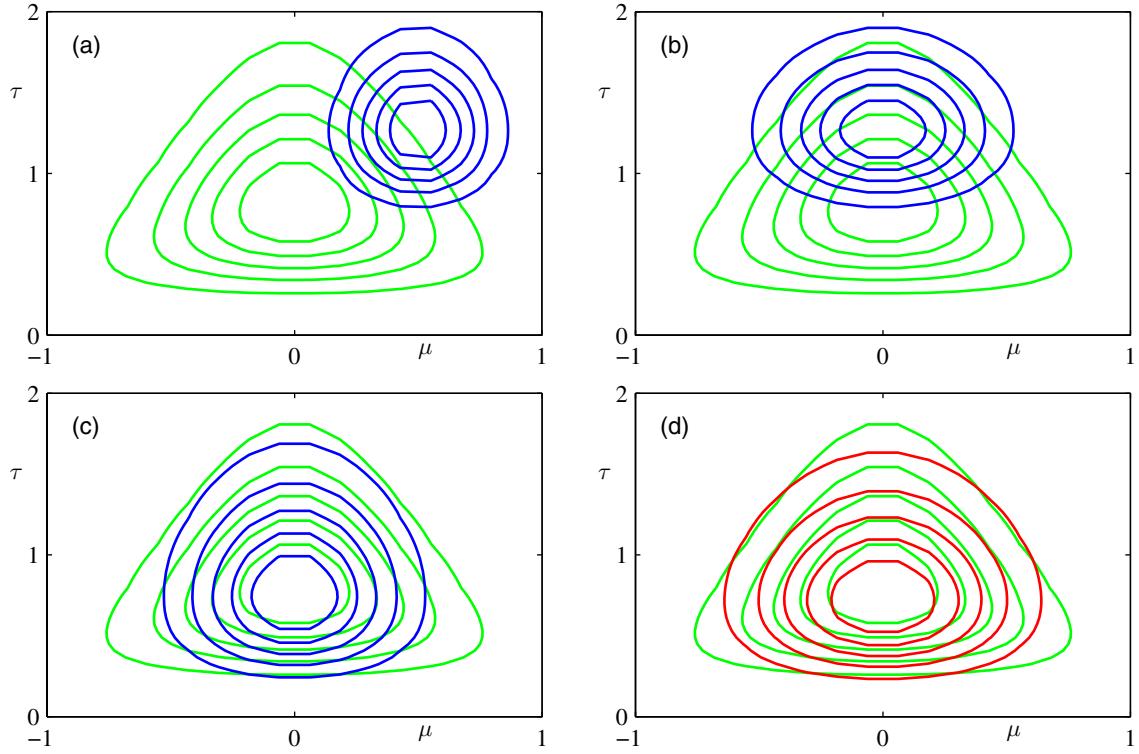


Figure 10.4 Illustration of variational inference for the mean μ and precision τ of a univariate Gaussian distribution. Contours of the true posterior distribution $p(\mu, \tau | D)$ are shown in green. (a) Contours of the initial factorized approximation $q_\mu(\mu)q_\tau(\tau)$ are shown in blue. (b) After re-estimating the factor $q_\mu(\mu)$. (c) After re-estimating the factor $q_\tau(\tau)$. (d) Contours of the optimal factorized approximation, to which the iterative scheme converges, are shown in red.

In general, we will need to use an iterative approach such as this in order to solve for the optimal factorized posterior distribution. For the very simple example we are considering here, however, we can find an explicit solution by solving the simultaneous equations for the optimal factors $q_\mu(\mu)$ and $q_\tau(\tau)$. Before doing this, we can simplify these expressions by considering broad, noninformative priors in which $\mu_0 = a_0 = b_0 = \lambda_0 = 0$. Although these parameter settings correspond to improper priors, we see that the posterior distribution is still well defined. Using the standard result $\mathbb{E}[\tau] = a_N/b_N$ for the mean of a gamma distribution, together with (10.29) and (10.30), we have

$$\frac{1}{\mathbb{E}[\tau]} = \mathbb{E}\left[\frac{1}{N} \sum_{n=1}^N (x_n - \mu)^2\right] = \bar{x}^2 - 2\bar{x}\mathbb{E}[\mu] + \mathbb{E}[\mu^2]. \quad (10.31)$$

$q_\mu(\mu)$ in the form

$$\mathbb{E}[\mu] = \bar{x}, \quad \mathbb{E}[\mu^2] = \bar{x}^2 + \frac{1}{N\mathbb{E}[\tau]}. \quad (10.32)$$

Exercise 10.9

We can now substitute these moments into (10.31) and then solve for $\mathbb{E}[\tau]$ to give

$$\begin{aligned} \frac{1}{\mathbb{E}[\tau]} &= \frac{1}{N-1}(\bar{x}^2 - \bar{x}^2) \\ &= \frac{1}{N-1} \sum_{n=1}^N (x_n - \bar{x})^2. \end{aligned} \quad (10.33)$$

Section 1.2.4

We recognize the right-hand side as the familiar unbiased estimator for the variance of a univariate Gaussian distribution, and so we see that the use of a Bayesian approach has avoided the bias of the maximum likelihood solution.

10.1.4 Model comparison

As well as performing inference over the hidden variables \mathbf{Z} , we may also wish to compare a set of candidate models, labelled by the index m , and having prior probabilities $p(m)$. Our goal is then to approximate the posterior probabilities $p(m|\mathbf{X})$, where \mathbf{X} is the observed data. This is a slightly more complex situation than that considered so far because different models may have different structure and indeed different dimensionality for the hidden variables \mathbf{Z} . We cannot therefore simply consider a factorized approximation $q(\mathbf{Z})q(m)$, but must instead recognize that the posterior over \mathbf{Z} must be conditioned on m , and so we must consider $q(\mathbf{Z}, m) = q(\mathbf{Z}|m)q(m)$. We can readily verify the following decomposition based on this variational distribution

$$\ln p(\mathbf{X}) = \mathcal{L}_m - \sum_m \sum_{\mathbf{Z}} q(\mathbf{Z}|m)q(m) \ln \left\{ \frac{p(\mathbf{Z}, m|\mathbf{X})}{q(\mathbf{Z}|m)q(m)} \right\} \quad (10.34)$$

Exercise 10.10

where the \mathcal{L}_m is a lower bound on $\ln p(\mathbf{X})$ and is given by

$$\mathcal{L}_m = \sum_m \sum_{\mathbf{Z}} q(\mathbf{Z}|m)q(m) \ln \left\{ \frac{p(\mathbf{Z}, \mathbf{X}, m)}{q(\mathbf{Z}|m)q(m)} \right\}. \quad (10.35)$$

Exercise 10.11

Here we are assuming discrete \mathbf{Z} , but the same analysis applies to continuous latent variables provided the summations are replaced with integrations. We can maximize \mathcal{L}_m with respect to the distribution $q(m)$ using a Lagrange multiplier, with the result

$$q(m) \propto p(m) \exp\{\mathcal{L}_m\}. \quad (10.36)$$

However, if we maximize \mathcal{L}_m with respect to the $q(\mathbf{Z}|m)$, we find that the solutions for different m are coupled, as we expect because they are conditioned on m . We proceed instead by first optimizing each of the $q(\mathbf{Z}|m)$ individually by optimization

of (10.35), and then subsequently determining the $q(m)$ using (10.36). After normalization the resulting values for $q(m)$ can be used for model selection or model averaging in the usual way.

10.2. Illustration: Variational Mixture of Gaussians

We now return to our discussion of the Gaussian mixture model and apply the variational inference machinery developed in the previous section. This will provide a good illustration of the application of variational methods and will also demonstrate how a Bayesian treatment elegantly resolves many of the difficulties associated with the maximum likelihood approach (Attias, 1999b). The reader is encouraged to work through this example in detail as it provides many insights into the practical application of variational methods. Many Bayesian models, corresponding to much more sophisticated distributions, can be solved by straightforward extensions and generalizations of this analysis.

Our starting point is the likelihood function for the Gaussian mixture model, illustrated by the graphical model in Figure 9.6. For each observation \mathbf{x}_n we have a corresponding latent variable \mathbf{z}_n comprising a 1-of- K binary vector with elements z_{nk} for $k = 1, \dots, K$. As before we denote the observed data set by $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, and similarly we denote the latent variables by $\mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_N\}$. From (9.10) we can write down the conditional distribution of \mathbf{Z} , given the mixing coefficients $\boldsymbol{\pi}$, in the form

$$p(\mathbf{Z}|\boldsymbol{\pi}) = \prod_{n=1}^N \prod_{k=1}^K \pi_k^{z_{nk}}. \quad (10.37)$$

Similarly, from (9.11), we can write down the conditional distribution of the observed data vectors, given the latent variables and the component parameters

$$p(\mathbf{X}|\mathbf{Z}, \boldsymbol{\mu}, \boldsymbol{\Lambda}) = \prod_{n=1}^N \prod_{k=1}^K \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})^{z_{nk}} \quad (10.38)$$

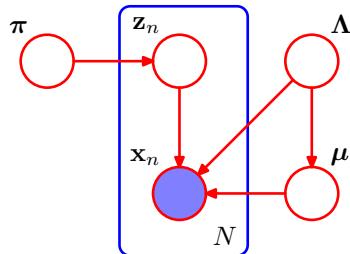
where $\boldsymbol{\mu} = \{\boldsymbol{\mu}_k\}$ and $\boldsymbol{\Lambda} = \{\boldsymbol{\Lambda}_k\}$. Note that we are working in terms of precision matrices rather than covariance matrices as this somewhat simplifies the mathematics.

Next we introduce priors over the parameters $\boldsymbol{\mu}$, $\boldsymbol{\Lambda}$ and $\boldsymbol{\pi}$. The analysis is considerably simplified if we use conjugate prior distributions. We therefore choose a Dirichlet distribution over the mixing coefficients $\boldsymbol{\pi}$

$$p(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}_0) = C(\boldsymbol{\alpha}_0) \prod_{k=1}^K \pi_k^{\alpha_0 - 1} \quad (10.39)$$

where by symmetry we have chosen the same parameter α_0 for each of the components, and $C(\boldsymbol{\alpha}_0)$ is the normalization constant for the Dirichlet distribution defined

Figure 10.5 Directed acyclic graph representing the Bayesian mixture of Gaussians model, in which the box (plate) denotes a set of N i.i.d. observations. Here μ denotes $\{\mu_k\}$ and Λ denotes $\{\Lambda_k\}$.



Section 2.2.1

by (B.23). As we have seen, the parameter α_0 can be interpreted as the effective prior number of observations associated with each component of the mixture. If the value of α_0 is small, then the posterior distribution will be influenced primarily by the data rather than by the prior.

Similarly, we introduce an independent Gaussian-Wishart prior governing the mean and precision of each Gaussian component, given by

$$\begin{aligned} p(\boldsymbol{\mu}, \boldsymbol{\Lambda}) &= p(\boldsymbol{\mu}|\boldsymbol{\Lambda})p(\boldsymbol{\Lambda}) \\ &= \prod_{k=1}^K \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{m}_0, (\beta_0 \boldsymbol{\Lambda}_k)^{-1}) \mathcal{W}(\boldsymbol{\Lambda}_k | \mathbf{W}_0, \nu_0) \end{aligned} \quad (10.40)$$

Section 2.3.6

because this represents the conjugate prior distribution when both the mean and precision are unknown. Typically we would choose $\mathbf{m}_0 = \mathbf{0}$ by symmetry.

The resulting model can be represented as a directed graph as shown in Figure 10.5. Note that there is a link from $\boldsymbol{\Lambda}$ to $\boldsymbol{\mu}$ since the variance of the distribution over $\boldsymbol{\mu}$ in (10.40) is a function of $\boldsymbol{\Lambda}$.

This example provides a nice illustration of the distinction between latent variables and parameters. Variables such as \mathbf{z}_n that appear inside the plate are regarded as latent variables because the number of such variables grows with the size of the data set. By contrast, variables such as $\boldsymbol{\mu}$ that are outside the plate are fixed in number independently of the size of the data set, and so are regarded as parameters. From the perspective of graphical models, however, there is really no fundamental difference between them.

10.2.1 Variational distribution

In order to formulate a variational treatment of this model, we next write down the joint distribution of all of the random variables, which is given by

$$p(\mathbf{X}, \mathbf{Z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}) = p(\mathbf{X}|\mathbf{Z}, \boldsymbol{\mu}, \boldsymbol{\Lambda})p(\mathbf{Z}|\boldsymbol{\pi})p(\boldsymbol{\pi})p(\boldsymbol{\mu}|\boldsymbol{\Lambda})p(\boldsymbol{\Lambda}) \quad (10.41)$$

in which the various factors are defined above. The reader should take a moment to verify that this decomposition does indeed correspond to the probabilistic graphical model shown in Figure 10.5. Note that only the variables $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ are observed.

We now consider a variational distribution which factorizes between the latent variables and the parameters so that

$$q(\mathbf{Z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}) = q(\mathbf{Z})q(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}). \quad (10.42)$$

It is remarkable that this is the *only* assumption that we need to make in order to obtain a tractable practical solution to our Bayesian mixture model. In particular, the functional form of the factors $q(\mathbf{Z})$ and $q(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda})$ will be determined automatically by optimization of the variational distribution. Note that we are omitting the subscripts on the q distributions, much as we do with the p distributions in (10.41), and are relying on the arguments to distinguish the different distributions.

The corresponding sequential update equations for these factors can be easily derived by making use of the general result (10.9). Let us consider the derivation of the update equation for the factor $q(\mathbf{Z})$. The log of the optimized factor is given by

$$\ln q^*(\mathbf{Z}) = \mathbb{E}_{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}} [\ln p(\mathbf{X}, \mathbf{Z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda})] + \text{const.} \quad (10.43)$$

We now make use of the decomposition (10.41). Note that we are only interested in the functional dependence of the right-hand side on the variable \mathbf{Z} . Thus any terms that do not depend on \mathbf{Z} can be absorbed into the additive normalization constant, giving

$$\ln q^*(\mathbf{Z}) = \mathbb{E}_{\boldsymbol{\pi}} [\ln p(\mathbf{Z} | \boldsymbol{\pi})] + \mathbb{E}_{\boldsymbol{\mu}, \boldsymbol{\Lambda}} [\ln p(\mathbf{X} | \mathbf{Z}, \boldsymbol{\mu}, \boldsymbol{\Lambda})] + \text{const.} \quad (10.44)$$

Substituting for the two conditional distributions on the right-hand side, and again absorbing any terms that are independent of \mathbf{Z} into the additive constant, we have

$$\ln q^*(\mathbf{Z}) = \sum_{n=1}^N \sum_{k=1}^K z_{nk} \ln \rho_{nk} + \text{const} \quad (10.45)$$

where we have defined

$$\begin{aligned} \ln \rho_{nk} &= \mathbb{E}[\ln \pi_k] + \frac{1}{2} \mathbb{E}[\ln |\boldsymbol{\Lambda}_k|] - \frac{D}{2} \ln(2\pi) \\ &\quad - \frac{1}{2} \mathbb{E}_{\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k} [(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Lambda}_k (\mathbf{x}_n - \boldsymbol{\mu}_k)] \end{aligned} \quad (10.46)$$

where D is the dimensionality of the data variable \mathbf{x} . Taking the exponential of both sides of (10.45) we obtain

$$q^*(\mathbf{Z}) \propto \prod_{n=1}^N \prod_{k=1}^K \rho_{nk}^{z_{nk}}. \quad (10.47)$$

Requiring that this distribution be normalized, and noting that for each value of n the quantities z_{nk} are binary and sum to 1 over all values of k , we obtain

$$q^*(\mathbf{Z}) = \prod_{n=1}^N \prod_{k=1}^K r_{nk}^{z_{nk}} \quad (10.48)$$

where

$$r_{nk} = \frac{\rho_{nk}}{\sum_{j=1}^K \rho_{nj}}. \quad (10.49)$$

We see that the optimal solution for the factor $q(\mathbf{Z})$ takes the same functional form as the prior $p(\mathbf{Z}|\boldsymbol{\pi})$. Note that because ρ_{nk} is given by the exponential of a real quantity, the quantities r_{nk} will be nonnegative and will sum to one, as required.

For the discrete distribution $q^*(\mathbf{Z})$ we have the standard result

$$\mathbb{E}[z_{nk}] = r_{nk} \quad (10.50)$$

from which we see that the quantities r_{nk} are playing the role of responsibilities. Note that the optimal solution for $q^*(\mathbf{Z})$ depends on moments evaluated with respect to the distributions of other variables, and so again the variational update equations are coupled and must be solved iteratively.

At this point, we shall find it convenient to define three statistics of the observed data set evaluated with respect to the responsibilities, given by

$$N_k = \sum_{n=1}^N r_{nk} \quad (10.51)$$

$$\bar{\mathbf{x}}_k = \frac{1}{N_k} \sum_{n=1}^N r_{nk} \mathbf{x}_n \quad (10.52)$$

$$\mathbf{S}_k = \frac{1}{N_k} \sum_{n=1}^N r_{nk} (\mathbf{x}_n - \bar{\mathbf{x}}_k)(\mathbf{x}_n - \bar{\mathbf{x}}_k)^T. \quad (10.53)$$

Note that these are analogous to quantities evaluated in the maximum likelihood EM algorithm for the Gaussian mixture model.

Now let us consider the factor $q(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda})$ in the variational posterior distribution. Again using the general result (10.9) we have

$$\begin{aligned} \ln q^*(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}) &= \ln p(\boldsymbol{\pi}) + \sum_{k=1}^K \ln p(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) + \mathbb{E}_{\mathbf{Z}} [\ln p(\mathbf{Z}|\boldsymbol{\pi})] \\ &\quad + \sum_{k=1}^K \sum_{n=1}^N \mathbb{E}[z_{nk}] \ln \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1}) + \text{const.} \end{aligned} \quad (10.54)$$

We observe that the right-hand side of this expression decomposes into a sum of terms involving only $\boldsymbol{\pi}$ together with terms only involving $\boldsymbol{\mu}$ and $\boldsymbol{\Lambda}$, which implies that the variational posterior $q(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda})$ factorizes to give $q(\boldsymbol{\pi})q(\boldsymbol{\mu}, \boldsymbol{\Lambda})$. Furthermore, the terms involving $\boldsymbol{\mu}$ and $\boldsymbol{\Lambda}$ themselves comprise a sum over k of terms involving $\boldsymbol{\mu}_k$ and $\boldsymbol{\Lambda}_k$ leading to the further factorization

$$q(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}) = q(\boldsymbol{\pi}) \prod_{k=1}^K q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k). \quad (10.55)$$

Identifying the terms on the right-hand side of (10.54) that depend on π , we have

$$\ln q^*(\boldsymbol{\pi}) = (\alpha_0 - 1) \sum_{k=1}^K \ln \pi_k + \sum_{k=1}^K \sum_{n=1}^N r_{nk} \ln \pi_k + \text{const} \quad (10.56)$$

where we have used (10.50). Taking the exponential of both sides, we recognize $q^*(\boldsymbol{\pi})$ as a Dirichlet distribution

$$q^*(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi} | \boldsymbol{\alpha}) \quad (10.57)$$

where $\boldsymbol{\alpha}$ has components α_k given by

$$\alpha_k = \alpha_0 + N_k. \quad (10.58)$$

Finally, the variational posterior distribution $q^*(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)$ does not factorize into the product of the marginals, but we can always use the product rule to write it in the form $q^*(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) = q^*(\boldsymbol{\mu}_k | \boldsymbol{\Lambda}_k)q^*(\boldsymbol{\Lambda}_k)$. The two factors can be found by inspecting (10.54) and reading off those terms that involve $\boldsymbol{\mu}_k$ and $\boldsymbol{\Lambda}_k$. The result, as expected, is a Gaussian-Wishart distribution and is given by

$$q^*(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) = \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{m}_k, (\beta_k \boldsymbol{\Lambda}_k)^{-1}) \mathcal{W}(\boldsymbol{\Lambda}_k | \mathbf{W}_k, \nu_k) \quad (10.59)$$

where we have defined

$$\beta_k = \beta_0 + N_k \quad (10.60)$$

$$\mathbf{m}_k = \frac{1}{\beta_k} (\beta_0 \mathbf{m}_0 + N_k \bar{\mathbf{x}}_k) \quad (10.61)$$

$$\mathbf{W}_k^{-1} = \mathbf{W}_0^{-1} + N_k \mathbf{S}_k + \frac{\beta_0 N_k}{\beta_0 + N_k} (\bar{\mathbf{x}}_k - \mathbf{m}_0)(\bar{\mathbf{x}}_k - \mathbf{m}_0)^T \quad (10.62)$$

$$\nu_k = \nu_0 + N_k. \quad (10.63)$$

These update equations are analogous to the M-step equations of the EM algorithm for the maximum likelihood solution of the mixture of Gaussians. We see that the computations that must be performed in order to update the variational posterior distribution over the model parameters involve evaluation of the same sums over the data set, as arose in the maximum likelihood treatment.

In order to perform this variational M step, we need the expectations $\mathbb{E}[z_{nk}] = r_{nk}$ representing the responsibilities. These are obtained by normalizing the ρ_{nk} that are given by (10.46). We see that this expression involves expectations with respect to the variational distributions of the parameters, and these are easily evaluated to give

$$\begin{aligned} \mathbb{E}_{\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k} [(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Lambda}_k (\mathbf{x}_n - \boldsymbol{\mu}_k)] \\ = D\beta_k^{-1} + \nu_k (\mathbf{x}_n - \mathbf{m}_k)^T \mathbf{W}_k (\mathbf{x}_n - \mathbf{m}_k) \end{aligned} \quad (10.64)$$

$$\ln \tilde{\Lambda}_k \equiv \mathbb{E} [\ln |\boldsymbol{\Lambda}_k|] = \sum_{i=1}^D \psi \left(\frac{\nu_k + 1 - i}{2} \right) + D \ln 2 + \ln |\mathbf{W}_k| \quad (10.65)$$

$$\ln \tilde{\pi}_k \equiv \mathbb{E} [\ln \pi_k] = \psi(\alpha_k) - \psi(\hat{\alpha}) \quad (10.66)$$

Exercise 10.14

Appendix B

where we have introduced definitions of $\tilde{\Lambda}_k$ and $\tilde{\pi}_k$, and $\psi(\cdot)$ is the digamma function defined by (B.25), with $\hat{\alpha} = \sum_k \alpha_k$. The results (10.65) and (10.66) follow from the standard properties of the Wishart and Dirichlet distributions.

If we substitute (10.64), (10.65), and (10.66) into (10.46) and make use of (10.49), we obtain the following result for the responsibilities

$$r_{nk} \propto \tilde{\pi}_k \tilde{\Lambda}_k^{1/2} \exp \left\{ -\frac{D}{2\beta_k} - \frac{\nu_k}{2} (\mathbf{x}_n - \mathbf{m}_k)^T \mathbf{W}_k (\mathbf{x}_n - \mathbf{m}_k) \right\}. \quad (10.67)$$

Notice the similarity to the corresponding result for the responsibilities in maximum likelihood EM, which from (9.13) can be written in the form

$$r_{nk} \propto \pi_k |\Lambda_k|^{1/2} \exp \left\{ -\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \Lambda_k (\mathbf{x}_n - \boldsymbol{\mu}_k) \right\} \quad (10.68)$$

where we have used the precision in place of the covariance to highlight the similarity to (10.67).

Thus the optimization of the variational posterior distribution involves cycling between two stages analogous to the E and M steps of the maximum likelihood EM algorithm. In the variational equivalent of the E step, we use the current distributions over the model parameters to evaluate the moments in (10.64), (10.65), and (10.66) and hence evaluate $\mathbb{E}[z_{nk}] = r_{nk}$. Then in the subsequent variational equivalent of the M step, we keep these responsibilities fixed and use them to re-compute the variational distribution over the parameters using (10.57) and (10.59). In each case, we see that the variational posterior distribution has the same functional form as the corresponding factor in the joint distribution (10.41). This is a general result and is a consequence of the choice of conjugate distributions.

Section 10.4.1

Figure 10.6 shows the results of applying this approach to the rescaled Old Faithful data set for a Gaussian mixture model having $K = 6$ components. We see that after convergence, there are only two components for which the expected values of the mixing coefficients are numerically distinguishable from their prior values. This effect can be understood qualitatively in terms of the automatic trade-off in a Bayesian model between fitting the data and the complexity of the model, in which the complexity penalty arises from components whose parameters are pushed away from their prior values. Components that take essentially no responsibility for explaining the data points have $r_{nk} \simeq 0$ and hence $N_k \simeq 0$. From (10.58), we see that $\alpha_k \simeq \alpha_0$ and from (10.60)–(10.63) we see that the other parameters revert to their prior values. In principle such components are fitted slightly to the data points, but for broad priors this effect is too small to be seen numerically. For the variational Gaussian mixture model the expected values of the mixing coefficients in the posterior distribution are given by

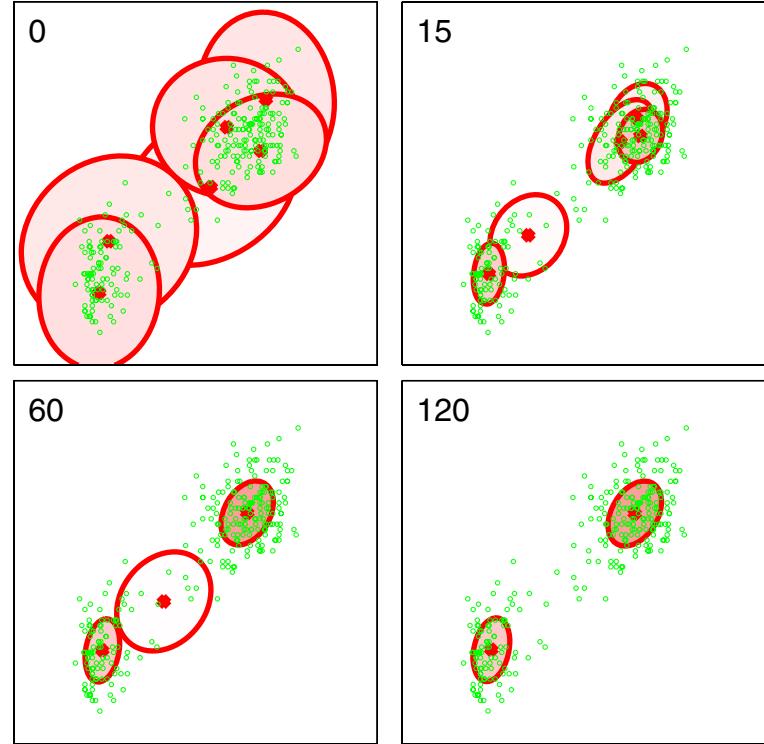
$$\mathbb{E}[\pi_k] = \frac{\alpha_k + N_k}{K\alpha_0 + N}. \quad (10.69)$$

Section 3.4

Exercise 10.15

Consider a component for which $N_k \simeq 0$ and $\alpha_k \simeq \alpha_0$. If the prior is broad so that $\alpha_0 \rightarrow 0$, then $\mathbb{E}[\pi_k] \rightarrow 0$ and the component plays no role in the model, whereas if

Figure 10.6 Variational Bayesian mixture of $K = 6$ Gaussians applied to the Old Faithful data set, in which the ellipses denote the one standard-deviation density contours for each of the components, and the density of red ink inside each ellipse corresponds to the mean value of the mixing coefficient for each component. The number in the top left of each diagram shows the number of iterations of variational inference. Components whose expected mixing coefficient are numerically indistinguishable from zero are not plotted.



the prior tightly constrains the mixing coefficients so that $\alpha_0 \rightarrow \infty$, then $\mathbb{E}[\pi_k] \rightarrow 1/K$.

In Figure 10.6, the prior over the mixing coefficients is a Dirichlet of the form (10.39). Recall from Figure 2.5 that for $\alpha_0 < 1$ the prior favours solutions in which some of the mixing coefficients are zero. Figure 10.6 was obtained using $\alpha_0 = 10^{-3}$, and resulted in two components having nonzero mixing coefficients. If instead we choose $\alpha_0 = 1$ we obtain three components with nonzero mixing coefficients, and for $\alpha = 10$ all six components have nonzero mixing coefficients.

As we have seen there is a close similarity between the variational solution for the Bayesian mixture of Gaussians and the EM algorithm for maximum likelihood. In fact if we consider the limit $N \rightarrow \infty$ then the Bayesian treatment converges to the maximum likelihood EM algorithm. For anything other than very small data sets, the dominant computational cost of the variational algorithm for Gaussian mixtures arises from the evaluation of the responsibilities, together with the evaluation and inversion of the weighted data covariance matrices. These computations mirror precisely those that arise in the maximum likelihood EM algorithm, and so there is little computational overhead in using this Bayesian approach as compared to the traditional maximum likelihood one. There are, however, some substantial advantages. First of all, the singularities that arise in maximum likelihood when a Gaussian component ‘collapses’ onto a specific data point are absent in the Bayesian treatment.

Indeed, these singularities are removed if we simply introduce a prior and then use a MAP estimate instead of maximum likelihood. Furthermore, there is no over-fitting if we choose a large number K of components in the mixture, as we saw in Figure 10.6. Finally, the variational treatment opens up the possibility of determining the optimal number of components in the mixture without resorting to techniques such as cross validation.

Section 10.2.4

10.2.2 Variational lower bound

We can also straightforwardly evaluate the lower bound (10.3) for this model. In practice, it is useful to be able to monitor the bound during the re-estimation in order to test for convergence. It can also provide a valuable check on both the mathematical expressions for the solutions and their software implementation, because at each step of the iterative re-estimation procedure the value of this bound should not decrease. We can take this a stage further to provide a deeper test of the correctness of both the mathematical derivation of the update equations and of their software implementation by using finite differences to check that each update does indeed give a (constrained) maximum of the bound (Svensén and Bishop, 2004).

For the variational mixture of Gaussians, the lower bound (10.3) is given by

$$\begin{aligned}\mathcal{L} &= \sum_{\mathbf{Z}} \iiint q(\mathbf{Z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}) \ln \left\{ \frac{p(\mathbf{X}, \mathbf{Z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda})}{q(\mathbf{Z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda})} \right\} d\boldsymbol{\pi} d\boldsymbol{\mu} d\boldsymbol{\Lambda} \\ &= \mathbb{E}[\ln p(\mathbf{X}, \mathbf{Z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda})] - \mathbb{E}[\ln q(\mathbf{Z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda})] \\ &= \mathbb{E}[\ln p(\mathbf{X}|\mathbf{Z}, \boldsymbol{\mu}, \boldsymbol{\Lambda})] + \mathbb{E}[\ln p(\mathbf{Z}|\boldsymbol{\pi})] + \mathbb{E}[\ln p(\boldsymbol{\pi})] + \mathbb{E}[\ln p(\boldsymbol{\mu}, \boldsymbol{\Lambda})] \\ &\quad - \mathbb{E}[\ln q(\mathbf{Z})] - \mathbb{E}[\ln q(\boldsymbol{\pi})] - \mathbb{E}[\ln q(\boldsymbol{\mu}, \boldsymbol{\Lambda})]\end{aligned}\tag{10.70}$$

where, to keep the notation uncluttered, we have omitted the $*$ superscript on the q distributions, along with the subscripts on the expectation operators because each expectation is taken with respect to all of the random variables in its argument. The various terms in the bound are easily evaluated to give the following results

$$\begin{aligned}\mathbb{E}[\ln p(\mathbf{X}|\mathbf{Z}, \boldsymbol{\mu}, \boldsymbol{\Lambda})] &= \frac{1}{2} \sum_{k=1}^K N_k \left\{ \ln \tilde{\Lambda}_k - D\beta_k^{-1} - \nu_k \text{Tr}(\mathbf{S}_k \mathbf{W}_k) \right. \\ &\quad \left. - \nu_k (\bar{\mathbf{x}}_k - \mathbf{m}_k)^T \mathbf{W}_k (\bar{\mathbf{x}}_k - \mathbf{m}_k) - D \ln(2\pi) \right\}\end{aligned}\tag{10.71}$$

$$\mathbb{E}[\ln p(\mathbf{Z}|\boldsymbol{\pi})] = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \ln \tilde{\pi}_k\tag{10.72}$$

$$\mathbb{E}[\ln p(\boldsymbol{\pi})] = \ln C(\alpha_0) + (\alpha_0 - 1) \sum_{k=1}^K \ln \tilde{\pi}_k\tag{10.73}$$

Exercise 10.16

$$\begin{aligned}\mathbb{E}[\ln p(\boldsymbol{\mu}, \boldsymbol{\Lambda})] &= \frac{1}{2} \sum_{k=1}^K \left\{ D \ln(\beta_0/2\pi) + \ln \tilde{\Lambda}_k - \frac{D\beta_0}{\beta_k} \right. \\ &\quad \left. - \beta_0 \nu_k (\mathbf{m}_k - \mathbf{m}_0)^T \mathbf{W}_k (\mathbf{m}_k - \mathbf{m}_0) \right\} + K \ln B(\mathbf{W}_0, \nu_0) \\ &\quad + \frac{(\nu_0 - D - 1)}{2} \sum_{k=1}^K \ln \tilde{\Lambda}_k - \frac{1}{2} \sum_{k=1}^K \nu_k \text{Tr}(\mathbf{W}_0^{-1} \mathbf{W}_k)\end{aligned}\quad (10.74)$$

$$\mathbb{E}[\ln q(\mathbf{Z})] = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \ln r_{nk} \quad (10.75)$$

$$\mathbb{E}[\ln q(\boldsymbol{\pi})] = \sum_{k=1}^K (\alpha_k - 1) \ln \tilde{\pi}_k + \ln C(\boldsymbol{\alpha}) \quad (10.76)$$

$$\mathbb{E}[\ln q(\boldsymbol{\mu}, \boldsymbol{\Lambda})] = \sum_{k=1}^K \left\{ \frac{1}{2} \ln \tilde{\Lambda}_k + \frac{D}{2} \ln \left(\frac{\beta_k}{2\pi} \right) - \frac{D}{2} - H[q(\boldsymbol{\Lambda}_k)] \right\} \quad (10.77)$$

where D is the dimensionality of \mathbf{x} , $H[q(\boldsymbol{\Lambda}_k)]$ is the entropy of the Wishart distribution given by (B.82), and the coefficients $C(\boldsymbol{\alpha})$ and $B(\mathbf{W}, \nu)$ are defined by (B.23) and (B.79), respectively. Note that the terms involving expectations of the logs of the q distributions simply represent the negative entropies of those distributions. Some simplifications and combination of terms can be performed when these expressions are summed to give the lower bound. However, we have kept the expressions separate for ease of understanding.

Finally, it is worth noting that the lower bound provides an alternative approach for deriving the variational re-estimation equations obtained in Section 10.2.1. To do this we use the fact that, since the model has conjugate priors, the functional form of the factors in the variational posterior distribution is known, namely discrete for \mathbf{Z} , Dirichlet for $\boldsymbol{\pi}$, and Gaussian-Wishart for $(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)$. By taking general parametric forms for these distributions we can derive the form of the lower bound as a function of the parameters of the distributions. Maximizing the bound with respect to these parameters then gives the required re-estimation equations.

10.2.3 Predictive density

In applications of the Bayesian mixture of Gaussians model we will often be interested in the predictive density for a new value $\hat{\mathbf{x}}$ of the observed variable. Associated with this observation will be a corresponding latent variable $\hat{\mathbf{z}}$, and the predictive density is then given by

$$p(\hat{\mathbf{x}}|\mathbf{X}) = \sum_{\hat{\mathbf{z}}} \iiint p(\hat{\mathbf{x}}|\hat{\mathbf{z}}, \boldsymbol{\mu}, \boldsymbol{\Lambda}) p(\hat{\mathbf{z}}|\boldsymbol{\pi}) p(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}|\mathbf{X}) d\boldsymbol{\pi} d\boldsymbol{\mu} d\boldsymbol{\Lambda} \quad (10.78)$$

Exercise 10.18

where $p(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda} | \mathbf{X})$ is the (unknown) true posterior distribution of the parameters. Using (10.37) and (10.38) we can first perform the summation over $\widehat{\mathbf{z}}$ to give

$$p(\widehat{\mathbf{x}} | \mathbf{X}) = \sum_{k=1}^K \iiint \pi_k \mathcal{N}(\widehat{\mathbf{x}} | \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1}) p(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda} | \mathbf{X}) d\boldsymbol{\pi} d\boldsymbol{\mu} d\boldsymbol{\Lambda}. \quad (10.79)$$

Because the remaining integrations are intractable, we approximate the predictive density by replacing the true posterior distribution $p(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda} | \mathbf{X})$ with its variational approximation $q(\boldsymbol{\pi})q(\boldsymbol{\mu}, \boldsymbol{\Lambda})$ to give

$$p(\widehat{\mathbf{x}} | \mathbf{X}) = \sum_{k=1}^K \iiint \pi_k \mathcal{N}(\widehat{\mathbf{x}} | \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1}) q(\boldsymbol{\pi})q(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) d\boldsymbol{\pi} d\boldsymbol{\mu}_k d\boldsymbol{\Lambda}_k \quad (10.80)$$

where we have made use of the factorization (10.55) and in each term we have implicitly integrated out all variables $\{\boldsymbol{\mu}_j, \boldsymbol{\Lambda}_j\}$ for $j \neq k$. The remaining integrations can now be evaluated analytically giving a mixture of Student's t-distributions

$$p(\widehat{\mathbf{x}} | \mathbf{X}) = \frac{1}{\bar{\alpha}} \sum_{k=1}^K \alpha_k \text{St}(\widehat{\mathbf{x}} | \mathbf{m}_k, \mathbf{L}_k, \nu_k + 1 - D) \quad (10.81)$$

in which the k^{th} component has mean \mathbf{m}_k , and the precision is given by

$$\mathbf{L}_k = \frac{(\nu_k + 1 - D)\beta_k}{(1 + \beta_k)} \mathbf{W}_k \quad (10.82)$$

in which ν_k is given by (10.63). When the size N of the data set is large the predictive distribution (10.81) reduces to a mixture of Gaussians.

10.2.4 Determining the number of components

We have seen that the variational lower bound can be used to determine a posterior distribution over the number K of components in the mixture model. There is, however, one subtlety that needs to be addressed. For any given setting of the parameters in a Gaussian mixture model (except for specific degenerate settings), there will exist other parameter settings for which the density over the observed variables will be identical. These parameter values differ only through a re-labelling of the components. For instance, consider a mixture of two Gaussians and a single observed variable x , in which the parameters have the values $\pi_1 = a, \pi_2 = b, \mu_1 = c, \mu_2 = d, \sigma_1 = e, \sigma_2 = f$. Then the parameter values $\pi_1 = b, \pi_2 = a, \mu_1 = d, \mu_2 = c, \sigma_1 = f, \sigma_2 = e$, in which the two components have been exchanged, will by symmetry give rise to the same value of $p(x)$. If we have a mixture model comprising K components, then each parameter setting will be a member of a family of $K!$ equivalent settings.

In the context of maximum likelihood, this redundancy is irrelevant because the parameter optimization algorithm (for example EM) will, depending on the initialization of the parameters, find one specific solution, and the other equivalent solutions play no role. In a Bayesian setting, however, we marginalize over all possible

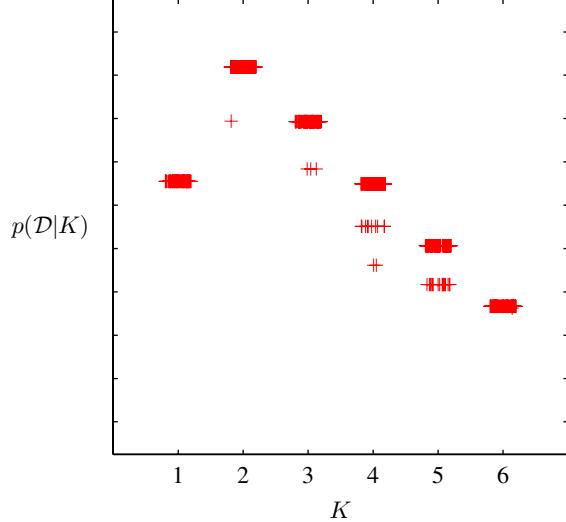
Exercise 10.19

Exercise 10.20

Section 10.1.4

Exercise 10.21

Figure 10.7 Plot of the variational lower bound \mathcal{L} versus the number K of components in the Gaussian mixture model, for the Old Faithful data, showing a distinct peak at $K = 2$ components. For each value of K , the model is trained from 100 different random starts, and the results shown as ‘+’ symbols plotted with small random horizontal perturbations so that they can be distinguished. Note that some solutions find suboptimal local maxima, but that this happens infrequently.



parameter values. We have seen in Figure 10.2 that if the true posterior distribution is multimodal, variational inference based on the minimization of $\text{KL}(q\|p)$ will tend to approximate the distribution in the neighbourhood of one of the modes and ignore the others. Again, because equivalent modes have equivalent predictive densities, this is of no concern provided we are considering a model having a specific number K of components. If, however, we wish to compare different values of K , then we need to take account of this multimodality. A simple approximate solution is to add a term $\ln K!$ onto the lower bound when used for model comparison and averaging.

Figure 10.7 shows a plot of the lower bound, including the multimodality factor, versus the number K of components for the Old Faithful data set. It is worth emphasizing once again that maximum likelihood would lead to values of the likelihood function that increase monotonically with K (assuming the singular solutions have been avoided, and discounting the effects of local maxima) and so cannot be used to determine an appropriate model complexity. By contrast, Bayesian inference automatically makes the trade-off between model complexity and fitting the data.

This approach to the determination of K requires that a range of models having different K values be trained and compared. An alternative approach to determining a suitable value for K is to treat the mixing coefficients π as parameters and make point estimates of their values by maximizing the lower bound (Corduneanu and Bishop, 2001) with respect to π instead of maintaining a probability distribution over them as in the fully Bayesian approach. This leads to the re-estimation equation

$$\pi_k = \frac{1}{N} \sum_{n=1}^N r_{nk} \quad (10.83)$$

and this maximization is interleaved with the variational updates for the q distribution over the remaining parameters. Components that provide insufficient contribution

Exercise 10.22

Section 3.4

Exercise 10.23

Section 7.2.2

to explaining the data will have their mixing coefficients driven to zero during the optimization, and so they are effectively removed from the model through *automatic relevance determination*. This allows us to make a single training run in which we start with a relatively large initial value of K , and allow surplus components to be pruned out of the model. The origins of the sparsity when optimizing with respect to hyperparameters is discussed in detail in the context of the relevance vector machine.

10.2.5 Induced factorizations

In deriving these variational update equations for the Gaussian mixture model, we assumed a particular factorization of the variational posterior distribution given by (10.42). However, the optimal solutions for the various factors exhibit additional factorizations. In particular, the solution for $q^*(\mu, \Lambda)$ is given by the product of an independent distribution $q^*(\mu_k, \Lambda_k)$ over each of the components k of the mixture, whereas the variational posterior distribution $q^*(\mathbf{Z})$ over the latent variables, given by (10.48), factorizes into an independent distribution $q^*(\mathbf{z}_n)$ for each observation n (note that it does not further factorize with respect to k because, for each value of n , the z_{nk} are constrained to sum to one over k). These additional factorizations are a consequence of the interaction between the assumed factorization and the conditional independence properties of the true distribution, as characterized by the directed graph in Figure 10.5.

We shall refer to these additional factorizations as *induced factorizations* because they arise from an interaction between the factorization assumed in the variational posterior distribution and the conditional independence properties of the true joint distribution. In a numerical implementation of the variational approach it is important to take account of such additional factorizations. For instance, it would be very inefficient to maintain a full precision matrix for the Gaussian distribution over a set of variables if the optimal form for that distribution always had a diagonal precision matrix (corresponding to a factorization with respect to the individual variables described by that Gaussian).

Such induced factorizations can easily be detected using a simple graphical test based on d-separation as follows. We partition the latent variables into three disjoint groups $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and then let us suppose that we are assuming a factorization between \mathbf{C} and the remaining latent variables, so that

$$q(\mathbf{A}, \mathbf{B}, \mathbf{C}) = q(\mathbf{A}, \mathbf{B})q(\mathbf{C}). \quad (10.84)$$

Using the general result (10.9), together with the product rule for probabilities, we see that the optimal solution for $q(\mathbf{A}, \mathbf{B})$ is given by

$$\begin{aligned} \ln q^*(\mathbf{A}, \mathbf{B}) &= \mathbb{E}_{\mathbf{C}}[\ln p(\mathbf{X}, \mathbf{A}, \mathbf{B}, \mathbf{C})] + \text{const} \\ &= \mathbb{E}_{\mathbf{C}}[\ln p(\mathbf{A}, \mathbf{B} | \mathbf{X}, \mathbf{C})] + \text{const}. \end{aligned} \quad (10.85)$$

We now ask whether this resulting solution will factorize between \mathbf{A} and \mathbf{B} , in other words whether $q^*(\mathbf{A}, \mathbf{B}) = q^*(\mathbf{A})q^*(\mathbf{B})$. This will happen if, and only if, $\ln p(\mathbf{A}, \mathbf{B} | \mathbf{X}, \mathbf{C}) = \ln p(\mathbf{A} | \mathbf{X}, \mathbf{C}) + \ln p(\mathbf{B} | \mathbf{X}, \mathbf{C})$, that is, if the conditional independence relation

$$\mathbf{A} \perp\!\!\!\perp \mathbf{B} | \mathbf{X}, \mathbf{C} \quad (10.86)$$

is satisfied. We can test to see if this relation does hold, for any choice of \mathbf{A} and \mathbf{B} by making use of the d-separation criterion.

To illustrate this, consider again the Bayesian mixture of Gaussians represented by the directed graph in Figure 10.5, in which we are assuming a variational factorization given by (10.42). We can see immediately that the variational posterior distribution over the parameters must factorize between π and the remaining parameters μ and Λ because all paths connecting π to either μ or Λ must pass through one of the nodes \mathbf{z}_n all of which are in the conditioning set for our conditional independence test and all of which are head-to-tail with respect to such paths.

10.3. Variational Linear Regression

As a second illustration of variational inference, we return to the Bayesian linear regression model of Section 3.3. In the evidence framework, we approximated the integration over α and β by making point estimates obtained by maximizing the log marginal likelihood. A fully Bayesian approach would integrate over the hyperparameters as well as over the parameters. Although exact integration is intractable, we can use variational methods to find a tractable approximation. In order to simplify the discussion, we shall suppose that the noise precision parameter β is known, and is fixed to its true value, although the framework is easily extended to include the distribution over β . For the linear regression model, the variational treatment will turn out to be equivalent to the evidence framework. Nevertheless, it provides a good exercise in the use of variational methods and will also lay the foundation for variational treatment of Bayesian logistic regression in Section 10.6.

Exercise 10.26

Recall that the likelihood function for \mathbf{w} , and the prior over \mathbf{w} , are given by

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^N \mathcal{N}(t_n | \mathbf{w}^T \phi_n, \beta^{-1}) \quad (10.87)$$

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1} \mathbf{I}) \quad (10.88)$$

where $\phi_n = \phi(\mathbf{x}_n)$. We now introduce a prior distribution over α . From our discussion in Section 2.3.6, we know that the conjugate prior for the precision of a Gaussian is given by a gamma distribution, and so we choose

$$p(\alpha) = \text{Gam}(\alpha|a_0, b_0) \quad (10.89)$$

where $\text{Gam}(\cdot|\cdot, \cdot)$ is defined by (B.26). Thus the joint distribution of all the variables is given by

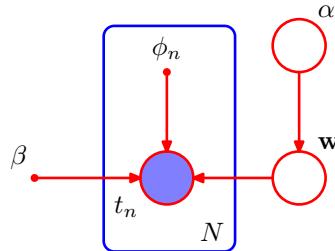
$$p(\mathbf{t}, \mathbf{w}, \alpha) = p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\alpha)p(\alpha). \quad (10.90)$$

This can be represented as a directed graphical model as shown in Figure 10.8.

10.3.1 Variational distribution

Our first goal is to find an approximation to the posterior distribution $p(\mathbf{w}, \alpha|\mathbf{t})$. To do this, we employ the variational framework of Section 10.1, with a variational

Figure 10.8 Probabilistic graphical model representing the joint distribution (10.90) for the Bayesian linear regression model.



posterior distribution given by the factorized expression

$$q(\mathbf{w}, \alpha) = q(\mathbf{w})q(\alpha). \quad (10.91)$$

We can find re-estimation equations for the factors in this distribution by making use of the general result (10.9). Recall that for each factor, we take the log of the joint distribution over all variables and then average with respect to those variables not in that factor. Consider first the distribution over α . Keeping only terms that have a functional dependence on α , we have

$$\begin{aligned} \ln q^*(\alpha) &= \ln p(\alpha) + \mathbb{E}_{\mathbf{w}} [\ln p(\mathbf{w}|\alpha)] + \text{const} \\ &= (a_0 - 1) \ln \alpha - b_0 \alpha + \frac{M}{2} \ln \alpha - \frac{\alpha}{2} \mathbb{E}[\mathbf{w}^T \mathbf{w}] + \text{const}. \end{aligned} \quad (10.92)$$

We recognize this as the log of a gamma distribution, and so identifying the coefficients of α and $\ln \alpha$ we obtain

$$q^*(\alpha) = \text{Gam}(\alpha | a_N, b_N) \quad (10.93)$$

where

$$a_N = a_0 + \frac{M}{2} \quad (10.94)$$

$$b_N = b_0 + \frac{1}{2} \mathbb{E}[\mathbf{w}^T \mathbf{w}]. \quad (10.95)$$

Similarly, we can find the variational re-estimation equation for the posterior distribution over \mathbf{w} . Again, using the general result (10.9), and keeping only those terms that have a functional dependence on \mathbf{w} , we have

$$\ln q^*(\mathbf{w}) = \ln p(\mathbf{t}|\mathbf{w}) + \mathbb{E}_{\alpha} [\ln p(\mathbf{w}|\alpha)] + \text{const} \quad (10.96)$$

$$= -\frac{\beta}{2} \sum_{n=1}^N \{\mathbf{w}^T \boldsymbol{\phi}_n - t_n\}^2 - \frac{1}{2} \mathbb{E}[\alpha] \mathbf{w}^T \mathbf{w} + \text{const} \quad (10.97)$$

$$= -\frac{1}{2} \mathbf{w}^T (\mathbb{E}[\alpha] \mathbf{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}) \mathbf{w} + \beta \mathbf{w}^T \boldsymbol{\Phi}^T \mathbf{t} + \text{const}. \quad (10.98)$$

Because this is a quadratic form, the distribution $q^*(\mathbf{w})$ is Gaussian, and so we can complete the square in the usual way to identify the mean and covariance, giving

$$q^*(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{m}_N, \mathbf{S}_N) \quad (10.99)$$

where

$$\mathbf{m}_N = \beta \mathbf{S}_N \boldsymbol{\Phi}^T \mathbf{t} \quad (10.100)$$

$$\mathbf{S}_N = (\mathbb{E}[\alpha] \mathbf{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1}. \quad (10.101)$$

Note the close similarity to the posterior distribution (3.52) obtained when α was treated as a fixed parameter. The difference is that here α is replaced by its expectation $\mathbb{E}[\alpha]$ under the variational distribution. Indeed, we have chosen to use the same notation for the covariance matrix \mathbf{S}_N in both cases.

Using the standard results (B.27), (B.38), and (B.39), we can obtain the required moments as follows

$$\mathbb{E}[\alpha] = a_N/b_N \quad (10.102)$$

$$\mathbb{E}[\mathbf{w}\mathbf{w}^T] = \mathbf{m}_N \mathbf{m}_N^T + \mathbf{S}_N. \quad (10.103)$$

The evaluation of the variational posterior distribution begins by initializing the parameters of one of the distributions $q(\mathbf{w})$ or $q(\alpha)$, and then alternately re-estimates these factors in turn until a suitable convergence criterion is satisfied (usually specified in terms of the lower bound to be discussed shortly).

It is instructive to relate the variational solution to that found using the evidence framework in Section 3.5. To do this consider the case $a_0 = b_0 = 0$, corresponding to the limit of an infinitely broad prior over α . The mean of the variational posterior distribution $q(\alpha)$ is then given by

$$\mathbb{E}[\alpha] = \frac{a_N}{b_N} = \frac{M/2}{\mathbb{E}[\mathbf{w}^T \mathbf{w}] / 2} = \frac{M}{\mathbf{m}_N^T \mathbf{m}_N + \text{Tr}(\mathbf{S}_N)}. \quad (10.104)$$

Comparison with (9.63) shows that in the case of this particularly simple model, the variational approach gives precisely the same expression as that obtained by maximizing the evidence function using EM except that the point estimate for α is replaced by its expected value. Because the distribution $q(\mathbf{w})$ depends on $q(\alpha)$ only through the expectation $\mathbb{E}[\alpha]$, we see that the two approaches will give identical results for the case of an infinitely broad prior.

10.3.2 Predictive distribution

The predictive distribution over t , given a new input \mathbf{x} , is easily evaluated for this model using the Gaussian variational posterior for the parameters

$$\begin{aligned} p(t|\mathbf{x}, \mathbf{t}) &= \int p(t|\mathbf{x}, \mathbf{w}) p(\mathbf{w}|\mathbf{t}) d\mathbf{w} \\ &\simeq \int p(t|\mathbf{x}, \mathbf{w}) q(\mathbf{w}) d\mathbf{w} \\ &= \int \mathcal{N}(t|\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}), \beta^{-1}) \mathcal{N}(\mathbf{w}|\mathbf{m}_N, \mathbf{S}_N) d\mathbf{w} \\ &= \mathcal{N}(t|\mathbf{m}_N^T \boldsymbol{\phi}(\mathbf{x}), \sigma^2(\mathbf{x})) \end{aligned} \quad (10.105)$$

where we have evaluated the integral by making use of the result (2.115) for the linear-Gaussian model. Here the input-dependent variance is given by

$$\sigma^2(\mathbf{x}) = \frac{1}{\beta} + \phi(\mathbf{x})^T \mathbf{S}_N \phi(\mathbf{x}). \quad (10.106)$$

Note that this takes the same form as the result (3.59) obtained with fixed α except that now the expected value $\mathbb{E}[\alpha]$ appears in the definition of \mathbf{S}_N .

10.3.3 Lower bound

Another quantity of importance is the lower bound \mathcal{L} defined by

$$\begin{aligned} \mathcal{L}(q) &= \mathbb{E}[\ln p(\mathbf{w}, \alpha, \mathbf{t})] - \mathbb{E}[\ln q(\mathbf{w}, \alpha)] \\ &= \mathbb{E}_{\mathbf{w}}[\ln p(\mathbf{t}|\mathbf{w})] + \mathbb{E}_{\mathbf{w}, \alpha}[\ln p(\mathbf{w}|\alpha)] + \mathbb{E}_{\alpha}[\ln p(\alpha)] \\ &\quad - \mathbb{E}_{\alpha}[\ln q(\mathbf{w})]_{\mathbf{w}} - \mathbb{E}[\ln q(\alpha)]. \end{aligned} \quad (10.107)$$

Exercise 10.27

Evaluation of the various terms is straightforward, making use of results obtained in previous chapters, and gives

$$\begin{aligned} \mathbb{E}[\ln p(\mathbf{t}|\mathbf{w})]_{\mathbf{w}} &= \frac{N}{2} \ln \left(\frac{\beta}{2\pi} \right) - \frac{\beta}{2} \mathbf{t}^T \mathbf{t} + \beta \mathbf{m}_N^T \Phi^T \mathbf{t} \\ &\quad - \frac{\beta}{2} \text{Tr} [\Phi^T \Phi (\mathbf{m}_N \mathbf{m}_N^T + \mathbf{S}_N)] \end{aligned} \quad (10.108)$$

$$\begin{aligned} \mathbb{E}[\ln p(\mathbf{w}|\alpha)]_{\mathbf{w}, \alpha} &= -\frac{M}{2} \ln(2\pi) + \frac{M}{2} (\psi(a_N) - \ln b_N) \\ &\quad - \frac{a_N}{2b_N} [\mathbf{m}_N^T \mathbf{m}_N + \text{Tr}(\mathbf{S}_N)] \end{aligned} \quad (10.109)$$

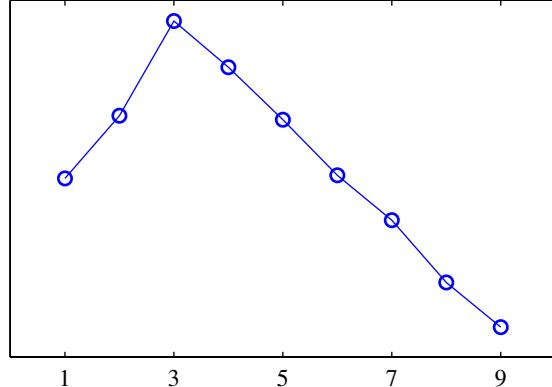
$$\begin{aligned} \mathbb{E}[\ln p(\alpha)]_{\alpha} &= a_0 \ln b_0 + (a_0 - 1) [\psi(a_N) - \ln b_N] \\ &\quad - b_0 \frac{a_N}{b_N} - \ln \Gamma(a_N) \end{aligned} \quad (10.110)$$

$$-\mathbb{E}[\ln q(\mathbf{w})]_{\mathbf{w}} = \frac{1}{2} \ln |\mathbf{S}_N| + \frac{M}{2} [1 + \ln(2\pi)] \quad (10.111)$$

$$-\mathbb{E}[\ln q(\alpha)]_{\alpha} = \ln \Gamma(a_N) - (a_N - 1)\psi(a_N) - \ln b_N + a_N. \quad (10.112)$$

Figure 10.9 shows a plot of the lower bound $\mathcal{L}(q)$ versus the degree of a polynomial model for a synthetic data set generated from a degree three polynomial. Here the prior parameters have been set to $a_0 = b_0 = 0$, corresponding to the noninformative prior $p(\alpha) \propto 1/\alpha$, which is uniform over $\ln \alpha$ as discussed in Section 2.3.6. As we saw in Section 10.1, the quantity \mathcal{L} represents lower bound on the log marginal likelihood $p(\mathbf{t}|M)$ for the model. If we assign equal prior probabilities $p(M)$ to the different values of M , then we can interpret \mathcal{L} as an approximation to the posterior model probability $p(M|\mathbf{t})$. Thus the variational framework assigns the highest probability to the model with $M = 3$. This should be contrasted with the maximum likelihood result, which assigns ever smaller residual error to models of increasing complexity until the residual error is driven to zero, causing maximum likelihood to favour severely over-fitted models.

Figure 10.9 Plot of the lower bound \mathcal{L} versus the order M of the polynomial, for a polynomial model, in which a set of 10 data points is generated from a polynomial with $M = 3$ sampled over the interval $(-5, 5)$ with additive Gaussian noise of variance 0.09. The value of the bound gives the log probability of the model, and we see that the value of the bound peaks at $M = 3$, corresponding to the true model from which the data set was generated.



10.4. Exponential Family Distributions

In Chapter 2, we discussed the important role played by the exponential family of distributions and their conjugate priors. For many of the models discussed in this book, the complete-data likelihood is drawn from the exponential family. However, in general this will not be the case for the marginal likelihood function for the observed data. For example, in a mixture of Gaussians, the joint distribution of observations \mathbf{x}_n and corresponding hidden variables \mathbf{z}_n is a member of the exponential family, whereas the marginal distribution of \mathbf{x}_n is a mixture of Gaussians and hence is not.

Up to now we have grouped the variables in the model into observed variables and hidden variables. We now make a further distinction between latent variables, denoted \mathbf{Z} , and parameters, denoted $\boldsymbol{\theta}$, where parameters are *intensive* (fixed in number independent of the size of the data set), whereas latent variables are *extensive* (scale in number with the size of the data set). For example, in a Gaussian mixture model, the indicator variables z_{kn} (which specify which component k is responsible for generating data point \mathbf{x}_n) represent the latent variables, whereas the means $\boldsymbol{\mu}_k$, precisions $\boldsymbol{\Lambda}_k$ and mixing proportions π_k represent the parameters.

Consider the case of independent identically distributed data. We denote the data values by $\mathbf{X} = \{\mathbf{x}_n\}$, where $n = 1, \dots, N$, with corresponding latent variables $\mathbf{Z} = \{\mathbf{z}_n\}$. Now suppose that the joint distribution of observed and latent variables is a member of the exponential family, parameterized by natural parameters $\boldsymbol{\eta}$ so that

$$p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\eta}) = \prod_{n=1}^N h(\mathbf{x}_n, \mathbf{z}_n) g(\boldsymbol{\eta}) \exp \left\{ \boldsymbol{\eta}^T \mathbf{u}(\mathbf{x}_n, \mathbf{z}_n) \right\}. \quad (10.113)$$

We shall also use a conjugate prior for $\boldsymbol{\eta}$, which can be written as

$$p(\boldsymbol{\eta} | \nu_0, \mathbf{v}_0) = f(\nu_0, \mathbf{v}_0) g(\boldsymbol{\eta})^{\nu_0} \exp \left\{ \nu_0 \boldsymbol{\eta}^T \mathbf{v}_0 \right\}. \quad (10.114)$$

Recall that the conjugate prior distribution can be interpreted as a prior number ν_0 of observations all having the value \mathbf{v}_0 for the \mathbf{u} vector. Now consider a variational

distribution that factorizes between the latent variables and the parameters, so that $q(\mathbf{Z}, \boldsymbol{\eta}) = q(\mathbf{Z})q(\boldsymbol{\eta})$. Using the general result (10.9), we can solve for the two factors as follows

$$\begin{aligned}\ln q^*(\mathbf{Z}) &= \mathbb{E}_{\boldsymbol{\eta}}[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\eta})] + \text{const} \\ &= \sum_{n=1}^N \left\{ \ln h(\mathbf{x}_n, \mathbf{z}_n) + \mathbb{E}[\boldsymbol{\eta}^T] \mathbf{u}(\mathbf{x}_n, \mathbf{z}_n) \right\} + \text{const.}\end{aligned}\quad (10.115)$$

Thus we see that this decomposes into a sum of independent terms, one for each value of n , and hence the solution for $q^*(\mathbf{Z})$ will factorize over n so that $q^*(\mathbf{Z}) = \prod_n q^*(\mathbf{z}_n)$. This is an example of an induced factorization. Taking the exponential of both sides, we have

$$q^*(\mathbf{z}_n) = h(\mathbf{x}_n, \mathbf{z}_n)g(\mathbb{E}[\boldsymbol{\eta}]) \exp\left\{\mathbb{E}[\boldsymbol{\eta}^T] \mathbf{u}(\mathbf{x}_n, \mathbf{z}_n)\right\} \quad (10.116)$$

where the normalization coefficient has been re-instated by comparison with the standard form for the exponential family.

Similarly, for the variational distribution over the parameters, we have

$$\ln q^*(\boldsymbol{\eta}) = \ln p(\boldsymbol{\eta}|\nu_0, \boldsymbol{\chi}_0) + \mathbb{E}_{\mathbf{Z}}[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\eta})] + \text{const} \quad (10.117)$$

$$= \nu_0 \ln g(\boldsymbol{\eta}) + \boldsymbol{\eta}^T \boldsymbol{\chi}_0 + \sum_{n=1}^N \left\{ \ln g(\boldsymbol{\eta}) + \boldsymbol{\eta}^T \mathbb{E}_{\mathbf{z}_n}[\mathbf{u}(\mathbf{x}_n, \mathbf{z}_n)] \right\} + \text{const.} \quad (10.118)$$

Again, taking the exponential of both sides, and re-instanting the normalization coefficient by inspection, we have

$$q^*(\boldsymbol{\eta}) = f(\nu_N, \boldsymbol{\chi}_N)g(\boldsymbol{\eta})^{\nu_N} \exp\left\{\boldsymbol{\eta}^T \boldsymbol{\chi}_N\right\} \quad (10.119)$$

where we have defined

$$\nu_N = \nu_0 + N \quad (10.120)$$

$$\boldsymbol{\chi}_N = \boldsymbol{\chi}_0 + \sum_{n=1}^N \mathbb{E}_{\mathbf{z}_n}[\mathbf{u}(\mathbf{x}_n, \mathbf{z}_n)]. \quad (10.121)$$

Note that the solutions for $q^*(\mathbf{z}_n)$ and $q^*(\boldsymbol{\eta})$ are coupled, and so we solve them iteratively in a two-stage procedure. In the variational E step, we evaluate the expected sufficient statistics $\mathbb{E}[\mathbf{u}(\mathbf{x}_n, \mathbf{z}_n)]$ using the current posterior distribution $q(\mathbf{z}_n)$ over the latent variables and use this to compute a revised posterior distribution $q(\boldsymbol{\eta})$ over the parameters. Then in the subsequent variational M step, we use this revised parameter posterior distribution to find the expected natural parameters $\mathbb{E}[\boldsymbol{\eta}^T]$, which gives rise to a revised variational distribution over the latent variables.

10.4.1 Variational message passing

We have illustrated the application of variational methods by considering a specific model, the Bayesian mixture of Gaussians, in some detail. This model can be

described by the directed graph shown in Figure 10.5. Here we consider more generally the use of variational methods for models described by directed graphs and derive a number of widely applicable results.

The joint distribution corresponding to a directed graph can be written using the decomposition

$$p(\mathbf{x}) = \prod_i p(\mathbf{x}_i | \text{pa}_i) \quad (10.122)$$

where \mathbf{x}_i denotes the variable(s) associated with node i , and pa_i denotes the parent set corresponding to node i . Note that \mathbf{x}_i may be a latent variable or it may belong to the set of observed variables. Now consider a variational approximation in which the distribution $q(\mathbf{x})$ is assumed to factorize with respect to the \mathbf{x}_i so that

$$q(\mathbf{x}) = \prod_i q_i(\mathbf{x}_i). \quad (10.123)$$

Note that for observed nodes, there is no factor $q(\mathbf{x}_i)$ in the variational distribution. We now substitute (10.122) into our general result (10.9) to give

$$\ln q_j^*(\mathbf{x}_j) = \mathbb{E}_{i \neq j} \left[\sum_i \ln p(\mathbf{x}_i | \text{pa}_i) \right] + \text{const.} \quad (10.124)$$

Any terms on the right-hand side that do not depend on \mathbf{x}_j can be absorbed into the additive constant. In fact, the only terms that do depend on \mathbf{x}_j are the conditional distribution for \mathbf{x}_j given by $p(\mathbf{x}_j | \text{pa}_j)$ together with any other conditional distributions that have \mathbf{x}_j in the conditioning set. By definition, these conditional distributions correspond to the children of node j , and they therefore also depend on the *co-parents* of the child nodes, i.e., the other parents of the child nodes besides node \mathbf{x}_j itself. We see that the set of all nodes on which $q^*(\mathbf{x}_j)$ depends corresponds to the Markov blanket of node \mathbf{x}_j , as illustrated in Figure 8.26. Thus the update of the factors in the variational posterior distribution represents a local calculation on the graph. This makes possible the construction of general purpose software for variational inference in which the form of the model does not need to be specified in advance (Bishop *et al.*, 2003).

If we now specialize to the case of a model in which all of the conditional distributions have a conjugate-exponential structure, then the variational update procedure can be cast in terms of a local message passing algorithm (Winn and Bishop, 2005). In particular, the distribution associated with a particular node can be updated once that node has received messages from all of its parents and all of its children. This in turn requires that the children have already received messages from their co-parents. The evaluation of the lower bound can also be simplified because many of the required quantities are already evaluated as part of the message passing scheme. This distributed message passing formulation has good scaling properties and is well suited to large networks.

10.5. Local Variational Methods

The variational framework discussed in Sections 10.1 and 10.2 can be considered a ‘global’ method in the sense that it directly seeks an approximation to the full posterior distribution over all random variables. An alternative ‘local’ approach involves finding bounds on functions over individual variables or groups of variables within a model. For instance, we might seek a bound on a conditional distribution $p(y|x)$, which is itself just one factor in a much larger probabilistic model specified by a directed graph. The purpose of introducing the bound of course is to simplify the resulting distribution. This local approximation can be applied to multiple variables in turn until a tractable approximation is obtained, and in Section 10.6.1 we shall give a practical example of this approach in the context of logistic regression. Here we focus on developing the bounds themselves.

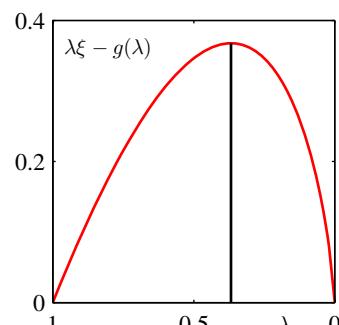
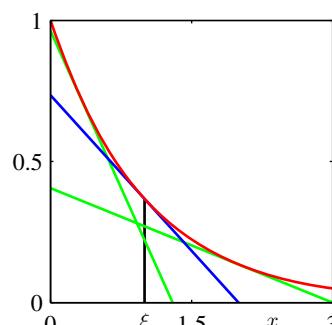
We have already seen in our discussion of the Kullback-Leibler divergence that the convexity of the logarithm function played a key role in developing the lower bound in the global variational approach. We have defined a (strictly) convex function as one for which every chord lies above the function. Convexity also plays a central role in the local variational framework. Note that our discussion will apply equally to concave functions with ‘min’ and ‘max’ interchanged and with lower bounds replaced by upper bounds.

Let us begin by considering a simple example, namely the function $f(x) = \exp(-x)$, which is a convex function of x , and which is shown in the left-hand plot of Figure 10.10. Our goal is to approximate $f(x)$ by a simpler function, in particular a linear function of x . From Figure 10.10, we see that this linear function will be a lower bound on $f(x)$ if it corresponds to a tangent. We can obtain the tangent line $y(x)$ at a specific value of x , say $x = \xi$, by making a first order Taylor expansion

$$y(x) = f(\xi) + f'(\xi)(x - \xi) \quad (10.125)$$

so that $y(x) \leq f(x)$ with equality when $x = \xi$. For our example function $f(x) =$

Figure 10.10 In the left-hand figure the red curve shows the function $\exp(-x)$, and the blue line shows the tangent at $x = \xi$ defined by (10.125) with $\xi = 1$. This line has slope $\lambda = f'(\xi) = -\exp(-\xi)$. Note that any other tangent line, for example the ones shown in green, will have a smaller value of y at $x = \xi$. The right-hand figure shows the corresponding plot of the function $\lambda\xi - g(\lambda)$, where $g(\lambda)$ is given by (10.131), versus λ for $\xi = 1$, in which the maximum corresponds to $\lambda = -\exp(-\xi) = -1/e$.



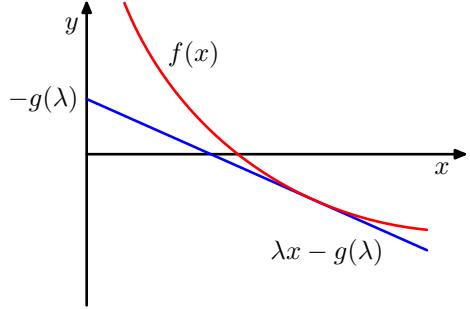
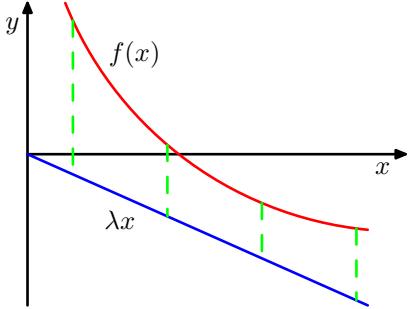


Figure 10.11 In the left-hand plot the red curve shows a convex function $f(x)$, and the blue line represents the linear function λx , which is a lower bound on $f(x)$ because $f(x) > \lambda x$ for all x . For the given value of slope λ the contact point of the tangent line having the same slope is found by minimizing with respect to x the discrepancy (shown by the green dashed lines) given by $f(x) - \lambda x$. This defines the dual function $g(\lambda)$, which corresponds to the (negative of the) intercept of the tangent line having slope λ .

$\exp(-x)$, we therefore obtain the tangent line in the form

$$y(x) = \exp(-\xi) - \exp(-\xi)(x - \xi) \quad (10.126)$$

which is a linear function parameterized by ξ . For consistency with subsequent discussion, let us define $\lambda = -\exp(-\xi)$ so that

$$y(x, \lambda) = \lambda x - \lambda + \lambda \ln(-\lambda). \quad (10.127)$$

Different values of λ correspond to different tangent lines, and because all such lines are lower bounds on the function, we have $f(x) \geq y(x, \lambda)$. Thus we can write the function in the form

$$f(x) = \max_{\lambda} \{\lambda x - \lambda + \lambda \ln(-\lambda)\}. \quad (10.128)$$

We have succeeded in approximating the convex function $f(x)$ by a simpler, linear function $y(x, \lambda)$. The price we have paid is that we have introduced a variational parameter λ , and to obtain the tightest bound we must optimize with respect to λ .

We can formulate this approach more generally using the framework of *convex duality* (Rockafellar, 1972; Jordan *et al.*, 1999). Consider the illustration of a convex function $f(x)$ shown in the left-hand plot in Figure 10.11. In this example, the function λx is a lower bound on $f(x)$ but it is not the best lower bound that can be achieved by a linear function having slope λ , because the tightest bound is given by the tangent line. Let us write the equation of the tangent line, having slope λ as $\lambda x - g(\lambda)$ where the (negative) intercept $g(\lambda)$ clearly depends on the slope λ of the tangent. To determine the intercept, we note that the line must be moved vertically by an amount equal to the smallest vertical distance between the line and the function, as shown in Figure 10.11. Thus

$$\begin{aligned} g(\lambda) &= -\min_x \{f(x) - \lambda x\} \\ &= \max_x \{\lambda x - f(x)\}. \end{aligned} \quad (10.129)$$

Now, instead of fixing λ and varying x , we can consider a particular x and then adjust λ until the tangent plane is tangent at that particular x . Because the y value of the tangent line at a particular x is maximized when that value coincides with its contact point, we have

$$f(x) = \max_{\lambda} \{ \lambda x - g(\lambda) \}. \quad (10.130)$$

We see that the functions $f(x)$ and $g(\lambda)$ play a dual role, and are related through (10.129) and (10.130).

Let us apply these duality relations to our simple example $f(x) = \exp(-x)$. From (10.129) we see that the maximizing value of x is given by $\xi = -\ln(-\lambda)$, and back-substituting we obtain the conjugate function $g(\lambda)$ in the form

$$g(\lambda) = \lambda - \lambda \ln(-\lambda) \quad (10.131)$$

as obtained previously. The function $\lambda\xi - g(\lambda)$ is shown, for $\xi = 1$ in the right-hand plot in Figure 10.10. As a check, we can substitute (10.131) into (10.130), which gives the maximizing value of $\lambda = -\exp(-x)$, and back-substituting then recovers the original function $f(x) = \exp(-x)$.

For concave functions, we can follow a similar argument to obtain upper bounds, in which max' is replaced with 'min', so that

$$f(x) = \min_{\lambda} \{ \lambda x - g(\lambda) \} \quad (10.132)$$

$$g(\lambda) = \min_x \{ \lambda x - f(x) \}. \quad (10.133)$$

If the function of interest is not convex (or concave), then we cannot directly apply the method above to obtain a bound. However, we can first seek invertible transformations either of the function or of its argument which change it into a convex form. We then calculate the conjugate function and then transform back to the original variables.

An important example, which arises frequently in pattern recognition, is the logistic sigmoid function defined by

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \quad (10.134)$$

As it stands this function is neither convex nor concave. However, if we take the logarithm we obtain a function which is concave, as is easily verified by finding the second derivative. From (10.133) the corresponding conjugate function then takes the form

$$g(\lambda) = \min_x \{ \lambda x - f(x) \} = -\lambda \ln \lambda - (1 - \lambda) \ln(1 - \lambda) \quad (10.135)$$

which we recognize as the binary entropy function for a variable whose probability of having the value 1 is λ . Using (10.132), we then obtain an upper bound on the log sigmoid

$$\ln \sigma(x) \leq \lambda x - g(\lambda) \quad (10.136)$$

Exercise 10.30

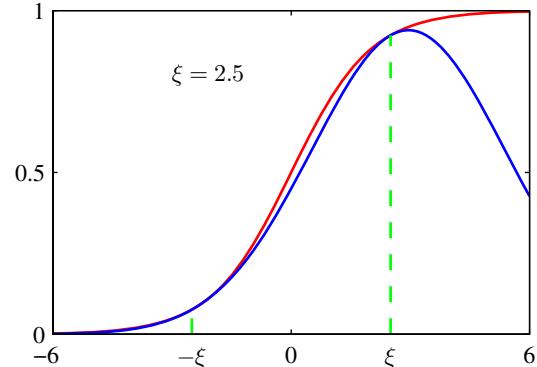
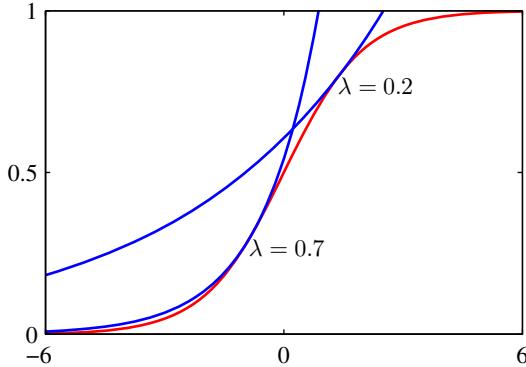


Figure 10.12 The left-hand plot shows the logistic sigmoid function $\sigma(x)$ defined by (10.134) in red, together with two examples of the exponential upper bound (10.137) shown in blue. The right-hand plot shows the logistic sigmoid again in red together with the Gaussian lower bound (10.144) shown in blue. Here the parameter $\xi = 2.5$, and the bound is exact at $x = \xi$ and $x = -\xi$, denoted by the dashed green lines.

and taking the exponential, we obtain an upper bound on the logistic sigmoid itself of the form

$$\sigma(x) \leq \exp(\lambda x - g(\lambda)) \quad (10.137)$$

which is plotted for two values of λ on the left-hand plot in Figure 10.12.

We can also obtain a lower bound on the sigmoid having the functional form of a Gaussian. To do this, we follow Jaakkola and Jordan (2000) and make transformations both of the input variable and of the function itself. First we take the log of the logistic function and then decompose it so that

$$\begin{aligned} \ln \sigma(x) &= -\ln(1 + e^{-x}) = -\ln \{e^{-x/2}(e^{x/2} + e^{-x/2})\} \\ &= x/2 - \ln(e^{x/2} + e^{-x/2}). \end{aligned} \quad (10.138)$$

We now note that the function $f(x) = -\ln(e^{x/2} + e^{-x/2})$ is a convex function of the variable x^2 , as can again be verified by finding the second derivative. This leads to a lower bound on $f(x)$, which is a linear function of x^2 whose conjugate function is given by

$$g(\lambda) = \max_{x^2} \left\{ \lambda x^2 - f\left(\sqrt{x^2}\right) \right\}. \quad (10.139)$$

The stationarity condition leads to

$$0 = \lambda - \frac{dx}{dx^2} \frac{d}{dx} f(x) = \lambda + \frac{1}{4x} \tanh\left(\frac{x}{2}\right). \quad (10.140)$$

If we denote this value of x , corresponding to the contact point of the tangent line for this particular value of λ , by ξ , then we have

$$\lambda(\xi) = -\frac{1}{4\xi} \tanh\left(\frac{\xi}{2}\right) = -\frac{1}{2\xi} \left[\sigma(\xi) - \frac{1}{2} \right]. \quad (10.141)$$

Exercise 10.31

Instead of thinking of λ as the variational parameter, we can let ξ play this role as this leads to simpler expressions for the conjugate function, which is then given by

$$g(\lambda) = \lambda(\xi)\xi^2 - f(\xi) = \lambda(\xi)\xi^2 + \ln(e^{\xi/2} + e^{-\xi/2}). \quad (10.142)$$

Hence the bound on $f(x)$ can be written as

$$f(x) \geq \lambda x^2 - g(\lambda) = \lambda x^2 - \lambda \xi^2 - \ln(e^{\xi/2} + e^{-\xi/2}). \quad (10.143)$$

The bound on the sigmoid then becomes

$$\sigma(x) \geq \sigma(\xi) \exp \left\{ (x - \xi)/2 - \lambda(\xi)(x^2 - \xi^2) \right\} \quad (10.144)$$

where $\lambda(\xi)$ is defined by (10.141). This bound is illustrated in the right-hand plot of Figure 10.12. We see that the bound has the form of the exponential of a quadratic function of x , which will prove useful when we seek Gaussian representations of posterior distributions defined through logistic sigmoid functions.

The logistic sigmoid arises frequently in probabilistic models over binary variables because it is the function that transforms a log odds ratio into a posterior probability. The corresponding transformation for a multiclass distribution is given by the softmax function. Unfortunately, the lower bound derived here for the logistic sigmoid does not directly extend to the softmax. Gibbs (1997) proposes a method for constructing a Gaussian distribution that is conjectured to be a bound (although no rigorous proof is given), which may be used to apply local variational methods to multiclass problems.

We shall see an example of the use of local variational bounds in Sections 10.6.1. For the moment, however, it is instructive to consider in general terms how these bounds can be used. Suppose we wish to evaluate an integral of the form

$$I = \int \sigma(a)p(a) da \quad (10.145)$$

where $\sigma(a)$ is the logistic sigmoid, and $p(a)$ is a Gaussian probability density. Such integrals arise in Bayesian models when, for instance, we wish to evaluate the predictive distribution, in which case $p(a)$ represents a posterior parameter distribution. Because the integral is intractable, we employ the variational bound (10.144), which we write in the form $\sigma(a) \geq f(a, \xi)$ where ξ is a variational parameter. The integral now becomes the product of two exponential-quadratic functions and so can be integrated analytically to give a bound on I

$$I \geq \int f(a, \xi)p(a) da = F(\xi). \quad (10.146)$$

We now have the freedom to choose the variational parameter ξ , which we do by finding the value ξ^* that maximizes the function $F(\xi)$. The resulting value $F(\xi^*)$ represents the tightest bound within this family of bounds and can be used as an approximation to I . This optimized bound, however, will in general not be exact.

Section 4.5

Section 4.3

Although the bound $\sigma(a) \geq f(a, \xi)$ on the logistic sigmoid can be optimized exactly, the required choice for ξ depends on the value of a , so that the bound is exact for one value of a only. Because the quantity $F(\xi)$ is obtained by integrating over all values of a , the value of ξ^* represents a compromise, weighted by the distribution $p(a)$.

10.6. Variational Logistic Regression

We now illustrate the use of local variational methods by returning to the Bayesian logistic regression model studied in Section 4.5. There we focussed on the use of the Laplace approximation, while here we consider a variational treatment based on the approach of Jaakkola and Jordan (2000). Like the Laplace method, this also leads to a Gaussian approximation to the posterior distribution. However, the greater flexibility of the variational approximation leads to improved accuracy compared to the Laplace method. Furthermore (unlike the Laplace method), the variational approach is optimizing a well defined objective function given by a rigorous bound on the model evidence. Logistic regression has also been treated by Dybowski and Roberts (2005) from a Bayesian perspective using Monte Carlo sampling techniques.

10.6.1 Variational posterior distribution

Here we shall make use of a variational approximation based on the local bounds introduced in Section 10.5. This allows the likelihood function for logistic regression, which is governed by the logistic sigmoid, to be approximated by the exponential of a quadratic form. It is therefore again convenient to choose a conjugate Gaussian prior of the form (4.140). For the moment, we shall treat the hyperparameters \mathbf{m}_0 and \mathbf{S}_0 as fixed constants. In Section 10.6.3, we shall demonstrate how the variational formalism can be extended to the case where there are unknown hyperparameters whose values are to be inferred from the data.

In the variational framework, we seek to maximize a lower bound on the marginal likelihood. For the Bayesian logistic regression model, the marginal likelihood takes the form

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{w})p(\mathbf{w}) d\mathbf{w} = \int \left[\prod_{n=1}^N p(t_n|\mathbf{w}) \right] p(\mathbf{w}) d\mathbf{w}. \quad (10.147)$$

We first note that the conditional distribution for t can be written as

$$\begin{aligned} p(t|\mathbf{w}) &= \sigma(a)^t \{1 - \sigma(a)\}^{1-t} \\ &= \left(\frac{1}{1 + e^{-a}} \right)^t \left(1 - \frac{1}{1 + e^{-a}} \right)^{1-t} \\ &= e^{at} \frac{e^{-a}}{1 + e^{-a}} = e^{at} \sigma(-a) \end{aligned} \quad (10.148)$$

where $a = \mathbf{w}^T \phi$. In order to obtain a lower bound on $p(\mathbf{t})$, we make use of the variational lower bound on the logistic sigmoid function given by (10.144), which

we reproduce here for convenience

$$\sigma(z) \geq \sigma(\xi) \exp \left\{ (z - \xi)/2 - \lambda(\xi)(z^2 - \xi^2) \right\} \quad (10.149)$$

where

$$\lambda(\xi) = \frac{1}{2\xi} \left[\sigma(\xi) - \frac{1}{2} \right]. \quad (10.150)$$

We can therefore write

$$p(t|\mathbf{w}) = e^{at} \sigma(-a) \geq e^{at} \sigma(\xi) \exp \left\{ -(a + \xi)/2 - \lambda(\xi)(a^2 - \xi^2) \right\}. \quad (10.151)$$

Note that because this bound is applied to each of the terms in the likelihood function separately, there is a variational parameter ξ_n corresponding to each training set observation (ϕ_n, t_n) . Using $a = \mathbf{w}^T \phi$, and multiplying by the prior distribution, we obtain the following bound on the joint distribution of \mathbf{t} and \mathbf{w}

$$p(\mathbf{t}, \mathbf{w}) = p(\mathbf{t}|\mathbf{w})p(\mathbf{w}) \geq h(\mathbf{w}, \boldsymbol{\xi})p(\mathbf{w}) \quad (10.152)$$

where $\boldsymbol{\xi}$ denotes the set $\{\xi_n\}$ of variational parameters, and

$$\begin{aligned} h(\mathbf{w}, \boldsymbol{\xi}) &= \prod_{n=1}^N \sigma(\xi_n) \exp \left\{ \mathbf{w}^T \phi_n t_n - (\mathbf{w}^T \phi_n + \xi_n)/2 \right. \\ &\quad \left. - \lambda(\xi_n)([\mathbf{w}^T \phi_n]^2 - \xi_n^2) \right\}. \end{aligned} \quad (10.153)$$

Evaluation of the exact posterior distribution would require normalization of the left-hand side of this inequality. Because this is intractable, we work instead with the right-hand side. Note that the function on the right-hand side cannot be interpreted as a probability density because it is not normalized. Once it is normalized to give a variational posterior distribution $q(\mathbf{w})$, however, it no longer represents a bound.

Because the logarithm function is monotonically increasing, the inequality $A \geq B$ implies $\ln A \geq \ln B$. This gives a lower bound on the log of the joint distribution of \mathbf{t} and \mathbf{w} of the form

$$\begin{aligned} \ln \{p(\mathbf{t}|\mathbf{w})p(\mathbf{w})\} &\geq \ln p(\mathbf{w}) + \sum_{n=1}^N \left\{ \ln \sigma(\xi_n) + \mathbf{w}^T \phi_n t_n \right. \\ &\quad \left. - (\mathbf{w}^T \phi_n + \xi_n)/2 - \lambda(\xi_n)([\mathbf{w}^T \phi_n]^2 - \xi_n^2) \right\}. \end{aligned} \quad (10.154)$$

Substituting for the prior $p(\mathbf{w})$, the right-hand side of this inequality becomes, as a function of \mathbf{w}

$$\begin{aligned} &- \frac{1}{2} (\mathbf{w} - \mathbf{m}_0)^T \mathbf{S}_0^{-1} (\mathbf{w} - \mathbf{m}_0) \\ &+ \sum_{n=1}^N \left\{ \mathbf{w}^T \phi_n (t_n - 1/2) - \lambda(\xi_n) \mathbf{w}^T (\phi_n \phi_n^T) \mathbf{w} \right\} + \text{const.} \end{aligned} \quad (10.155)$$

This is a quadratic function of \mathbf{w} , and so we can obtain the corresponding variational approximation to the posterior distribution by identifying the linear and quadratic terms in \mathbf{w} , giving a Gaussian variational posterior of the form

$$q(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{m}_N, \mathbf{S}_N) \quad (10.156)$$

where

$$\mathbf{m}_N = \mathbf{S}_N \left(\mathbf{S}_0^{-1} \mathbf{m}_0 + \sum_{n=1}^N (t_n - 1/2) \boldsymbol{\phi}_n \right) \quad (10.157)$$

$$\mathbf{S}_N^{-1} = \mathbf{S}_0^{-1} + 2 \sum_{n=1}^N \lambda(\xi_n) \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T. \quad (10.158)$$

As with the Laplace framework, we have again obtained a Gaussian approximation to the posterior distribution. However, the additional flexibility provided by the variational parameters $\{\xi_n\}$ leads to improved accuracy in the approximation (Jaakkola and Jordan, 2000).

Here we have considered a batch learning context in which all of the training data is available at once. However, Bayesian methods are intrinsically well suited to sequential learning in which the data points are processed one at a time and then discarded. The formulation of this variational approach for the sequential case is straightforward.

Note that the bound given by (10.149) applies only to the two-class problem and so this approach does not directly generalize to classification problems with $K > 2$ classes. An alternative bound for the multiclass case has been explored by Gibbs (1997).

10.6.2 Optimizing the variational parameters

We now have a normalized Gaussian approximation to the posterior distribution, which we shall use shortly to evaluate the predictive distribution for new data points. First, however, we need to determine the variational parameters $\{\xi_n\}$ by maximizing the lower bound on the marginal likelihood.

To do this, we substitute the inequality (10.152) back into the marginal likelihood to give

$$\ln p(\mathbf{t}) = \ln \int p(\mathbf{t} | \mathbf{w}) p(\mathbf{w}) d\mathbf{w} \geq \ln \int h(\mathbf{w}, \boldsymbol{\xi}) p(\mathbf{w}) d\mathbf{w} = \mathcal{L}(\boldsymbol{\xi}). \quad (10.159)$$

As with the optimization of the hyperparameter α in the linear regression model of Section 3.5, there are two approaches to determining the ξ_n . In the first approach, we recognize that the function $\mathcal{L}(\boldsymbol{\xi})$ is defined by an integration over \mathbf{w} and so we can view \mathbf{w} as a latent variable and invoke the EM algorithm. In the second approach, we integrate over \mathbf{w} analytically and then perform a direct maximization over $\boldsymbol{\xi}$. Let us begin by considering the EM approach.

The EM algorithm starts by choosing some initial values for the parameters $\{\xi_n\}$, which we denote collectively by $\boldsymbol{\xi}^{\text{old}}$. In the E step of the EM algorithm,

Exercise 10.32

we then use these parameter values to find the posterior distribution over \mathbf{w} , which is given by (10.156). In the M step, we then maximize the expected complete-data log likelihood which is given by

$$Q(\boldsymbol{\xi}, \boldsymbol{\xi}^{\text{old}}) = \mathbb{E} [\ln h(\mathbf{w}, \boldsymbol{\xi}) p(\mathbf{w})] \quad (10.160)$$

where the expectation is taken with respect to the posterior distribution $q(\mathbf{w})$ evaluated using $\boldsymbol{\xi}^{\text{old}}$. Noting that $p(\mathbf{w})$ does not depend on $\boldsymbol{\xi}$, and substituting for $h(\mathbf{w}, \boldsymbol{\xi})$ we obtain

$$Q(\boldsymbol{\xi}, \boldsymbol{\xi}^{\text{old}}) = \sum_{n=1}^N \left\{ \ln \sigma(\xi_n) - \xi_n/2 - \lambda(\xi_n) (\boldsymbol{\phi}_n^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \boldsymbol{\phi}_n - \xi_n^2) \right\} + \text{const} \quad (10.161)$$

where ‘const’ denotes terms that are independent of $\boldsymbol{\xi}$. We now set the derivative with respect to ξ_n equal to zero. A few lines of algebra, making use of the definitions of $\sigma(\xi)$ and $\lambda(\xi)$, then gives

$$0 = \lambda'(\xi_n) (\boldsymbol{\phi}_n^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \boldsymbol{\phi}_n - \xi_n^2). \quad (10.162)$$

We now note that $\lambda'(\xi)$ is a monotonic function of ξ for $\xi \geq 0$, and that we can restrict attention to nonnegative values of ξ without loss of generality due to the symmetry of the bound around $\xi = 0$. Thus $\lambda'(\xi) \neq 0$, and hence we obtain the following re-estimation equations

$$(\xi_n^{\text{new}})^2 = \boldsymbol{\phi}_n^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \boldsymbol{\phi}_n = \boldsymbol{\phi}_n^T (\mathbf{S}_N + \mathbf{m}_N \mathbf{m}_N^T) \boldsymbol{\phi}_n \quad (10.163)$$

where we have used (10.156).

Let us summarize the EM algorithm for finding the variational posterior distribution. We first initialize the variational parameters $\boldsymbol{\xi}^{\text{old}}$. In the E step, we evaluate the posterior distribution over \mathbf{w} given by (10.156), in which the mean and covariance are defined by (10.157) and (10.158). In the M step, we then use this variational posterior to compute a new value for $\boldsymbol{\xi}$ given by (10.163). The E and M steps are repeated until a suitable convergence criterion is satisfied, which in practice typically requires only a few iterations.

An alternative approach to obtaining re-estimation equations for $\boldsymbol{\xi}$ is to note that in the integral over \mathbf{w} in the definition (10.159) of the lower bound $\mathcal{L}(\boldsymbol{\xi})$, the integrand has a Gaussian-like form and so the integral can be evaluated analytically. Having evaluated the integral, we can then differentiate with respect to ξ_n . It turns out that this gives rise to exactly the same re-estimation equations as does the EM approach given by (10.163).

As we have emphasized already, in the application of variational methods it is useful to be able to evaluate the lower bound $\mathcal{L}(\boldsymbol{\xi})$ given by (10.159). The integration over \mathbf{w} can be performed analytically by noting that $p(\mathbf{w})$ is Gaussian and $h(\mathbf{w}, \boldsymbol{\xi})$ is the exponential of a quadratic function of \mathbf{w} . Thus, by completing the square and making use of the standard result for the normalization coefficient of a Gaussian distribution, we can obtain a closed form solution which takes the form

Exercise 10.33

Exercise 10.34

Exercise 10.35

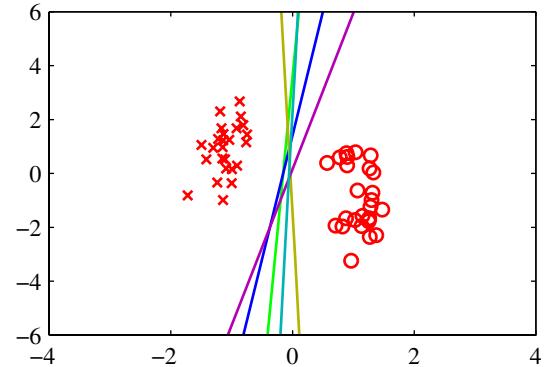
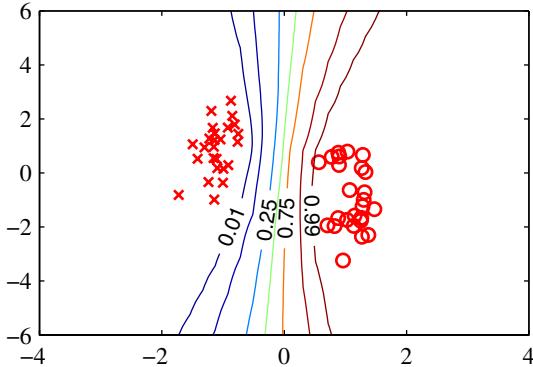


Figure 10.13 Illustration of the Bayesian approach to logistic regression for a simple linearly separable data set. The plot on the left shows the predictive distribution obtained using variational inference. We see that the decision boundary lies roughly mid way between the clusters of data points, and that the contours of the predictive distribution splay out away from the data reflecting the greater uncertainty in the classification of such regions. The plot on the right shows the decision boundaries corresponding to five samples of the parameter vector \mathbf{w} drawn from the posterior distribution $p(\mathbf{w}|\mathbf{t})$.

$$\begin{aligned}\mathcal{L}(\xi) = & \frac{1}{2} \ln \frac{|\mathbf{S}_N|}{|\mathbf{S}_0|} - \frac{1}{2} \mathbf{m}_N^T \mathbf{S}_N^{-1} \mathbf{m}_N + \frac{1}{2} \mathbf{m}_0^T \mathbf{S}_0^{-1} \mathbf{m}_0 \\ & + \sum_{n=1}^N \left\{ \ln \sigma(\xi_n) - \frac{1}{2} \xi_n - \lambda(\xi_n) \xi_n^2 \right\}. \end{aligned} \quad (10.164)$$

This variational framework can also be applied to situations in which the data is arriving sequentially (Jaakkola and Jordan, 2000). In this case we maintain a Gaussian posterior distribution over \mathbf{w} , which is initialized using the prior $p(\mathbf{w})$. As each data point arrives, the posterior is updated by making use of the bound (10.151) and then normalized to give an updated posterior distribution.

The predictive distribution is obtained by marginalizing over the posterior distribution, and takes the same form as for the Laplace approximation discussed in Section 4.5.2. Figure 10.13 shows the variational predictive distributions for a synthetic data set. This example provides interesting insights into the concept of ‘large margin’, which was discussed in Section 7.1 and which has qualitatively similar behaviour to the Bayesian solution.

10.6.3 Inference of hyperparameters

So far, we have treated the hyperparameter α in the prior distribution as a known constant. We now extend the Bayesian logistic regression model to allow the value of this parameter to be inferred from the data set. This can be achieved by combining the global and local variational approximations into a single framework, so as to maintain a lower bound on the marginal likelihood at each stage. Such a combined approach was adopted by Bishop and Svensén (2003) in the context of a Bayesian treatment of the hierarchical mixture of experts model.

Specifically, we consider once again a simple isotropic Gaussian prior distribution of the form

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}). \quad (10.165)$$

Our analysis is readily extended to more general Gaussian priors, for instance if we wish to associate a different hyperparameter with different subsets of the parameters w_j . As usual, we consider a conjugate hyperprior over α given by a gamma distribution

$$p(\alpha) = \text{Gam}(\alpha|a_0, b_0) \quad (10.166)$$

governed by the constants a_0 and b_0 .

The marginal likelihood for this model now takes the form

$$p(\mathbf{t}) = \iint p(\mathbf{w}, \alpha, \mathbf{t}) d\mathbf{w} d\alpha \quad (10.167)$$

where the joint distribution is given by

$$p(\mathbf{w}, \alpha, \mathbf{t}) = p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\alpha)p(\alpha). \quad (10.168)$$

We are now faced with an analytically intractable integration over \mathbf{w} and α , which we shall tackle by using both the local and global variational approaches in the same model

To begin with, we introduce a variational distribution $q(\mathbf{w}, \alpha)$, and then apply the decomposition (10.2), which in this instance takes the form

$$\ln p(\mathbf{t}) = \mathcal{L}(q) + \text{KL}(q\|p) \quad (10.169)$$

where the lower bound $\mathcal{L}(q)$ and the Kullback-Leibler divergence $\text{KL}(q\|p)$ are defined by

$$\mathcal{L}(q) = \iint q(\mathbf{w}, \alpha) \ln \left\{ \frac{p(\mathbf{w}, \alpha, \mathbf{t})}{q(\mathbf{w}, \alpha)} \right\} d\mathbf{w} d\alpha \quad (10.170)$$

$$\text{KL}(q\|p) = - \iint q(\mathbf{w}, \alpha) \ln \left\{ \frac{p(\mathbf{w}, \alpha|\mathbf{t})}{q(\mathbf{w}, \alpha)} \right\} d\mathbf{w} d\alpha. \quad (10.171)$$

At this point, the lower bound $\mathcal{L}(q)$ is still intractable due to the form of the likelihood factor $p(\mathbf{t}|\mathbf{w})$. We therefore apply the local variational bound to each of the logistic sigmoid factors as before. This allows us to use the inequality (10.152) and place a lower bound on $\mathcal{L}(q)$, which will therefore also be a lower bound on the log marginal likelihood

$$\begin{aligned} \ln p(\mathbf{t}) &\geq \mathcal{L}(q) \geq \widetilde{\mathcal{L}}(q, \boldsymbol{\xi}) \\ &= \iint q(\mathbf{w}, \alpha) \ln \left\{ \frac{h(\mathbf{w}, \boldsymbol{\xi})p(\mathbf{w}|\alpha)p(\alpha)}{q(\mathbf{w}, \alpha)} \right\} d\mathbf{w} d\alpha. \end{aligned} \quad (10.172)$$

Next we assume that the variational distribution factorizes between parameters and hyperparameters so that

$$q(\mathbf{w}, \alpha) = q(\mathbf{w})q(\alpha). \quad (10.173)$$

With this factorization we can appeal to the general result (10.9) to find expressions for the optimal factors. Consider first the distribution $q(\mathbf{w})$. Discarding terms that are independent of \mathbf{w} , we have

$$\begin{aligned}\ln q(\mathbf{w}) &= \mathbb{E}_\alpha [\ln \{h(\mathbf{w}, \boldsymbol{\xi})p(\mathbf{w}|\alpha)p(\alpha)\}] + \text{const} \\ &= \ln h(\mathbf{w}, \boldsymbol{\xi}) + \mathbb{E}_\alpha [\ln p(\mathbf{w}|\alpha)] + \text{const}.\end{aligned}$$

We now substitute for $\ln h(\mathbf{w}, \boldsymbol{\xi})$ using (10.153), and for $\ln p(\mathbf{w}|\alpha)$ using (10.165), giving

$$\ln q(\mathbf{w}) = -\frac{\mathbb{E}[\alpha]}{2} \mathbf{w}^T \mathbf{w} + \sum_{n=1}^N \{(t_n - 1/2) \mathbf{w}^T \boldsymbol{\phi}_n - \lambda(\xi_n) \mathbf{w}^T \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T \mathbf{w}\} + \text{const.}$$

We see that this is a quadratic function of \mathbf{w} and so the solution for $q(\mathbf{w})$ will be Gaussian. Completing the square in the usual way, we obtain

$$q(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N) \quad (10.174)$$

where we have defined

$$\boldsymbol{\Sigma}_N^{-1} \boldsymbol{\mu}_N = \sum_{n=1}^N (t_n - 1/2) \boldsymbol{\phi}_n \quad (10.175)$$

$$\boldsymbol{\Sigma}_N^{-1} = \mathbb{E}[\alpha] \mathbf{I} + 2 \sum_{n=1}^N \lambda(\xi_n) \boldsymbol{\phi}_n \boldsymbol{\phi}_n^T. \quad (10.176)$$

Similarly, the optimal solution for the factor $q(\alpha)$ is obtained from

$$\ln q(\alpha) = \mathbb{E}_{\mathbf{w}} [\ln p(\mathbf{w}|\alpha)] + \ln p(\alpha) + \text{const.}$$

Substituting for $\ln p(\mathbf{w}|\alpha)$ using (10.165), and for $\ln p(\alpha)$ using (10.166), we obtain

$$\ln q(\alpha) = \frac{M}{2} \ln \alpha - \frac{\alpha}{2} \mathbb{E} [\mathbf{w}^T \mathbf{w}] + (a_0 - 1) \ln \alpha - b_0 \alpha + \text{const.}$$

We recognize this as the log of a gamma distribution, and so we obtain

$$q(\alpha) = \text{Gam}(\alpha | a_N, b_N) = \frac{1}{\Gamma(a_0)} a_0^{b_0} \alpha^{a_0-1} e^{-b_0 \alpha} \quad (10.177)$$

where

$$a_N = a_0 + \frac{M}{2} \quad (10.178)$$

$$b_N = b_0 + \frac{1}{2} \mathbb{E}_{\mathbf{w}} [\mathbf{w}^T \mathbf{w}]. \quad (10.179)$$

We also need to optimize the variational parameters ξ_n , and this is also done by maximizing the lower bound $\tilde{\mathcal{L}}(q, \xi)$. Omitting terms that are independent of ξ , and integrating over α , we have

$$\tilde{\mathcal{L}}(q, \xi) = \int q(\mathbf{w}) \ln h(\mathbf{w}, \xi) d\mathbf{w} + \text{const.} \quad (10.180)$$

Note that this has precisely the same form as (10.159), and so we can again appeal to our earlier result (10.163), which can be obtained by direct optimization of the marginal likelihood function, leading to re-estimation equations of the form

$$(\xi_n^{\text{new}})^2 = \boldsymbol{\phi}_n^T (\boldsymbol{\Sigma}_N + \boldsymbol{\mu}_N \boldsymbol{\mu}_N^T) \boldsymbol{\phi}_n. \quad (10.181)$$

We have obtained re-estimation equations for the three quantities $q(\mathbf{w})$, $q(\alpha)$, and ξ , and so after making suitable initializations, we can cycle through these quantities, updating each in turn. The required moments are given by

$$\mathbb{E}[\alpha] = \frac{a_N}{b_N} \quad (10.182)$$

$$\mathbb{E}[\mathbf{w}^T \mathbf{w}] = \boldsymbol{\Sigma}_N + \boldsymbol{\mu}_N^T \boldsymbol{\mu}_N. \quad (10.183)$$

Appendix B

10.7. Expectation Propagation

We conclude this chapter by discussing an alternative form of deterministic approximate inference, known as *expectation propagation* or *EP* (Minka, 2001a; Minka, 2001b). As with the variational Bayes methods discussed so far, this too is based on the minimization of a Kullback-Leibler divergence but now of the reverse form, which gives the approximation rather different properties.

Consider for a moment the problem of minimizing $\text{KL}(p||q)$ with respect to $q(\mathbf{z})$ when $p(\mathbf{z})$ is a fixed distribution and $q(\mathbf{z})$ is a member of the exponential family and so, from (2.194), can be written in the form

$$q(\mathbf{z}) = h(\mathbf{z})g(\boldsymbol{\eta}) \exp\{\boldsymbol{\eta}^T \mathbf{u}(\mathbf{z})\}. \quad (10.184)$$

As a function of $\boldsymbol{\eta}$, the Kullback-Leibler divergence then becomes

$$\text{KL}(p||q) = -\ln g(\boldsymbol{\eta}) - \boldsymbol{\eta}^T \mathbb{E}_{p(\mathbf{z})}[\mathbf{u}(\mathbf{z})] + \text{const} \quad (10.185)$$

where the constant terms are independent of the natural parameters $\boldsymbol{\eta}$. We can minimize $\text{KL}(p||q)$ within this family of distributions by setting the gradient with respect to $\boldsymbol{\eta}$ to zero, giving

$$-\nabla \ln g(\boldsymbol{\eta}) = \mathbb{E}_{p(\mathbf{z})}[\mathbf{u}(\mathbf{z})]. \quad (10.186)$$

However, we have already seen in (2.226) that the negative gradient of $\ln g(\boldsymbol{\eta})$ is given by the expectation of $\mathbf{u}(\mathbf{z})$ under the distribution $q(\mathbf{z})$. Equating these two results, we obtain

$$\mathbb{E}_{q(\mathbf{z})}[\mathbf{u}(\mathbf{z})] = \mathbb{E}_{p(\mathbf{z})}[\mathbf{u}(\mathbf{z})]. \quad (10.187)$$

We see that the optimum solution simply corresponds to matching the expected sufficient statistics. So, for instance, if $q(\mathbf{z})$ is a Gaussian $\mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ then we minimize the Kullback-Leibler divergence by setting the mean $\boldsymbol{\mu}$ of $q(\mathbf{z})$ equal to the mean of the distribution $p(\mathbf{z})$ and the covariance $\boldsymbol{\Sigma}$ equal to the covariance of $p(\mathbf{z})$. This is sometimes called *moment matching*. An example of this was seen in Figure 10.3(a).

Now let us exploit this result to obtain a practical algorithm for approximate inference. For many probabilistic models, the joint distribution of data \mathcal{D} and hidden variables (including parameters) $\boldsymbol{\theta}$ comprises a product of factors in the form

$$p(\mathcal{D}, \boldsymbol{\theta}) = \prod_i f_i(\boldsymbol{\theta}). \quad (10.188)$$

This would arise, for example, in a model for independent, identically distributed data in which there is one factor $f_n(\boldsymbol{\theta}) = p(\mathbf{x}_n|\boldsymbol{\theta})$ for each data point \mathbf{x}_n , along with a factor $f_0(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$ corresponding to the prior. More generally, it would also apply to any model defined by a directed probabilistic graph in which each factor is a conditional distribution corresponding to one of the nodes, or an undirected graph in which each factor is a clique potential. We are interested in evaluating the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ for the purpose of making predictions, as well as the model evidence $p(\mathcal{D})$ for the purpose of model comparison. From (10.188) the posterior is given by

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{p(\mathcal{D})} \prod_i f_i(\boldsymbol{\theta}) \quad (10.189)$$

and the model evidence is given by

$$p(\mathcal{D}) = \int \prod_i f_i(\boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (10.190)$$

Here we are considering continuous variables, but the following discussion applies equally to discrete variables with integrals replaced by summations. We shall suppose that the marginalization over $\boldsymbol{\theta}$, along with the marginalizations with respect to the posterior distribution required to make predictions, are intractable so that some form of approximation is required.

Expectation propagation is based on an approximation to the posterior distribution which is also given by a product of factors

$$q(\boldsymbol{\theta}) = \frac{1}{Z} \prod_i \tilde{f}_i(\boldsymbol{\theta}) \quad (10.191)$$

in which each factor $\tilde{f}_i(\boldsymbol{\theta})$ in the approximation corresponds to one of the factors $f_i(\boldsymbol{\theta})$ in the true posterior (10.189), and the factor $1/Z$ is the normalizing constant needed to ensure that the left-hand side of (10.191) integrates to unity. In order to obtain a practical algorithm, we need to constrain the factors $\tilde{f}_i(\boldsymbol{\theta})$ in some way, and in particular we shall assume that they come from the exponential family. The product of the factors will therefore also be from the exponential family and so can

be described by a finite set of sufficient statistics. For example, if each of the $\tilde{f}_i(\theta)$ is a Gaussian, then the overall approximation $q(\theta)$ will also be Gaussian.

Ideally we would like to determine the $\tilde{f}_i(\theta)$ by minimizing the Kullback-Leibler divergence between the true posterior and the approximation given by

$$\text{KL}(p\|q) = \text{KL}\left(\frac{1}{p(\mathcal{D})} \prod_i f_i(\theta) \middle\| \frac{1}{Z} \prod_i \tilde{f}_i(\theta)\right). \quad (10.192)$$

Note that this is the reverse form of KL divergence compared with that used in variational inference. In general, this minimization will be intractable because the KL divergence involves averaging with respect to the true distribution. As a rough approximation, we could instead minimize the KL divergences between the corresponding pairs $f_i(\theta)$ and $\tilde{f}_i(\theta)$ of factors. This represents a much simpler problem to solve, and has the advantage that the algorithm is noniterative. However, because each factor is individually approximated, the product of the factors could well give a poor approximation.

Expectation propagation makes a much better approximation by optimizing each factor in turn in the context of all of the remaining factors. It starts by initializing the factors $\tilde{f}_i(\theta)$, and then cycles through the factors refining them one at a time. This is similar in spirit to the update of factors in the variational Bayes framework considered earlier. Suppose we wish to refine factor $\tilde{f}_j(\theta)$. We first remove this factor from the product to give $\prod_{i \neq j} \tilde{f}_i(\theta)$. Conceptually, we will now determine a revised form of the factor $\tilde{f}_j(\theta)$ by ensuring that the product

$$q^{\text{new}}(\theta) \propto \tilde{f}_j(\theta) \prod_{i \neq j} \tilde{f}_i(\theta) \quad (10.193)$$

is as close as possible to

$$f_j(\theta) \prod_{i \neq j} \tilde{f}_i(\theta) \quad (10.194)$$

in which we keep fixed all of the factors $\tilde{f}_i(\theta)$ for $i \neq j$. This ensures that the approximation is most accurate in the regions of high posterior probability as defined by the remaining factors. We shall see an example of this effect when we apply EP to the ‘clutter problem’. To achieve this, we first remove the factor $\tilde{f}_j(\theta)$ from the current approximation to the posterior by defining the unnormalized distribution

$$q^{\setminus j}(\theta) = \frac{q(\theta)}{\tilde{f}_j(\theta)}. \quad (10.195)$$

Note that we could instead find $q^{\setminus j}(\theta)$ from the product of factors $i \neq j$, although in practice division is usually easier. This is now combined with the factor $f_j(\theta)$ to give a distribution

$$\frac{1}{Z_j} f_j(\theta) q^{\setminus j}(\theta) \quad (10.196)$$

Section 10.7.1

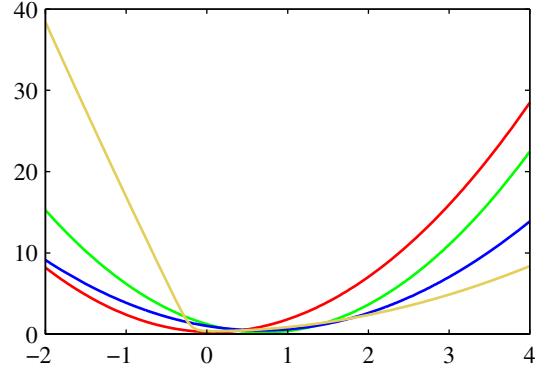
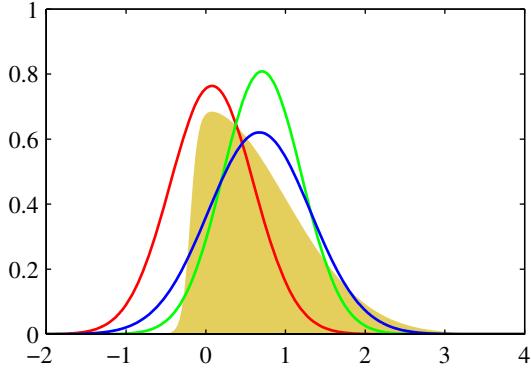


Figure 10.14 Illustration of the expectation propagation approximation using a Gaussian distribution for the example considered earlier in Figures 4.14 and 10.1. The left-hand plot shows the original distribution (yellow) along with the Laplace (red), global variational (green), and EP (blue) approximations, and the right-hand plot shows the corresponding negative logarithms of the distributions. Note that the EP distribution is broader than that variational inference, as a consequence of the different form of KL divergence.

where Z_j is the normalization constant given by

$$Z_j = \int f_j(\boldsymbol{\theta}) q^{\backslash j}(\boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (10.197)$$

We now determine a revised factor $\tilde{f}_j(\boldsymbol{\theta})$ by minimizing the Kullback-Leibler divergence

$$\text{KL} \left(\frac{f_j(\boldsymbol{\theta}) q^{\backslash j}(\boldsymbol{\theta})}{Z_j} \middle\| q^{\text{new}}(\boldsymbol{\theta}) \right). \quad (10.198)$$

This is easily solved because the approximating distribution $q^{\text{new}}(\boldsymbol{\theta})$ is from the exponential family, and so we can appeal to the result (10.187), which tells us that the parameters of $q^{\text{new}}(\boldsymbol{\theta})$ are obtained by matching its expected sufficient statistics to the corresponding moments of (10.196). We shall assume that this is a tractable operation. For example, if we choose $q(\boldsymbol{\theta})$ to be a Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then $\boldsymbol{\mu}$ is set equal to the mean of the (unnormalized) distribution $f_j(\boldsymbol{\theta}) q^{\backslash j}(\boldsymbol{\theta})$, and $\boldsymbol{\Sigma}$ is set to its covariance. More generally, it is straightforward to obtain the required expectations for any member of the exponential family, provided it can be normalized, because the expected statistics can be related to the derivatives of the normalization coefficient, as given by (2.226). The EP approximation is illustrated in Figure 10.14.

From (10.193), we see that the revised factor $\tilde{f}_j(\boldsymbol{\theta})$ can be found by taking $q^{\text{new}}(\boldsymbol{\theta})$ and dividing out the remaining factors so that

$$\tilde{f}_j(\boldsymbol{\theta}) = K \frac{q^{\text{new}}(\boldsymbol{\theta})}{q^{\backslash j}(\boldsymbol{\theta})} \quad (10.199)$$

where we have used (10.195). The coefficient K is determined by multiplying both

sides of (10.199) by $q^{\backslash i}(\boldsymbol{\theta})$ and integrating to give

$$K = \int \tilde{f}_j(\boldsymbol{\theta}) q^{\backslash j}(\boldsymbol{\theta}) d\boldsymbol{\theta} \quad (10.200)$$

where we have used the fact that $q^{\text{new}}(\boldsymbol{\theta})$ is normalized. The value of K can therefore be found by matching zeroth-order moments

$$\int \tilde{f}_j(\boldsymbol{\theta}) q^{\backslash j}(\boldsymbol{\theta}) d\boldsymbol{\theta} = \int f_j(\boldsymbol{\theta}) q^{\backslash j}(\boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (10.201)$$

Combining this with (10.197), we then see that $K = Z_j$ and so can be found by evaluating the integral in (10.197).

In practice, several passes are made through the set of factors, revising each factor in turn. The posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ is then approximated using (10.191), and the model evidence $p(\mathcal{D})$ can be approximated by using (10.190) with the factors $f_i(\boldsymbol{\theta})$ replaced by their approximations $\tilde{f}_i(\boldsymbol{\theta})$.

Expectation Propagation

We are given a joint distribution over observed data \mathcal{D} and stochastic variables $\boldsymbol{\theta}$ in the form of a product of factors

$$p(\mathcal{D}, \boldsymbol{\theta}) = \prod_i f_i(\boldsymbol{\theta}) \quad (10.202)$$

and we wish to approximate the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ by a distribution of the form

$$q(\boldsymbol{\theta}) = \frac{1}{Z} \prod_i \tilde{f}_i(\boldsymbol{\theta}). \quad (10.203)$$

We also wish to approximate the model evidence $p(\mathcal{D})$.

1. Initialize all of the approximating factors $\tilde{f}_i(\boldsymbol{\theta})$.
2. Initialize the posterior approximation by setting

$$q(\boldsymbol{\theta}) \propto \prod_i \tilde{f}_i(\boldsymbol{\theta}). \quad (10.204)$$

3. Until convergence:
 - (a) Choose a factor $\tilde{f}_j(\boldsymbol{\theta})$ to refine.
 - (b) Remove $\tilde{f}_j(\boldsymbol{\theta})$ from the posterior by division

$$q^{\backslash j}(\boldsymbol{\theta}) = \frac{q(\boldsymbol{\theta})}{\tilde{f}_j(\boldsymbol{\theta})}. \quad (10.205)$$

- (c) Evaluate the new posterior by setting the sufficient statistics (moments) of $q^{\text{new}}(\boldsymbol{\theta})$ equal to those of $q^{\backslash j}(\boldsymbol{\theta})f_j(\boldsymbol{\theta})$, including evaluation of the normalization constant

$$Z_j = \int q^{\backslash j}(\boldsymbol{\theta})f_j(\boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (10.206)$$

- (d) Evaluate and store the new factor

$$\tilde{f}_j(\boldsymbol{\theta}) = Z_j \frac{q^{\text{new}}(\boldsymbol{\theta})}{q^{\backslash j}(\boldsymbol{\theta})}. \quad (10.207)$$

4. Evaluate the approximation to the model evidence

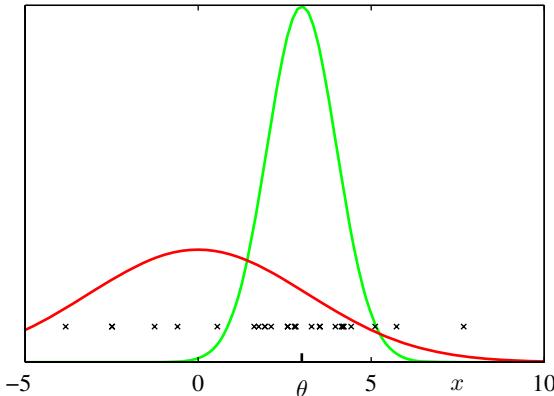
$$p(\mathcal{D}) \simeq \int \prod_i \tilde{f}_i(\boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (10.208)$$

A special case of EP, known as *assumed density filtering* (ADF) or *moment matching* (Maybeck, 1982; Lauritzen, 1992; Boyen and Koller, 1998; Opper and Winther, 1999), is obtained by initializing all of the approximating factors except the first to unity and then making one pass through the factors updating each of them once. Assumed density filtering can be appropriate for on-line learning in which data points are arriving in a sequence and we need to learn from each data point and then discard it before considering the next point. However, in a batch setting we have the opportunity to re-use the data points many times in order to achieve improved accuracy, and it is this idea that is exploited in expectation propagation. Furthermore, if we apply ADF to batch data, the results will have an undesirable dependence on the (arbitrary) order in which the data points are considered, which again EP can overcome.

One disadvantage of expectation propagation is that there is no guarantee that the iterations will converge. However, for approximations $q(\boldsymbol{\theta})$ in the exponential family, if the iterations do converge, the resulting solution will be a stationary point of a particular energy function (Minka, 2001a), although each iteration of EP does not necessarily decrease the value of this energy function. This is in contrast to variational Bayes, which iteratively maximizes a lower bound on the log marginal likelihood, in which each iteration is guaranteed not to decrease the bound. It is possible to optimize the EP cost function directly, in which case it is guaranteed to converge, although the resulting algorithms can be slower and more complex to implement.

Another difference between variational Bayes and EP arises from the form of KL divergence that is minimized by the two algorithms, because the former minimizes $\text{KL}(q\|p)$ whereas the latter minimizes $\text{KL}(p\|q)$. As we saw in Figure 10.3, for distributions $p(\boldsymbol{\theta})$ which are multimodal, minimizing $\text{KL}(p\|q)$ can lead to poor approximations. In particular, if EP is applied to mixtures the results are not sensible because the approximation tries to capture all of the modes of the posterior distribution. Conversely, in logistic-type models, EP often out-performs both local variational methods and the Laplace approximation (Kuss and Rasmussen, 2006).

Figure 10.15 Illustration of the clutter problem for a data space dimensionality of $D = 1$. Training data points, denoted by the crosses, are drawn from a mixture of two Gaussians with components shown in red and green. The goal is to infer the mean of the green Gaussian from the observed data.



10.7.1 Example: The clutter problem

Following Minka (2001b), we illustrate the EP algorithm using a simple example in which the goal is to infer the mean θ of a multivariate Gaussian distribution over a variable x given a set of observations drawn from that distribution. To make the problem more interesting, the observations are embedded in background clutter, which itself is also Gaussian distributed, as illustrated in Figure 10.15. The distribution of observed values x is therefore a mixture of Gaussians, which we take to be of the form

$$p(\mathbf{x}|\boldsymbol{\theta}) = (1-w)\mathcal{N}(\mathbf{x}|\boldsymbol{\theta}, \mathbf{I}) + w\mathcal{N}(\mathbf{x}|\mathbf{0}, a\mathbf{I}) \quad (10.209)$$

where w is the proportion of background clutter and is assumed to be known. The prior over $\boldsymbol{\theta}$ is taken to be Gaussian

$$p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, b\mathbf{I}) \quad (10.210)$$

and Minka (2001a) chooses the parameter values $a = 10$, $b = 100$ and $w = 0.5$. The joint distribution of N observations $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ and $\boldsymbol{\theta}$ is given by

$$p(\mathcal{D}, \boldsymbol{\theta}) = p(\boldsymbol{\theta}) \prod_{n=1}^N p(\mathbf{x}_n|\boldsymbol{\theta}) \quad (10.211)$$

and so the posterior distribution comprises a mixture of 2^N Gaussians. Thus the computational cost of solving this problem exactly would grow exponentially with the size of the data set, and so an exact solution is intractable for moderately large N .

To apply EP to the clutter problem, we first identify the factors $f_0(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$ and $f_n(\boldsymbol{\theta}) = p(\mathbf{x}_n|\boldsymbol{\theta})$. Next we select an approximating distribution from the exponential family, and for this example it is convenient to choose a spherical Gaussian

$$q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{m}, v\mathbf{I}). \quad (10.212)$$

The factor approximations will therefore take the form of exponential-quadratic functions of the form

$$\tilde{f}_n(\boldsymbol{\theta}) = s_n \mathcal{N}(\boldsymbol{\theta} | \mathbf{m}_n, v_n \mathbf{I}) \quad (10.213)$$

where $n = 1, \dots, N$, and we set $\tilde{f}_0(\boldsymbol{\theta})$ equal to the prior $p(\boldsymbol{\theta})$. Note that the use of $\mathcal{N}(\boldsymbol{\theta} | \cdot, \cdot)$ does not imply that the right-hand side is a well-defined Gaussian density (in fact, as we shall see, the variance parameter v_n can be negative) but is simply a convenient shorthand notation. The approximations $\tilde{f}_n(\boldsymbol{\theta})$, for $n = 1, \dots, N$, can be initialized to unity, corresponding to $s_n = (2\pi v_n)^{D/2}$, $v_n \rightarrow \infty$ and $\mathbf{m}_n = \mathbf{0}$, where D is the dimensionality of \mathbf{x} and hence of $\boldsymbol{\theta}$. The initial $q(\boldsymbol{\theta})$, defined by (10.191), is therefore equal to the prior.

We then iteratively refine the factors by taking one factor $f_n(\boldsymbol{\theta})$ at a time and applying (10.205), (10.206), and (10.207). Note that we do not need to revise the term $f_0(\boldsymbol{\theta})$ because an EP update will leave this term unchanged. Here we state the results and leave the reader to fill in the details.

First we remove the current estimate $f_n(\boldsymbol{\theta})$ from $q(\boldsymbol{\theta})$ by division using (10.205) to give $q^{\setminus n}(\boldsymbol{\theta})$, which has mean and inverse variance given by

$$\mathbf{m}^{\setminus n} = \mathbf{m} + v^{\setminus n} v_n^{-1} (\mathbf{m} - \mathbf{m}_n) \quad (10.214)$$

$$(v^{\setminus n})^{-1} = v^{-1} - v_n^{-1}. \quad (10.215)$$

Next we evaluate the normalization constant Z_n using (10.206) to give

$$Z_n = (1 - w) \mathcal{N}(\mathbf{x}_n | \mathbf{m}^{\setminus n}, (v^{\setminus n} + 1) \mathbf{I}) + w \mathcal{N}(\mathbf{x}_n | \mathbf{0}, a \mathbf{I}). \quad (10.216)$$

Similarly, we compute the mean and variance of $q^{\text{new}}(\boldsymbol{\theta})$ by finding the mean and variance of $q^{\setminus n}(\boldsymbol{\theta}) f_n(\boldsymbol{\theta})$ to give

$$\mathbf{m} = \mathbf{m}^{\setminus n} + \rho_n \frac{v^{\setminus n}}{v^{\setminus n} + 1} (\mathbf{x}_n - \mathbf{m}^{\setminus n}) \quad (10.217)$$

$$v = v^{\setminus n} - \rho_n \frac{(v^{\setminus n})^2}{v^{\setminus n} + 1} + \rho_n (1 - \rho_n) \frac{(v^{\setminus n})^2 \|\mathbf{x}_n - \mathbf{m}^{\setminus n}\|^2}{D(v^{\setminus n} + 1)^2} \quad (10.218)$$

where the quantity

$$\rho_n = 1 - \frac{w}{Z_n} \mathcal{N}(\mathbf{x}_n | \mathbf{0}, a \mathbf{I}) \quad (10.219)$$

has a simple interpretation as the probability of the point \mathbf{x}_n not being clutter. Then we use (10.207) to compute the refined factor $\tilde{f}_n(\boldsymbol{\theta})$ whose parameters are given by

$$v_n^{-1} = (v^{\text{new}})^{-1} - (v^{\setminus n})^{-1} \quad (10.220)$$

$$\mathbf{m}_n = \mathbf{m}^{\setminus n} + (v_n + v^{\setminus n})(v^{\setminus n})^{-1} (\mathbf{m}^{\text{new}} - \mathbf{m}^{\setminus n}) \quad (10.221)$$

$$s_n = \frac{Z_n}{(2\pi v_n)^{D/2} \mathcal{N}(\mathbf{m}_n | \mathbf{m}^{\setminus n}, (v_n + v^{\setminus n}) \mathbf{I})}. \quad (10.222)$$

This refinement process is repeated until a suitable termination criterion is satisfied, for instance that the maximum change in parameter values resulting from a complete

Exercise 10.37

Exercise 10.38

Exercise 10.39

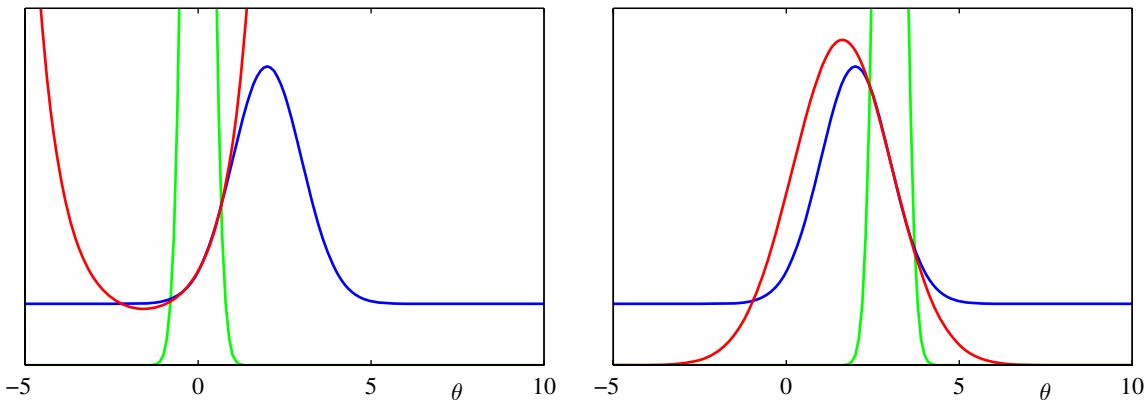


Figure 10.16 Examples of the approximation of specific factors for a one-dimensional version of the clutter problem, showing $f_n(\theta)$ in blue, $\tilde{f}_n(\theta)$ in red, and $q^{(n)}(\theta)$ in green. Notice that the current form for $q^{(n)}(\theta)$ controls the range of θ over which $\tilde{f}_n(\theta)$ will be a good approximation to $f_n(\theta)$.

pass through all factors is less than some threshold. Finally, we use (10.208) to evaluate the approximation to the model evidence, given by

$$p(\mathcal{D}) \simeq (2\pi v^{\text{new}})^{D/2} \exp(B/2) \prod_{n=1}^N \{s_n (2\pi v_n)^{-D/2}\} \quad (10.223)$$

where

$$B = \frac{(\mathbf{m}^{\text{new}})^T \mathbf{m}^{\text{new}}}{v} - \sum_{n=1}^N \frac{\mathbf{m}_n^T \mathbf{m}_n}{v_n}. \quad (10.224)$$

Examples factor approximations for the clutter problem with a one-dimensional parameter space θ are shown in Figure 10.16. Note that the factor approximations can have infinite or even negative values for the ‘variance’ parameter v_n . This simply corresponds to approximations that curve upwards instead of downwards and are not necessarily problematic provided the overall approximate posterior $q(\theta)$ has positive variance. Figure 10.17 compares the performance of EP with variational Bayes (mean field theory) and the Laplace approximation on the clutter problem.

10.7.2 Expectation propagation on graphs

So far in our general discussion of EP, we have allowed the factors $f_i(\theta)$ in the distribution $p(\theta)$ to be functions of all of the components of θ , and similarly for the approximating factors $\tilde{f}(\theta)$ in the approximating distribution $q(\theta)$. We now consider situations in which the factors depend only on subsets of the variables. Such restrictions can be conveniently expressed using the framework of probabilistic graphical models, as discussed in Chapter 8. Here we use a factor graph representation because this encompasses both directed and undirected graphs.

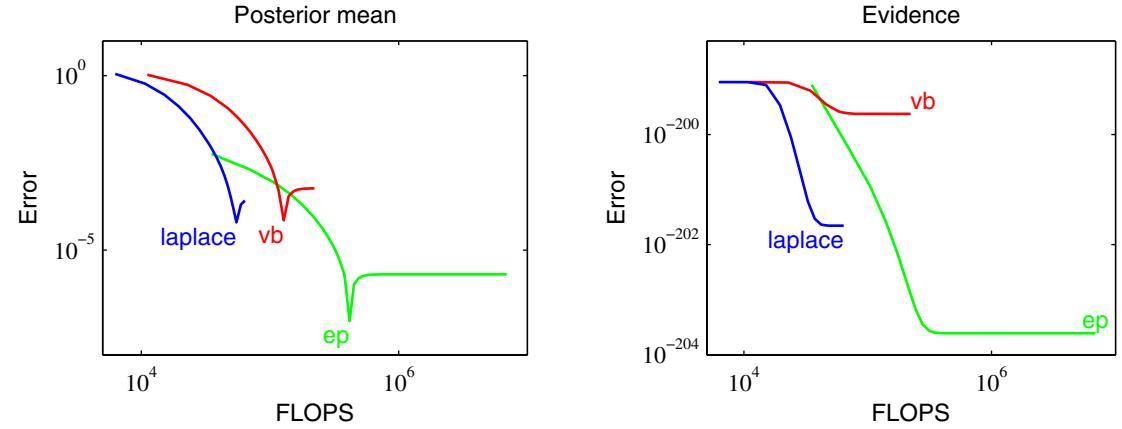


Figure 10.17 Comparison of expectation propagation, variational inference, and the Laplace approximation on the clutter problem. The left-hand plot shows the error in the predicted posterior mean versus the number of floating point operations, and the right-hand plot shows the corresponding results for the model evidence.

We shall focus on the case in which the approximating distribution is fully factorized, and we shall show that in this case expectation propagation reduces to loopy belief propagation (Minka, 2001a). To start with, we show this in the context of a simple example, and then we shall explore the general case.

First of all, recall from (10.17) that if we minimize the Kullback-Leibler divergence $\text{KL}(p\|q)$ with respect to a factorized distribution q , then the optimal solution for each factor is simply the corresponding marginal of p .

Now consider the factor graph shown on the left in Figure 10.18, which was introduced earlier in the context of the sum-product algorithm. The joint distribution is given by

$$p(\mathbf{x}) = f_a(x_1, x_2)f_b(x_2, x_3)f_c(x_2, x_4). \quad (10.225)$$

We seek an approximation $q(\mathbf{x})$ that has the same factorization, so that

$$q(\mathbf{x}) \propto \tilde{f}_a(x_1, x_2)\tilde{f}_b(x_2, x_3)\tilde{f}_c(x_2, x_4). \quad (10.226)$$

Note that normalization constants have been omitted, and these can be re-instated at the end by local normalization, as is generally done in belief propagation. Now suppose we restrict attention to approximations in which the factors themselves factorize with respect to the individual variables so that

$$q(\mathbf{x}) \propto \tilde{f}_{a1}(x_1)\tilde{f}_{a2}(x_2)\tilde{f}_{b2}(x_2)\tilde{f}_{b3}(x_3)\tilde{f}_{c2}(x_2)\tilde{f}_{c4}(x_4) \quad (10.227)$$

which corresponds to the factor graph shown on the right in Figure 10.18. Because the individual factors are factorized, the overall distribution $q(\mathbf{x})$ is itself fully factorized.

Now we apply the EP algorithm using the fully factorized approximation. Suppose that we have initialized all of the factors and that we choose to refine factor

Section 8.4.4

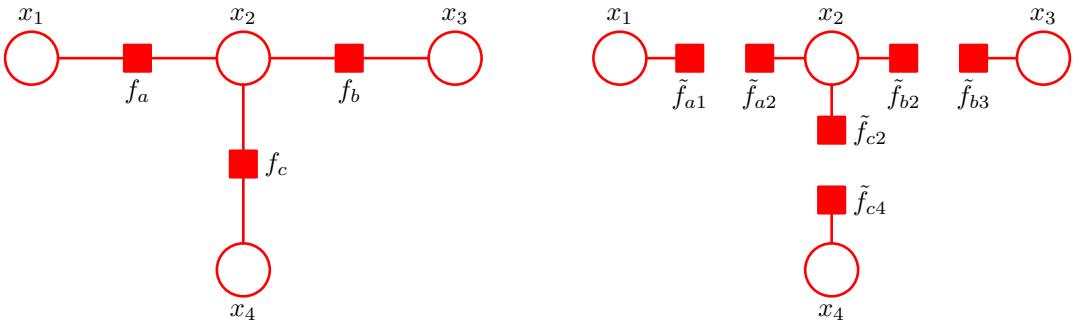


Figure 10.18 On the left is a simple factor graph from Figure 8.51 and reproduced here for convenience. On the right is the corresponding factorized approximation.

$\tilde{f}_b(x_2, x_3) = \tilde{f}_{b2}(x_2)\tilde{f}_{b3}(x_3)$. We first remove this factor from the approximating distribution to give

$$q^{\backslash b}(\mathbf{x}) = \tilde{f}_{a1}(x_1)\tilde{f}_{a2}(x_2)\tilde{f}_{c2}(x_2)\tilde{f}_{c4}(x_4) \quad (10.228)$$

and we then multiply this by the exact factor $f_b(x_2, x_3)$ to give

$$\hat{p}(\mathbf{x}) = q^{\backslash b}(\mathbf{x})f_b(x_2, x_3) = \tilde{f}_{a1}(x_1)\tilde{f}_{a2}(x_2)\tilde{f}_{c2}(x_2)\tilde{f}_{c4}(x_4)f_b(x_2, x_3). \quad (10.229)$$

We now find $q^{\text{new}}(\mathbf{x})$ by minimizing the Kullback-Leibler divergence $\text{KL}(\hat{p} \| q^{\text{new}})$. The result, as noted above, is that $q^{\text{new}}(\mathbf{z})$ comprises the product of factors, one for each variable x_i , in which each factor is given by the corresponding marginal of $\hat{p}(\mathbf{x})$. These four marginals are given by

$$\hat{p}(x_1) \propto \tilde{f}_{a1}(x_1) \quad (10.230)$$

$$\hat{p}(x_2) \propto \tilde{f}_{a2}(x_2)\tilde{f}_{c2}(x_2) \sum_{x_3} f_b(x_2, x_3) \quad (10.231)$$

$$\hat{p}(x_3) \propto \sum_{x_2} \left\{ f_b(x_2, x_3)\tilde{f}_{a2}(x_2)\tilde{f}_{c2}(x_2) \right\} \quad (10.232)$$

$$\hat{p}(x_4) \propto \tilde{f}_{c4}(x_4) \quad (10.233)$$

and $q^{\text{new}}(\mathbf{x})$ is obtained by multiplying these marginals together. We see that the only factors in $q(\mathbf{x})$ that change when we update $\tilde{f}_b(x_2, x_3)$ are those that involve the variables in f_b namely x_2 and x_3 . To obtain the refined factor $\tilde{f}_b(x_2, x_3) = \tilde{f}_{b2}(x_2)\tilde{f}_{b3}(x_3)$ we simply divide $q^{\text{new}}(\mathbf{x})$ by $q^{\backslash b}(\mathbf{x})$, which gives

$$\tilde{f}_{b2}(x_2) \propto \sum_{x_3} f_b(x_2, x_3) \quad (10.234)$$

$$\tilde{f}_{b3}(x_3) \propto \sum_{x_2} \left\{ f_b(x_2, x_3)\tilde{f}_{a2}(x_2)\tilde{f}_{c2}(x_2) \right\}. \quad (10.235)$$

Section 8.4.4

These are precisely the messages obtained using belief propagation in which messages from variable nodes to factor nodes have been folded into the messages from factor nodes to variable nodes. In particular, $\tilde{f}_{b2}(x_2)$ corresponds to the message $\mu_{f_b \rightarrow x_2}(x_2)$ sent by factor node f_b to variable node x_2 and is given by (8.81). Similarly, if we substitute (8.78) into (8.79), we obtain (10.235) in which $\tilde{f}_{a2}(x_2)$ corresponds to $\mu_{f_a \rightarrow x_2}(x_2)$ and $\tilde{f}_{c2}(x_2)$ corresponds to $\mu_{f_c \rightarrow x_2}(x_2)$, giving the message $\tilde{f}_{b3}(x_3)$ which corresponds to $\mu_{f_b \rightarrow x_3}(x_3)$.

This result differs slightly from standard belief propagation in that messages are passed in both directions at the same time. We can easily modify the EP procedure to give the standard form of the sum-product algorithm by updating just one of the factors at a time, for instance if we refine only $\tilde{f}_{b3}(x_3)$, then $\tilde{f}_{b2}(x_2)$ is unchanged by definition, while the refined version of $\tilde{f}_{b3}(x_3)$ is again given by (10.235). If we are refining only one term at a time, then we can choose the order in which the refinements are done as we wish. In particular, for a tree-structured graph we can follow a two-pass update scheme, corresponding to the standard belief propagation schedule, which will result in exact inference of the variable and factor marginals. The initialization of the approximation factors in this case is unimportant.

Now let us consider a general factor graph corresponding to the distribution

$$p(\boldsymbol{\theta}) = \prod_i f_i(\boldsymbol{\theta}_i) \quad (10.236)$$

where $\boldsymbol{\theta}_i$ represents the subset of variables associated with factor f_i . We approximate this using a fully factorized distribution of the form

$$q(\boldsymbol{\theta}) \propto \prod_i \prod_k \tilde{f}_{ik}(\boldsymbol{\theta}_k) \quad (10.237)$$

where $\boldsymbol{\theta}_k$ corresponds to an individual variable node. Suppose that we wish to refine the particular term $\tilde{f}_{jl}(\boldsymbol{\theta}_l)$ keeping all other terms fixed. We first remove the term $\tilde{f}_j(\boldsymbol{\theta}_j)$ from $q(\boldsymbol{\theta})$ to give

$$q^{\setminus j}(\boldsymbol{\theta}) \propto \prod_{i \neq j} \prod_k \tilde{f}_{ik}(\boldsymbol{\theta}_k) \quad (10.238)$$

and then multiply by the exact factor $f_j(\boldsymbol{\theta}_j)$. To determine the refined term $\tilde{f}_{jl}(\boldsymbol{\theta}_l)$, we need only consider the functional dependence on $\boldsymbol{\theta}_l$, and so we simply find the corresponding marginal of

$$q^{\setminus j}(\boldsymbol{\theta}) f_j(\boldsymbol{\theta}_j). \quad (10.239)$$

Up to a multiplicative constant, this involves taking the marginal of $f_j(\boldsymbol{\theta}_j)$ multiplied by any terms from $q^{\setminus j}(\boldsymbol{\theta})$ that are functions of any of the variables in $\boldsymbol{\theta}_j$. Terms that correspond to other factors $\tilde{f}_i(\boldsymbol{\theta}_i)$ for $i \neq j$ will cancel between numerator and denominator when we subsequently divide by $q^{\setminus j}(\boldsymbol{\theta})$. We therefore obtain

$$\tilde{f}_{jl}(\boldsymbol{\theta}_l) \propto \sum_{\boldsymbol{\theta}_{m \neq l} \in \boldsymbol{\theta}_j} f_j(\boldsymbol{\theta}_j) \prod_k \prod_{m \neq l} \tilde{f}_{km}(\boldsymbol{\theta}_m). \quad (10.240)$$

We recognize this as the sum-product rule in the form in which messages from variable nodes to factor nodes have been eliminated, as illustrated by the example shown in Figure 8.50. The quantity $\tilde{f}_{jm}(\theta_m)$ corresponds to the message $\mu_{f_j \rightarrow \theta_m}(\theta_m)$, which factor node j sends to variable node m , and the product over k in (10.240) is over all factors that depend on the variables θ_m that have variables (other than variable θ_l) in common with factor $f_j(\theta_j)$. In other words, to compute the outgoing message from a factor node, we take the product of all the incoming messages from other factor nodes, multiply by the local factor, and then marginalize.

Thus, the sum-product algorithm arises as a special case of expectation propagation if we use an approximating distribution that is fully factorized. This suggests that more flexible approximating distributions, corresponding to partially disconnected graphs, could be used to achieve higher accuracy. Another generalization is to group factors $f_i(\theta_i)$ together into sets and to refine all the factors in a set together at each iteration. Both of these approaches can lead to improvements in accuracy (Minka, 2001b). In general, the problem of choosing the best combination of grouping and disconnection is an open research issue.

We have seen that variational message passing and expectation propagation optimize two different forms of the Kullback-Leibler divergence. Minka (2005) has shown that a broad range of message passing algorithms can be derived from a common framework involving minimization of members of the alpha family of divergences, given by (10.19). These include variational message passing, loopy belief propagation, and expectation propagation, as well as a range of other algorithms, which we do not have space to discuss here, such as *tree-reweighted message passing* (Wainwright *et al.*, 2005), *fractional belief propagation* (Wiegerinck and Heskes, 2003), and *power EP* (Minka, 2004).

Exercises

- 10.1** (★) **www** Verify that the log marginal distribution of the observed data $\ln p(\mathbf{X})$ can be decomposed into two terms in the form (10.2) where $\mathcal{L}(q)$ is given by (10.3) and $\text{KL}(q\|p)$ is given by (10.4).
- 10.2** (★) Use the properties $\mathbb{E}[z_1] = m_1$ and $\mathbb{E}[z_2] = m_2$ to solve the simultaneous equations (10.13) and (10.15), and hence show that, provided the original distribution $p(\mathbf{z})$ is nonsingular, the unique solution for the means of the factors in the approximation distribution is given by $\mathbb{E}[z_1] = \mu_1$ and $\mathbb{E}[z_2] = \mu_2$.
- 10.3** (★★) **www** Consider a factorized variational distribution $q(\mathbf{Z})$ of the form (10.5). By using the technique of Lagrange multipliers, verify that minimization of the Kullback-Leibler divergence $\text{KL}(p\|q)$ with respect to one of the factors $q_i(\mathbf{Z}_i)$, keeping all other factors fixed, leads to the solution (10.17).
- 10.4** (★★) Suppose that $p(\mathbf{x})$ is some fixed distribution and that we wish to approximate it using a Gaussian distribution $q(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. By writing down the form of the KL divergence $\text{KL}(p\|q)$ for a Gaussian $q(\mathbf{x})$ and then differentiating, show that

minimization of $\text{KL}(p\|q)$ with respect to μ and Σ leads to the result that μ is given by the expectation of \mathbf{x} under $p(\mathbf{x})$ and that Σ is given by the covariance.

- 10.5** (**) **www** Consider a model in which the set of all hidden stochastic variables, denoted collectively by \mathbf{Z} , comprises some latent variables \mathbf{z} together with some model parameters $\boldsymbol{\theta}$. Suppose we use a variational distribution that factorizes between latent variables and parameters so that $q(\mathbf{z}, \boldsymbol{\theta}) = q_{\mathbf{z}}(\mathbf{z})q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$, in which the distribution $q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ is approximated by a point estimate of the form $q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$ where $\boldsymbol{\theta}_0$ is a vector of free parameters. Show that variational optimization of this factorized distribution is equivalent to an EM algorithm, in which the E step optimizes $q_{\mathbf{z}}(\mathbf{z})$, and the M step maximizes the expected complete-data log posterior distribution of $\boldsymbol{\theta}$ with respect to $\boldsymbol{\theta}_0$.
- 10.6** (**) The alpha family of divergences is defined by (10.19). Show that the Kullback-Leibler divergence $\text{KL}(p\|q)$ corresponds to $\alpha \rightarrow 1$. This can be done by writing $p^\epsilon = \exp(\epsilon \ln p) = 1 + \epsilon \ln p + O(\epsilon^2)$ and then taking $\epsilon \rightarrow 0$. Similarly show that $\text{KL}(q\|p)$ corresponds to $\alpha \rightarrow -1$.
- 10.7** (**) Consider the problem of inferring the mean and precision of a univariate Gaussian using a factorized variational approximation, as considered in Section 10.1.3. Show that the factor $q_\mu(\mu)$ is a Gaussian of the form $\mathcal{N}(\mu|\mu_N, \lambda_N^{-1})$ with mean and precision given by (10.26) and (10.27), respectively. Similarly show that the factor $q_\tau(\tau)$ is a gamma distribution of the form $\text{Gam}(\tau|a_N, b_N)$ with parameters given by (10.29) and (10.30).
- 10.8** (*) Consider the variational posterior distribution for the precision of a univariate Gaussian whose parameters are given by (10.29) and (10.30). By using the standard results for the mean and variance of the gamma distribution given by (B.27) and (B.28), show that if we let $N \rightarrow \infty$, this variational posterior distribution has a mean given by the inverse of the maximum likelihood estimator for the variance of the data, and a variance that goes to zero.
- 10.9** (**) By making use of the standard result $\mathbb{E}[\tau] = a_N/b_N$ for the mean of a gamma distribution, together with (10.26), (10.27), (10.29), and (10.30), derive the result (10.33) for the reciprocal of the expected precision in the factorized variational treatment of a univariate Gaussian.
- 10.10** (*) **www** Derive the decomposition given by (10.34) that is used to find approximate posterior distributions over models using variational inference.
- 10.11** (**) **www** By using a Lagrange multiplier to enforce the normalization constraint on the distribution $q(m)$, show that the maximum of the lower bound (10.35) is given by (10.36).
- 10.12** (**) Starting from the joint distribution (10.41), and applying the general result (10.9), show that the optimal variational distribution $q^*(\mathbf{Z})$ over the latent variables for the Bayesian mixture of Gaussians is given by (10.48) by verifying the steps given in the text.

- 10.13** (**) **www** Starting from (10.54), derive the result (10.59) for the optimum variational posterior distribution over μ_k and Λ_k in the Bayesian mixture of Gaussians, and hence verify the expressions for the parameters of this distribution given by (10.60)–(10.63).
- 10.14** (**) Using the distribution (10.59), verify the result (10.64).
- 10.15** (*) Using the result (B.17), show that the expected value of the mixing coefficients in the variational mixture of Gaussians is given by (10.69).
- 10.16** (**) **www** Verify the results (10.71) and (10.72) for the first two terms in the lower bound for the variational Gaussian mixture model given by (10.70).
- 10.17** (***) Verify the results (10.73)–(10.77) for the remaining terms in the lower bound for the variational Gaussian mixture model given by (10.70).
- 10.18** (****) In this exercise, we shall derive the variational re-estimation equations for the Gaussian mixture model by direct differentiation of the lower bound. To do this we assume that the variational distribution has the factorization defined by (10.42) and (10.55) with factors given by (10.48), (10.57), and (10.59). Substitute these into (10.70) and hence obtain the lower bound as a function of the parameters of the variational distribution. Then, by maximizing the bound with respect to these parameters, derive the re-estimation equations for the factors in the variational distribution, and show that these are the same as those obtained in Section 10.2.1.
- 10.19** (**) Derive the result (10.81) for the predictive distribution in the variational treatment of the Bayesian mixture of Gaussians model.
- 10.20** (**) **www** This exercise explores the variational Bayes solution for the mixture of Gaussians model when the size N of the data set is large and shows that it reduces (as we would expect) to the maximum likelihood solution based on EM derived in Chapter 9. Note that results from Appendix B may be used to help answer this exercise. First show that the posterior distribution $q^*(\Lambda_k)$ of the precisions becomes sharply peaked around the maximum likelihood solution. Do the same for the posterior distribution of the means $q^*(\mu_k | \Lambda_k)$. Next consider the posterior distribution $q^*(\pi)$ for the mixing coefficients and show that this too becomes sharply peaked around the maximum likelihood solution. Similarly, show that the responsibilities become equal to the corresponding maximum likelihood values for large N , by making use of the following asymptotic result for the digamma function for large x

$$\psi(x) = \ln x + O(1/x). \quad (10.241)$$

Finally, by making use of (10.80), show that for large N , the predictive distribution becomes a mixture of Gaussians.

- 10.21** (*) Show that the number of equivalent parameter settings due to interchange symmetries in a mixture model with K components is $K!$.

- 10.22** (**) We have seen that each mode of the posterior distribution in a Gaussian mixture model is a member of a family of $K!$ equivalent modes. Suppose that the result of running the variational inference algorithm is an approximate posterior distribution q that is localized in the neighbourhood of one of the modes. We can then approximate the full posterior distribution as a mixture of $K!$ such q distributions, once centred on each mode and having equal mixing coefficients. Show that if we assume negligible overlap between the components of the q mixture, the resulting lower bound differs from that for a single component q distribution through the addition of an extra term $\ln K!$.
- 10.23** (**) **www** Consider a variational Gaussian mixture model in which there is no prior distribution over mixing coefficients $\{\pi_k\}$. Instead, the mixing coefficients are treated as parameters, whose values are to be found by maximizing the variational lower bound on the log marginal likelihood. Show that maximizing this lower bound with respect to the mixing coefficients, using a Lagrange multiplier to enforce the constraint that the mixing coefficients sum to one, leads to the re-estimation result (10.83). Note that there is no need to consider all of the terms in the lower bound but only the dependence of the bound on the $\{\pi_k\}$.
- 10.24** (**) **www** We have seen in Section 10.2 that the singularities arising in the maximum likelihood treatment of Gaussian mixture models do not arise in a Bayesian treatment. Discuss whether such singularities would arise if the Bayesian model were solved using maximum posterior (MAP) estimation.
- 10.25** (**) The variational treatment of the Bayesian mixture of Gaussians, discussed in Section 10.2, made use of a factorized approximation (10.5) to the posterior distribution. As we saw in Figure 10.2, the factorized assumption causes the variance of the posterior distribution to be under-estimated for certain directions in parameter space. Discuss qualitatively the effect this will have on the variational approximation to the model evidence, and how this effect will vary with the number of components in the mixture. Hence explain whether the variational Gaussian mixture will tend to under-estimate or over-estimate the optimal number of components.
- 10.26** (***) Extend the variational treatment of Bayesian linear regression to include a gamma hyperprior $\text{Gam}(\beta|c_0, d_0)$ over β and solve variationally, by assuming a factorized variational distribution of the form $q(\mathbf{w})q(\alpha)q(\beta)$. Derive the variational update equations for the three factors in the variational distribution and also obtain an expression for the lower bound and for the predictive distribution.
- 10.27** (**) By making use of the formulae given in Appendix B show that the variational lower bound for the linear basis function regression model, defined by (10.107), can be written in the form (10.107) with the various terms defined by (10.108)–(10.112).
- 10.28** (***) Rewrite the model for the Bayesian mixture of Gaussians, introduced in Section 10.2, as a conjugate model from the exponential family, as discussed in Section 10.4. Hence use the general results (10.115) and (10.119) to derive the specific results (10.48), (10.57), and (10.59).

- 10.29** (*) **www** Show that the function $f(x) = \ln(x)$ is concave for $0 < x < \infty$ by computing its second derivative. Determine the form of the dual function $g(\lambda)$ defined by (10.133), and verify that minimization of $\lambda x - g(\lambda)$ with respect to λ according to (10.132) indeed recovers the function $\ln(x)$.
- 10.30** (*) By evaluating the second derivative, show that the log logistic function $f(x) = -\ln(1 + e^{-x})$ is concave. Derive the variational upper bound (10.137) directly by making a second order Taylor expansion of the log logistic function around a point $x = \xi$.
- 10.31** (**) By finding the second derivative with respect to x , show that the function $f(x) = -\ln(e^{x/2} + e^{-x/2})$ is a concave function of x . Now consider the second derivatives with respect to the variable x^2 and hence show that it is a convex function of x^2 . Plot graphs of $f(x)$ against x and against x^2 . Derive the lower bound (10.144) on the logistic sigmoid function directly by making a first order Taylor series expansion of the function $f(x)$ in the variable x^2 centred on the value ξ^2 .
- 10.32** (**) **www** Consider the variational treatment of logistic regression with sequential learning in which data points are arriving one at a time and each must be processed and discarded before the next data point arrives. Show that a Gaussian approximation to the posterior distribution can be maintained through the use of the lower bound (10.151), in which the distribution is initialized using the prior, and as each data point is absorbed its corresponding variational parameter ξ_n is optimized.
- 10.33** (*) By differentiating the quantity $Q(\xi, \xi^{\text{old}})$ defined by (10.161) with respect to the variational parameter ξ_n show that the update equation for ξ_n for the Bayesian logistic regression model is given by (10.163).
- 10.34** (**) In this exercise we derive re-estimation equations for the variational parameters ξ in the Bayesian logistic regression model of Section 4.5 by direct maximization of the lower bound given by (10.164). To do this set the derivative of $\mathcal{L}(\xi)$ with respect to ξ_n equal to zero, making use of the result (3.117) for the derivative of the log of a determinant, together with the expressions (10.157) and (10.158) which define the mean and covariance of the variational posterior distribution $q(\mathbf{w})$.
- 10.35** (**) Derive the result (10.164) for the lower bound $\mathcal{L}(\xi)$ in the variational logistic regression model. This is most easily done by substituting the expressions for the Gaussian prior $q(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{m}_0, \mathbf{S}_0)$, together with the lower bound $h(\mathbf{w}, \xi)$ on the likelihood function, into the integral (10.159) which defines $\mathcal{L}(\xi)$. Next gather together the terms which depend on \mathbf{w} in the exponential and complete the square to give a Gaussian integral, which can then be evaluated by invoking the standard result for the normalization coefficient of a multivariate Gaussian. Finally take the logarithm to obtain (10.164).
- 10.36** (**) Consider the ADF approximation scheme discussed in Section 10.7, and show that inclusion of the factor $f_j(\theta)$ leads to an update of the model evidence of the form

$$p_j(\mathcal{D}) \simeq p_{j-1}(\mathcal{D}) Z_j \quad (10.242)$$

where Z_j is the normalization constant defined by (10.197). By applying this result recursively, and initializing with $p_0(\mathcal{D}) = 1$, derive the result

$$p(\mathcal{D}) \simeq \prod_j Z_j. \quad (10.243)$$

10.37 (*) **www** Consider the expectation propagation algorithm from Section 10.7, and suppose that one of the factors $f_0(\boldsymbol{\theta})$ in the definition (10.188) has the same exponential family functional form as the approximating distribution $q(\boldsymbol{\theta})$. Show that if the factor $\tilde{f}_0(\boldsymbol{\theta})$ is initialized to be $f_0(\boldsymbol{\theta})$, then an EP update to refine $\tilde{f}_0(\boldsymbol{\theta})$ leaves $\tilde{f}_0(\boldsymbol{\theta})$ unchanged. This situation typically arises when one of the factors is the prior $p(\boldsymbol{\theta})$, and so we see that the prior factor can be incorporated once exactly and does not need to be refined.

10.38 (***) In this exercise and the next, we shall verify the results (10.214)–(10.224) for the expectation propagation algorithm applied to the clutter problem. Begin by using the division formula (10.205) to derive the expressions (10.214) and (10.215) by completing the square inside the exponential to identify the mean and variance. Also, show that the normalization constant Z_n , defined by (10.206), is given for the clutter problem by (10.216). This can be done by making use of the general result (2.115).

10.39 (***) Show that the mean and variance of $q^{\text{new}}(\boldsymbol{\theta})$ for EP applied to the clutter problem are given by (10.217) and (10.218). To do this, first prove the following results for the expectations of $\boldsymbol{\theta}$ and $\boldsymbol{\theta}\boldsymbol{\theta}^T$ under $q^{\text{new}}(\boldsymbol{\theta})$

$$\mathbb{E}[\boldsymbol{\theta}] = \mathbf{m}^{\backslash n} + v^{\backslash n} \nabla_{\mathbf{m}^{\backslash n}} \ln Z_n \quad (10.244)$$

$$\mathbb{E}[\boldsymbol{\theta}^T \boldsymbol{\theta}] = 2(v^{\backslash n})^2 \nabla_{v^{\backslash n}} \ln Z_n + 2\mathbb{E}[\boldsymbol{\theta}]^T \mathbf{m}^{\backslash n} - \|\mathbf{m}^{\backslash n}\|^2 \quad (10.245)$$

and then make use of the result (10.216) for Z_n . Next, prove the results (10.220)–(10.222) by using (10.207) and completing the square in the exponential. Finally, use (10.208) to derive the result (10.223).

11

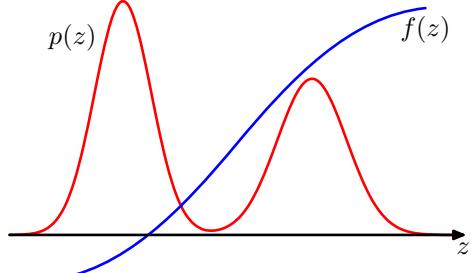
Sampling Methods



For most probabilistic models of practical interest, exact inference is intractable, and so we have to resort to some form of approximation. In Chapter 10, we discussed inference algorithms based on deterministic approximations, which include methods such as variational Bayes and expectation propagation. Here we consider approximate inference methods based on numerical sampling, also known as *Monte Carlo* techniques.

Although for some applications the posterior distribution over unobserved variables will be of direct interest in itself, for most situations the posterior distribution is required primarily for the purpose of evaluating expectations, for example in order to make predictions. The fundamental problem that we therefore wish to address in this chapter involves finding the expectation of some function $f(\mathbf{z})$ with respect to a probability distribution $p(\mathbf{z})$. Here, the components of \mathbf{z} might comprise discrete or continuous variables or some combination of the two. Thus in the case of continuous

Figure 11.1 Schematic illustration of a function $f(z)$ whose expectation is to be evaluated with respect to a distribution $p(z)$.



variables, we wish to evaluate the expectation

$$\mathbb{E}[f] = \int f(\mathbf{z})p(\mathbf{z}) d\mathbf{z} \quad (11.1)$$

where the integral is replaced by summation in the case of discrete variables. This is illustrated schematically for a single continuous variable in Figure 11.1. We shall suppose that such expectations are too complex to be evaluated exactly using analytical techniques.

The general idea behind sampling methods is to obtain a set of samples $\mathbf{z}^{(l)}$ (where $l = 1, \dots, L$) drawn independently from the distribution $p(\mathbf{z})$. This allows the expectation (11.1) to be approximated by a finite sum

$$\hat{f} = \frac{1}{L} \sum_{l=1}^L f(\mathbf{z}^{(l)}). \quad (11.2)$$

As long as the samples $\mathbf{z}^{(l)}$ are drawn from the distribution $p(\mathbf{z})$, then $\mathbb{E}[\hat{f}] = \mathbb{E}[f]$ and so the estimator \hat{f} has the correct mean. The variance of the estimator is given by

$$\text{var}[\hat{f}] = \frac{1}{L} \mathbb{E} [(f - \mathbb{E}[f])^2] \quad (11.3)$$

is the variance of the function $f(\mathbf{z})$ under the distribution $p(\mathbf{z})$. It is worth emphasizing that the accuracy of the estimator therefore does not depend on the dimensionality of \mathbf{z} , and that, in principle, high accuracy may be achievable with a relatively small number of samples $\mathbf{z}^{(l)}$. In practice, ten or twenty independent samples may suffice to estimate an expectation to sufficient accuracy.

The problem, however, is that the samples $\{\mathbf{z}^{(l)}\}$ might not be independent, and so the effective sample size might be much smaller than the apparent sample size. Also, referring back to Figure 11.1, we note that if $f(\mathbf{z})$ is small in regions where $p(\mathbf{z})$ is large, and vice versa, then the expectation may be dominated by regions of small probability, implying that relatively large sample sizes will be required to achieve sufficient accuracy.

For many models, the joint distribution $p(\mathbf{z})$ is conveniently specified in terms of a graphical model. In the case of a directed graph with no observed variables, it is

Exercise 11.1

straightforward to sample from the joint distribution (assuming that it is possible to sample from the conditional distributions at each node) using the following *ancestral sampling* approach, discussed briefly in Section 8.1.2. The joint distribution is specified by

$$p(\mathbf{z}) = \prod_{i=1}^M p(\mathbf{z}_i | \text{pa}_i) \quad (11.4)$$

where \mathbf{z}_i are the set of variables associated with node i , and pa_i denotes the set of variables associated with the parents of node i . To obtain a sample from the joint distribution, we make one pass through the set of variables in the order $\mathbf{z}_1, \dots, \mathbf{z}_M$ sampling from the conditional distributions $p(\mathbf{z}_i | \text{pa}_i)$. This is always possible because at each step all of the parent values will have been instantiated. After one pass through the graph, we will have obtained a sample from the joint distribution.

Now consider the case of a directed graph in which some of the nodes are instantiated with observed values. We can in principle extend the above procedure, at least in the case of nodes representing discrete variables, to give the following *logic sampling* approach (Henrion, 1988), which can be seen as a special case of *importance sampling* discussed in Section 11.1.4. At each step, when a sampled value is obtained for a variable \mathbf{z}_i whose value is observed, the sampled value is compared to the observed value, and if they agree then the sample value is retained and the algorithm proceeds to the next variable in turn. However, if the sampled value and the observed value disagree, then the whole sample so far is discarded and the algorithm starts again with the first node in the graph. This algorithm samples correctly from the posterior distribution because it corresponds simply to drawing samples from the joint distribution of hidden variables and data variables and then discarding those samples that disagree with the observed data (with the slight saving of not continuing with the sampling from the joint distribution as soon as one contradictory value is observed). However, the overall probability of accepting a sample from the posterior decreases rapidly as the number of observed variables increases and as the number of states that those variables can take increases, and so this approach is rarely used in practice.

In the case of probability distributions defined by an undirected graph, there is no one-pass sampling strategy that will sample even from the prior distribution with no observed variables. Instead, computationally more expensive techniques must be employed, such as Gibbs sampling, which is discussed in Section 11.3.

As well as sampling from conditional distributions, we may also require samples from a marginal distribution. If we already have a strategy for sampling from a joint distribution $p(\mathbf{u}, \mathbf{v})$, then it is straightforward to obtain samples from the marginal distribution $p(\mathbf{u})$ simply by ignoring the values for \mathbf{v} in each sample.

There are numerous texts dealing with Monte Carlo methods. Those of particular interest from the statistical inference perspective include Chen *et al.* (2001), Gamerman (1997), Gilks *et al.* (1996), Liu (2001), Neal (1996), and Robert and Casella (1999). Also there are review articles by Besag *et al.* (1995), Brooks (1998), Diaconis and Saloff-Coste (1998), Jerrum and Sinclair (1996), Neal (1993), Tierney (1994), and Andrieu *et al.* (2003) that provide additional information on sampling

methods for statistical inference.

Diagnostic tests for convergence of Markov chain Monte Carlo algorithms are summarized in Robert and Casella (1999), and some practical guidance on the use of sampling methods in the context of machine learning is given in Bishop and Nabney (2008).

11.1. Basic Sampling Algorithms

In this section, we consider some simple strategies for generating random samples from a given distribution. Because the samples will be generated by a computer algorithm they will in fact be *pseudo-random* numbers, that is, they will be deterministically calculated, but must nevertheless pass appropriate tests for randomness. Generating such numbers raises several subtleties (Press *et al.*, 1992) that lie outside the scope of this book. Here we shall assume that an algorithm has been provided that generates pseudo-random numbers distributed uniformly over $(0, 1)$, and indeed most software environments have such a facility built in.

11.1.1 Standard distributions

We first consider how to generate random numbers from simple nonuniform distributions, assuming that we already have available a source of uniformly distributed random numbers. Suppose that z is uniformly distributed over the interval $(0, 1)$, and that we transform the values of z using some function $f(\cdot)$ so that $y = f(z)$. The distribution of y will be governed by

$$p(y) = p(z) \left| \frac{dz}{dy} \right| \quad (11.5)$$

where, in this case, $p(z) = 1$. Our goal is to choose the function $f(z)$ such that the resulting values of y have some specific desired distribution $p(y)$. Integrating (11.5) we obtain

$$z = h(y) \equiv \int_{-\infty}^y p(\hat{y}) d\hat{y} \quad (11.6)$$

which is the indefinite integral of $p(y)$. Thus, $y = h^{-1}(z)$, and so we have to transform the uniformly distributed random numbers using a function which is the inverse of the indefinite integral of the desired distribution. This is illustrated in Figure 11.2.

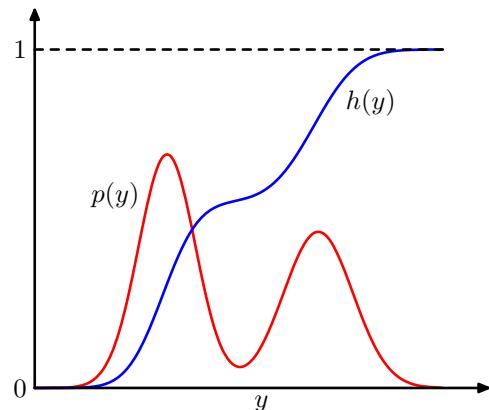
Consider for example the *exponential distribution*

$$p(y) = \lambda \exp(-\lambda y) \quad (11.7)$$

where $0 \leq y < \infty$. In this case the lower limit of the integral in (11.6) is 0, and so $h(y) = 1 - \exp(-\lambda y)$. Thus, if we transform our uniformly distributed variable z using $y = -\lambda^{-1} \ln(1 - z)$, then y will have an exponential distribution.

Exercise 11.2

Figure 11.2 Geometrical interpretation of the transformation method for generating nonuniformly distributed random numbers. $p(y)$ is the indefinite integral of the desired distribution $p(y)$. If a uniformly distributed random variable z is transformed using $y = h^{-1}(z)$, then y will be distributed according to $p(y)$.



Another example of a distribution to which the transformation method can be applied is given by the Cauchy distribution

$$p(y) = \frac{1}{\pi} \frac{1}{1 + y^2}. \quad (11.8)$$

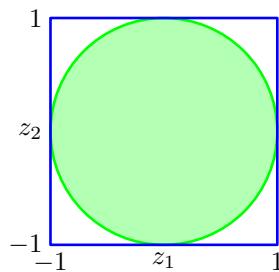
In this case, the inverse of the indefinite integral can be expressed in terms of the ‘tan’ function.

The generalization to multiple variables is straightforward and involves the Jacobian of the change of variables, so that

$$p(y_1, \dots, y_M) = p(z_1, \dots, z_M) \left| \frac{\partial(z_1, \dots, z_M)}{\partial(y_1, \dots, y_M)} \right|. \quad (11.9)$$

As a final example of the transformation method we consider the Box-Muller method for generating samples from a Gaussian distribution. First, suppose we generate pairs of uniformly distributed random numbers $z_1, z_2 \in (-1, 1)$, which we can do by transforming a variable distributed uniformly over $(0, 1)$ using $z \rightarrow 2z - 1$. Next we discard each pair unless it satisfies $z_1^2 + z_2^2 \leq 1$. This leads to a uniform distribution of points inside the unit circle with $p(z_1, z_2) = 1/\pi$, as illustrated in Figure 11.3. Then, for each pair z_1, z_2 we evaluate the quantities

Figure 11.3 The Box-Muller method for generating Gaussian distributed random numbers starts by generating samples from a uniform distribution inside the unit circle.



$$y_1 = z_1 \left(\frac{-2 \ln z_1}{r^2} \right)^{1/2} \quad (11.10)$$

$$y_2 = z_2 \left(\frac{-2 \ln z_2}{r^2} \right)^{1/2} \quad (11.11)$$

Exercise 11.4

where $r^2 = z_1^2 + z_2^2$. Then the joint distribution of y_1 and y_2 is given by

$$\begin{aligned} p(y_1, y_2) &= p(z_1, z_2) \left| \frac{\partial(z_1, z_2)}{\partial(y_1, y_2)} \right| \\ &= \left[\frac{1}{\sqrt{2\pi}} \exp(-y_1^2/2) \right] \left[\frac{1}{\sqrt{2\pi}} \exp(-y_2^2/2) \right] \end{aligned} \quad (11.12)$$

and so y_1 and y_2 are independent and each has a Gaussian distribution with zero mean and unit variance.

If \mathbf{y} has a Gaussian distribution with zero mean and unit variance, then $\sigma\mathbf{y} + \mu$ will have a Gaussian distribution with mean μ and variance σ^2 . To generate vector-valued variables having a multivariate Gaussian distribution with mean μ and covariance Σ , we can make use of the *Cholesky decomposition*, which takes the form $\Sigma = \mathbf{L}\mathbf{L}^T$ (Press *et al.*, 1992). Then, if \mathbf{z} is a vector valued random variable whose components are independent and Gaussian distributed with zero mean and unit variance, then $\mathbf{y} = \mu + \mathbf{L}\mathbf{z}$ will have mean μ and covariance Σ .

Exercise 11.5

Obviously, the transformation technique depends for its success on the ability to calculate and then invert the indefinite integral of the required distribution. Such operations will only be feasible for a limited number of simple distributions, and so we must turn to alternative approaches in search of a more general strategy. Here we consider two techniques called *rejection sampling* and *importance sampling*. Although mainly limited to univariate distributions and thus not directly applicable to complex problems in many dimensions, they do form important components in more general strategies.

11.1.2 Rejection sampling

The rejection sampling framework allows us to sample from relatively complex distributions, subject to certain constraints. We begin by considering univariate distributions and discuss the extension to multiple dimensions subsequently.

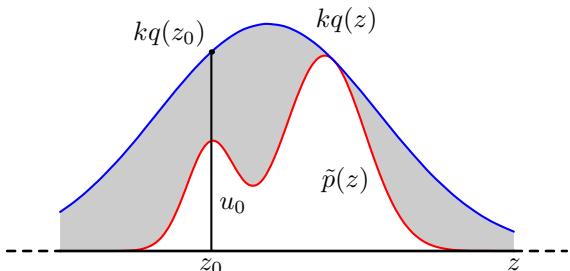
Suppose we wish to sample from a distribution $p(\mathbf{z})$ that is not one of the simple, standard distributions considered so far, and that sampling directly from $p(\mathbf{z})$ is difficult. Furthermore suppose, as is often the case, that we are easily able to evaluate $p(\mathbf{z})$ for any given value of \mathbf{z} , up to some normalizing constant Z , so that

$$p(z) = \frac{1}{Z_p} \tilde{p}(z) \quad (11.13)$$

where $\tilde{p}(z)$ can readily be evaluated, but Z_p is unknown.

In order to apply rejection sampling, we need some simpler distribution $q(z)$, sometimes called a *proposal distribution*, from which we can readily draw samples.

Figure 11.4 In the rejection sampling method, samples are drawn from a simple distribution $q(z)$ and rejected if they fall in the grey area between the unnormalized distribution $\tilde{p}(z)$ and the scaled distribution $kq(z)$. The resulting samples are distributed according to $p(z)$, which is the normalized version of $\tilde{p}(z)$.



We next introduce a constant k whose value is chosen such that $kq(z) \geq \tilde{p}(z)$ for all values of z . The function $kq(z)$ is called the comparison function and is illustrated for a univariate distribution in Figure 11.4. Each step of the rejection sampler involves generating two random numbers. First, we generate a number z_0 from the distribution $q(z)$. Next, we generate a number u_0 from the uniform distribution over $[0, kq(z_0)]$. This pair of random numbers has uniform distribution under the curve of the function $kq(z)$. Finally, if $u_0 > \tilde{p}(z_0)$ then the sample is rejected, otherwise u_0 is retained. Thus the pair is rejected if it lies in the grey shaded region in Figure 11.4. The remaining pairs then have uniform distribution under the curve of $\tilde{p}(z)$, and hence the corresponding z values are distributed according to $p(z)$, as desired.

The original values of z are generated from the distribution $q(z)$, and these samples are then accepted with probability $\tilde{p}(z)/kq(z)$, and so the probability that a sample will be accepted is given by

$$\begin{aligned} p(\text{accept}) &= \int \{\tilde{p}(z)/kq(z)\} q(z) dz \\ &= \frac{1}{k} \int \tilde{p}(z) dz. \end{aligned} \quad (11.14)$$

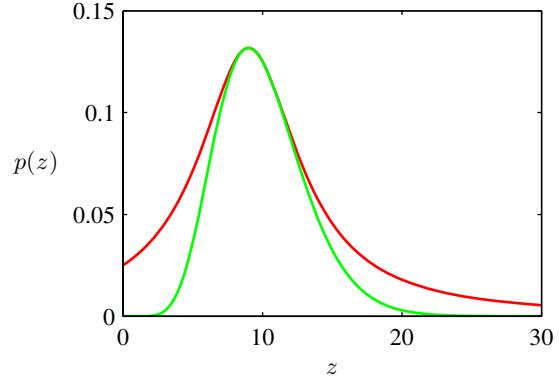
Thus the fraction of points that are rejected by this method depends on the ratio of the area under the unnormalized distribution $\tilde{p}(z)$ to the area under the curve $kq(z)$. We therefore see that the constant k should be as small as possible subject to the limitation that $kq(z)$ must be nowhere less than $\tilde{p}(z)$.

As an illustration of the use of rejection sampling, consider the task of sampling from the gamma distribution

$$\text{Gam}(z|a, b) = \frac{b^a z^{a-1} \exp(-bz)}{\Gamma(a)} \quad (11.15)$$

which, for $a > 1$, has a bell-shaped form, as shown in Figure 11.5. A suitable proposal distribution is therefore the Cauchy (11.8) because this too is bell-shaped and because we can use the transformation method, discussed earlier, to sample from it. We need to generalize the Cauchy slightly to ensure that it nowhere has a smaller value than the gamma distribution. This can be achieved by transforming a uniform random variable y using $z = b \tan y + c$, which gives random numbers distributed according to.

Figure 11.5 Plot showing the gamma distribution given by (11.15) as the green curve, with a scaled Cauchy proposal distribution shown by the red curve. Samples from the gamma distribution can be obtained by sampling from the Cauchy and then applying the rejection sampling criterion.



$$q(z) = \frac{k}{1 + (z - c)^2/b^2}. \quad (11.16)$$

The minimum reject rate is obtained by setting $c = a - 1$, $b^2 = 2a - 1$ and choosing the constant k to be as small as possible while still satisfying the requirement $kq(z) \geq \tilde{p}(z)$. The resulting comparison function is also illustrated in Figure 11.5.

11.1.3 Adaptive rejection sampling

In many instances where we might wish to apply rejection sampling, it proves difficult to determine a suitable analytic form for the envelope distribution $q(z)$. An alternative approach is to construct the envelope function on the fly based on measured values of the distribution $p(z)$ (Gilks and Wild, 1992). Construction of an envelope function is particularly straightforward for cases in which $p(z)$ is log concave, in other words when $\ln p(z)$ has derivatives that are nonincreasing functions of z . The construction of a suitable envelope function is illustrated graphically in Figure 11.6.

The function $\ln p(z)$ and its gradient are evaluated at some initial set of grid points, and the intersections of the resulting tangent lines are used to construct the envelope function. Next a sample value is drawn from the envelope distribution. This is straightforward because the log of the envelope distribution is a succession

Exercise 11.9

Figure 11.6 In the case of distributions that are log concave, an envelope function for use in rejection sampling can be constructed using the tangent lines computed at a set of grid points. If a sample point is rejected, it is added to the set of grid points and used to refine the envelope distribution.

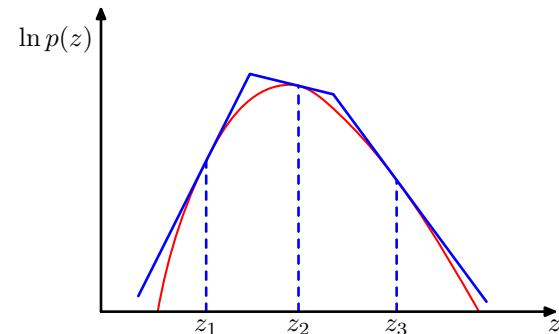
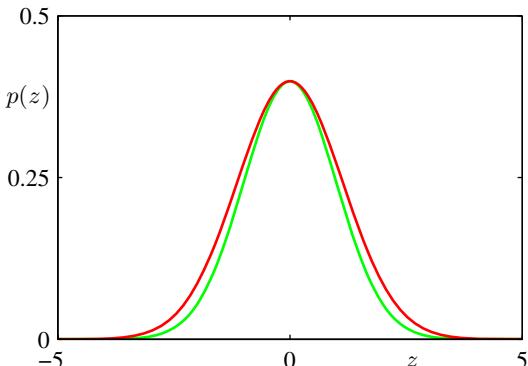


Figure 11.7 Illustrative example of rejection sampling involving sampling from a Gaussian distribution $p(z)$ shown by the green curve, by using rejection sampling from a proposal distribution $q(z)$ that is also Gaussian and whose scaled version $kq(z)$ is shown by the red curve.



of linear functions, and hence the envelope distribution itself comprises a piecewise exponential distribution of the form

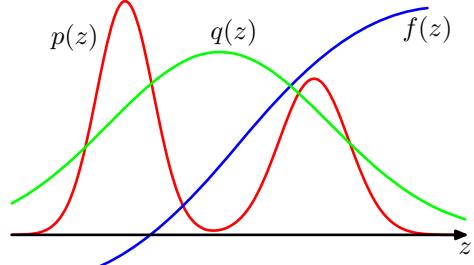
$$q(z) = k_i \lambda_i \exp \{-\lambda_i(z - z_{i-1})\} \quad z_{i-1} < z \leq z_i. \quad (11.17)$$

Once a sample has been drawn, the usual rejection criterion can be applied. If the sample is accepted, then it will be a draw from the desired distribution. If, however, the sample is rejected, then it is incorporated into the set of grid points, a new tangent line is computed, and the envelope function is thereby refined. As the number of grid points increases, so the envelope function becomes a better approximation of the desired distribution $p(z)$ and the probability of rejection decreases.

A variant of the algorithm exists that avoids the evaluation of derivatives (Gilks, 1992). The adaptive rejection sampling framework can also be extended to distributions that are not log concave, simply by following each rejection sampling step with a Metropolis-Hastings step (to be discussed in Section 11.2.2), giving rise to *adaptive rejection Metropolis* sampling (Gilks *et al.*, 1995).

Clearly for rejection sampling to be of practical value, we require that the comparison function be close to the required distribution so that the rate of rejection is kept to a minimum. Now let us examine what happens when we try to use rejection sampling in spaces of high dimensionality. Consider, for the sake of illustration, a somewhat artificial problem in which we wish to sample from a zero-mean multivariate Gaussian distribution with covariance $\sigma_p^2 \mathbf{I}$, where \mathbf{I} is the unit matrix, by rejection sampling from a proposal distribution that is itself a zero-mean Gaussian distribution having covariance $\sigma_q^2 \mathbf{I}$. Obviously, we must have $\sigma_q^2 \geq \sigma_p^2$ in order that there exists a k such that $kq(z) \geq p(z)$. In D -dimensions the optimum value of k is given by $k = (\sigma_q/\sigma_p)^D$, as illustrated for $D = 1$ in Figure 11.7. The acceptance rate will be the ratio of volumes under $p(z)$ and $kq(z)$, which, because both distributions are normalized, is just $1/k$. Thus the acceptance rate diminishes exponentially with dimensionality. Even if σ_q exceeds σ_p by just one percent, for $D = 1,000$ the acceptance ratio will be approximately $1/20,000$. In this illustrative example the comparison function is close to the required distribution. For more practical examples, where the desired distribution may be multimodal and sharply peaked, it will be extremely difficult to find a good proposal distribution and comparison function.

Figure 11.8 Importance sampling addresses the problem of evaluating the expectation of a function $f(z)$ with respect to a distribution $p(z)$ from which it is difficult to draw samples directly. Instead, samples $\{z^{(l)}\}$ are drawn from a simpler distribution $q(z)$, and the corresponding terms in the summation are weighted by the ratios $p(z^{(l)})/q(z^{(l)})$.



Furthermore, the exponential decrease of acceptance rate with dimensionality is a generic feature of rejection sampling. Although rejection can be a useful technique in one or two dimensions it is unsuited to problems of high dimensionality. It can, however, play a role as a subroutine in more sophisticated algorithms for sampling in high dimensional spaces.

11.1.4 Importance sampling

One of the principal reasons for wishing to sample from complicated probability distributions is to be able to evaluate expectations of the form (11.1). The technique of *importance sampling* provides a framework for approximating expectations directly but does not itself provide a mechanism for drawing samples from distribution $p(\mathbf{z})$.

The finite sum approximation to the expectation, given by (11.2), depends on being able to draw samples from the distribution $p(\mathbf{z})$. Suppose, however, that it is impractical to sample directly from $p(\mathbf{z})$ but that we can evaluate $p(\mathbf{z})$ easily for any given value of \mathbf{z} . One simplistic strategy for evaluating expectations would be to discretize \mathbf{z} -space into a uniform grid and to evaluate the integrand as a sum of the form

$$\mathbb{E}[f] \simeq \sum_{l=1}^L p(\mathbf{z}^{(l)}) f(\mathbf{z}^{(l)}). \quad (11.18)$$

An obvious problem with this approach is that the number of terms in the summation grows exponentially with the dimensionality of \mathbf{z} . Furthermore, as we have already noted, the kinds of probability distributions of interest will often have much of their mass confined to relatively small regions of \mathbf{z} space and so uniform sampling will be very inefficient because in high-dimensional problems, only a very small proportion of the samples will make a significant contribution to the sum. We would really like to choose the sample points to fall in regions where $p(\mathbf{z})$ is large, or ideally where the product $p(\mathbf{z}) f(\mathbf{z})$ is large.

As in the case of rejection sampling, importance sampling is based on the use of a proposal distribution $q(\mathbf{z})$ from which it is easy to draw samples, as illustrated in Figure 11.8. We can then express the expectation in the form of a finite sum over

samples $\{\mathbf{z}^{(l)}\}$ drawn from $q(\mathbf{z})$

$$\begin{aligned}\mathbb{E}[f] &= \int f(\mathbf{z})p(\mathbf{z}) d\mathbf{z} \\ &= \int f(\mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z})} q(\mathbf{z}) d\mathbf{z} \\ &\simeq \frac{1}{L} \sum_{l=1}^L \frac{p(\mathbf{z}^{(l)})}{q(\mathbf{z}^{(l)})} f(\mathbf{z}^{(l)}).\end{aligned}\quad (11.19)$$

The quantities $r_l = p(\mathbf{z}^{(l)})/q(\mathbf{z}^{(l)})$ are known as *importance weights*, and they correct the bias introduced by sampling from the wrong distribution. Note that, unlike rejection sampling, all of the samples generated are retained.

It will often be the case that the distribution $p(\mathbf{z})$ can only be evaluated up to a normalization constant, so that $p(\mathbf{z}) = \tilde{p}(\mathbf{z})/Z_p$ where $\tilde{p}(\mathbf{z})$ can be evaluated easily, whereas Z_p is unknown. Similarly, we may wish to use an importance sampling distribution $q(\mathbf{z}) = \tilde{q}(\mathbf{z})/Z_q$, which has the same property. We then have

$$\begin{aligned}\mathbb{E}[f] &= \int f(\mathbf{z})p(\mathbf{z}) d\mathbf{z} \\ &= \frac{Z_q}{Z_p} \int f(\mathbf{z}) \frac{\tilde{p}(\mathbf{z})}{\tilde{q}(\mathbf{z})} q(\mathbf{z}) d\mathbf{z} \\ &\simeq \frac{Z_q}{Z_p} \frac{1}{L} \sum_{l=1}^L \tilde{r}_l f(\mathbf{z}^{(l)}).\end{aligned}\quad (11.20)$$

where $\tilde{r}_l = \tilde{p}(\mathbf{z}^{(l)})/\tilde{q}(\mathbf{z}^{(l)})$. We can use the same sample set to evaluate the ratio Z_p/Z_q with the result

$$\begin{aligned}\frac{Z_p}{Z_q} &= \frac{1}{Z_q} \int \tilde{p}(\mathbf{z}) d\mathbf{z} = \int \frac{\tilde{p}(\mathbf{z})}{\tilde{q}(\mathbf{z})} q(\mathbf{z}) d\mathbf{z} \\ &\simeq \frac{1}{L} \sum_{l=1}^L \tilde{r}_l\end{aligned}\quad (11.21)$$

and hence

$$\mathbb{E}[f] \simeq \sum_{l=1}^L w_l f(\mathbf{z}^{(l)}) \quad (11.22)$$

where we have defined

$$w_l = \frac{\tilde{r}_l}{\sum_m \tilde{r}_m} = \frac{\tilde{p}(\mathbf{z}^{(l)})/q(\mathbf{z}^{(l)})}{\sum_m \tilde{p}(\mathbf{z}^{(m)})/q(\mathbf{z}^{(m)})}.\quad (11.23)$$

As with rejection sampling, the success of the importance sampling approach depends crucially on how well the sampling distribution $q(\mathbf{z})$ matches the desired

distribution $p(\mathbf{z})$. If, as is often the case, $p(\mathbf{z})f(\mathbf{z})$ is strongly varying and has a significant proportion of its mass concentrated over relatively small regions of \mathbf{z} space, then the set of importance weights $\{r_l\}$ may be dominated by a few weights having large values, with the remaining weights being relatively insignificant. Thus the effective sample size can be much smaller than the apparent sample size L . The problem is even more severe if none of the samples falls in the regions where $p(\mathbf{z})f(\mathbf{z})$ is large. In that case, the apparent variances of r_l and $r_l f(\mathbf{z}^{(l)})$ may be small even though the estimate of the expectation may be severely wrong. Hence a major drawback of the importance sampling method is the potential to produce results that are arbitrarily in error and with no diagnostic indication. This also highlights a key requirement for the sampling distribution $q(\mathbf{z})$, namely that it should not be small or zero in regions where $p(\mathbf{z})$ may be significant.

For distributions defined in terms of a graphical model, we can apply the importance sampling technique in various ways. For discrete variables, a simple approach is called *uniform sampling*. The joint distribution for a directed graph is defined by (11.4). Each sample from the joint distribution is obtained by first setting those variables \mathbf{z}_i that are in the evidence set equal to their observed values. Each of the remaining variables is then sampled independently from a uniform distribution over the space of possible instantiations. To determine the corresponding weight associated with a sample $\mathbf{z}^{(l)}$, we note that the sampling distribution $\tilde{q}(\mathbf{z})$ is uniform over the possible choices for \mathbf{z} , and that $\tilde{p}(\mathbf{z}|\mathbf{x}) = \tilde{p}(\mathbf{z})$, where \mathbf{x} denotes the subset of variables that are observed, and the equality follows from the fact that every sample \mathbf{z} that is generated is necessarily consistent with the evidence. Thus the weights r_l are simply proportional to $p(\mathbf{z})$. Note that the variables can be sampled in any order. This approach can yield poor results if the posterior distribution is far from uniform, as is often the case in practice.

An improvement on this approach is called *likelihood weighted sampling* (Fung and Chang, 1990; Shachter and Peot, 1990) and is based on ancestral sampling of the variables. For each variable in turn, if that variable is in the evidence set, then it is just set to its instantiated value. If it is not in the evidence set, then it is sampled from the conditional distribution $p(\mathbf{z}_i|\text{pa}_i)$ in which the conditioning variables are set to their currently sampled values. The weighting associated with the resulting sample \mathbf{z} is then given by

$$r(\mathbf{z}) = \prod_{\mathbf{z}_i \notin \mathbf{e}} \frac{p(\mathbf{z}_i|\text{pa}_i)}{p(\mathbf{z}_i|\text{pa}_i)} \prod_{\mathbf{z}_i \in \mathbf{e}} \frac{p(\mathbf{z}_i|\text{pa}_i)}{1} = \prod_{\mathbf{z}_i \in \mathbf{e}} p(\mathbf{z}_i|\text{pa}_i). \quad (11.24)$$

This method can be further extended using *self-importance sampling* (Shachter and Peot, 1990) in which the importance sampling distribution is continually updated to reflect the current estimated posterior distribution.

11.1.5 Sampling-importance-resampling

The rejection sampling method discussed in Section 11.1.2 depends in part for its success on the determination of a suitable value for the constant k . For many pairs of distributions $p(\mathbf{z})$ and $q(\mathbf{z})$, it will be impractical to determine a suitable

value for k in that any value that is sufficiently large to guarantee a bound on the desired distribution will lead to impractically small acceptance rates.

As in the case of rejection sampling, the *sampling-importance-resampling* (SIR) approach also makes use of a sampling distribution $q(\mathbf{z})$ but avoids having to determine the constant k . There are two stages to the scheme. In the first stage, L samples $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}$ are drawn from $q(\mathbf{z})$. Then in the second stage, weights w_1, \dots, w_L are constructed using (11.23). Finally, a second set of L samples is drawn from the discrete distribution $(\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)})$ with probabilities given by the weights (w_1, \dots, w_L) .

The resulting L samples are only approximately distributed according to $p(\mathbf{z})$, but the distribution becomes correct in the limit $L \rightarrow \infty$. To see this, consider the univariate case, and note that the cumulative distribution of the resampled values is given by

$$\begin{aligned} p(z \leq a) &= \sum_{l: z^{(l)} \leq a} w_l \\ &= \frac{\sum_l I(z^{(l)} \leq a) \tilde{p}(z^{(l)}) / q(z^{(l)})}{\sum_l \tilde{p}(z^{(l)}) / q(z^{(l)})} \end{aligned} \quad (11.25)$$

where $I(\cdot)$ is the indicator function (which equals 1 if its argument is true and 0 otherwise). Taking the limit $L \rightarrow \infty$, and assuming suitable regularity of the distributions, we can replace the sums by integrals weighted according to the original sampling distribution $q(z)$

$$\begin{aligned} p(z \leq a) &= \frac{\int I(z \leq a) \{\tilde{p}(z)/q(z)\} q(z) dz}{\int \{\tilde{p}(z)/q(z)\} q(z) dz} \\ &= \frac{\int I(z \leq a) \tilde{p}(z) dz}{\int \tilde{p}(z) dz} \\ &= \int I(z \leq a) p(z) dz \end{aligned} \quad (11.26)$$

which is the cumulative distribution function of $p(z)$. Again, we see that the normalization of $p(z)$ is not required.

For a finite value of L , and a given initial sample set, the resampled values will only approximately be drawn from the desired distribution. As with rejection sampling, the approximation improves as the sampling distribution $q(\mathbf{z})$ gets closer to the desired distribution $p(\mathbf{z})$. When $q(\mathbf{z}) = p(\mathbf{z})$, the initial samples $(\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)})$ have the desired distribution, and the weights $w_n = 1/L$ so that the resampled values also have the desired distribution.

If moments with respect to the distribution $p(\mathbf{z})$ are required, then they can be

evaluated directly using the original samples together with the weights, because

$$\begin{aligned}\mathbb{E}[f(\mathbf{z})] &= \int f(\mathbf{z})p(\mathbf{z}) d\mathbf{z} \\ &= \frac{\int f(\mathbf{z})[\tilde{p}(\mathbf{z})/q(\mathbf{z})]q(\mathbf{z}) d\mathbf{z}}{\int [\tilde{p}(\mathbf{z})/q(\mathbf{z})]q(\mathbf{z}) d\mathbf{z}} \\ &\simeq \sum_{l=1}^L w_l f(\mathbf{z}_l).\end{aligned}\tag{11.27}$$

11.1.6 Sampling and the EM algorithm

In addition to providing a mechanism for direct implementation of the Bayesian framework, Monte Carlo methods can also play a role in the frequentist paradigm, for example to find maximum likelihood solutions. In particular, sampling methods can be used to approximate the E step of the EM algorithm for models in which the E step cannot be performed analytically. Consider a model with hidden variables \mathbf{Z} , visible (observed) variables \mathbf{X} , and parameters $\boldsymbol{\theta}$. The function that is optimized with respect to $\boldsymbol{\theta}$ in the M step is the expected complete-data log likelihood, given by

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \int p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{Z}, \mathbf{X}|\boldsymbol{\theta}) d\mathbf{Z}.\tag{11.28}$$

We can use sampling methods to approximate this integral by a finite sum over samples $\{\mathbf{Z}^{(l)}\}$, which are drawn from the current estimate for the posterior distribution $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$, so that

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) \simeq \frac{1}{L} \sum_{l=1}^L \ln p(\mathbf{Z}^{(l)}, \mathbf{X}|\boldsymbol{\theta}).\tag{11.29}$$

The Q function is then optimized in the usual way in the M step. This procedure is called the *Monte Carlo EM algorithm*.

It is straightforward to extend this to the problem of finding the mode of the posterior distribution over $\boldsymbol{\theta}$ (the MAP estimate) when a prior distribution $p(\boldsymbol{\theta})$ has been defined, simply by adding $\ln p(\boldsymbol{\theta})$ to the function $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ before performing the M step.

A particular instance of the Monte Carlo EM algorithm, called *stochastic EM*, arises if we consider a finite mixture model, and draw just one sample at each E step. Here the latent variable \mathbf{Z} characterizes which of the K components of the mixture is responsible for generating each data point. In the E step, a sample of \mathbf{Z} is taken from the posterior distribution $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$ where \mathbf{X} is the data set. This effectively makes a hard assignment of each data point to one of the components in the mixture. In the M step, this sampled approximation to the posterior distribution is used to update the model parameters in the usual way.

Now suppose we move from a maximum likelihood approach to a full Bayesian treatment in which we wish to sample from the posterior distribution over the parameter vector θ . In principle, we would like to draw samples from the joint posterior $p(\theta, \mathbf{Z}|\mathbf{X})$, but we shall suppose that this is computationally difficult. Suppose further that it is relatively straightforward to sample from the complete-data parameter posterior $p(\theta|\mathbf{Z}, \mathbf{X})$. This inspires the *data augmentation* algorithm, which alternates between two steps known as the I-step (imputation step, analogous to an E step) and the P-step (posterior step, analogous to an M step).

IP Algorithm

I-step. We wish to sample from $p(\mathbf{Z}|\mathbf{X})$ but we cannot do this directly. We therefore note the relation

$$p(\mathbf{Z}|\mathbf{X}) = \int p(\mathbf{Z}|\theta, \mathbf{X})p(\theta|\mathbf{X}) d\theta \quad (11.30)$$

and hence for $l = 1, \dots, L$ we first draw a sample $\theta^{(l)}$ from the current estimate for $p(\theta|\mathbf{X})$, and then use this to draw a sample $\mathbf{Z}^{(l)}$ from $p(\mathbf{Z}|\theta^{(l)}, \mathbf{X})$.

P-step. Given the relation

$$p(\theta|\mathbf{X}) = \int p(\theta|\mathbf{Z}, \mathbf{X})p(\mathbf{Z}|\mathbf{X}) d\mathbf{Z} \quad (11.31)$$

we use the samples $\{\mathbf{Z}^{(l)}\}$ obtained from the I-step to compute a revised estimate of the posterior distribution over θ given by

$$p(\theta|\mathbf{X}) \simeq \frac{1}{L} \sum_{l=1}^L p(\theta|\mathbf{Z}^{(l)}, \mathbf{X}). \quad (11.32)$$

By assumption, it will be feasible to sample from this approximation in the I-step.

Note that we are making a (somewhat artificial) distinction between parameters θ and hidden variables \mathbf{Z} . From now on, we blur this distinction and focus simply on the problem of drawing samples from a given posterior distribution.

11.2. Markov Chain Monte Carlo

In the previous section, we discussed the rejection sampling and importance sampling strategies for evaluating expectations of functions, and we saw that they suffer from severe limitations particularly in spaces of high dimensionality. We therefore turn in this section to a very general and powerful framework called Markov chain Monte Carlo (MCMC), which allows sampling from a large class of distributions,

and which scales well with the dimensionality of the sample space. Markov chain Monte Carlo methods have their origins in physics (Metropolis and Ulam, 1949), and it was only towards the end of the 1980s that they started to have a significant impact in the field of statistics.

As with rejection and importance sampling, we again sample from a proposal distribution. This time, however, we maintain a record of the current state $\mathbf{z}^{(\tau)}$, and the proposal distribution $q(\mathbf{z}|\mathbf{z}^{(\tau)})$ depends on this current state, and so the sequence of samples $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \dots$ forms a Markov chain. Again, if we write $p(\mathbf{z}) = \tilde{p}(\mathbf{z})/Z_p$, we will assume that $\tilde{p}(\mathbf{z})$ can readily be evaluated for any given value of \mathbf{z} , although the value of Z_p may be unknown. The proposal distribution itself is chosen to be sufficiently simple that it is straightforward to draw samples from it directly. At each cycle of the algorithm, we generate a candidate sample \mathbf{z}^* from the proposal distribution and then accept the sample according to an appropriate criterion.

In the basic *Metropolis* algorithm (Metropolis *et al.*, 1953), we assume that the proposal distribution is symmetric, that is $q(\mathbf{z}_A|\mathbf{z}_B) = q(\mathbf{z}_B|\mathbf{z}_A)$ for all values of \mathbf{z}_A and \mathbf{z}_B . The candidate sample is then accepted with probability

$$A(\mathbf{z}^*, \mathbf{z}^{(\tau)}) = \min \left(1, \frac{\tilde{p}(\mathbf{z}^*)}{\tilde{p}(\mathbf{z}^{(\tau)})} \right). \quad (11.33)$$

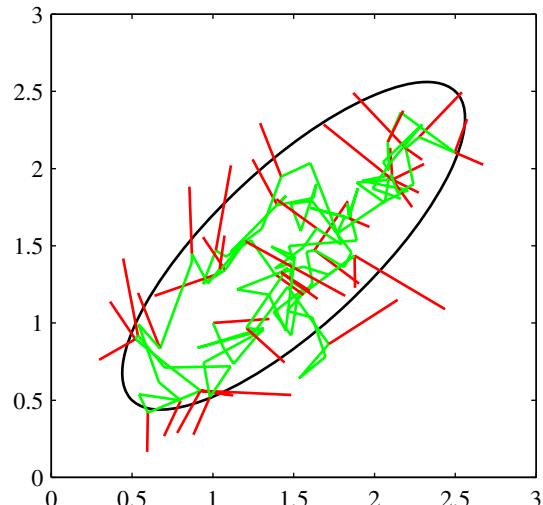
This can be achieved by choosing a random number u with uniform distribution over the unit interval $(0, 1)$ and then accepting the sample if $A(\mathbf{z}^*, \mathbf{z}^{(\tau)}) > u$. Note that if the step from $\mathbf{z}^{(\tau)}$ to \mathbf{z}^* causes an increase in the value of $p(\mathbf{z})$, then the candidate point is certain to be kept.

If the candidate sample is accepted, then $\mathbf{z}^{(\tau+1)} = \mathbf{z}^*$, otherwise the candidate point \mathbf{z}^* is discarded, $\mathbf{z}^{(\tau+1)}$ is set to $\mathbf{z}^{(\tau)}$ and another candidate sample is drawn from the distribution $q(\mathbf{z}|\mathbf{z}^{(\tau+1)})$. This is in contrast to rejection sampling, where rejected samples are simply discarded. In the Metropolis algorithm when a candidate point is rejected, the previous sample is included instead in the final list of samples, leading to multiple copies of samples. Of course, in a practical implementation, only a single copy of each retained sample would be kept, along with an integer weighting factor recording how many times that state appears. As we shall see, as long as $q(\mathbf{z}_A|\mathbf{z}_B)$ is positive for any values of \mathbf{z}_A and \mathbf{z}_B (this is a sufficient but not necessary condition), the distribution of $\mathbf{z}^{(\tau)}$ tends to $p(\mathbf{z})$ as $\tau \rightarrow \infty$. It should be emphasized, however, that the sequence $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \dots$ is not a set of independent samples from $p(\mathbf{z})$ because successive samples are highly correlated. If we wish to obtain independent samples, then we can discard most of the sequence and just retain every M^{th} sample. For M sufficiently large, the retained samples will for all practical purposes be independent. Figure 11.9 shows a simple illustrative example of sampling from a two-dimensional Gaussian distribution using the Metropolis algorithm in which the proposal distribution is an isotropic Gaussian.

Further insight into the nature of Markov chain Monte Carlo algorithms can be gleaned by looking at the properties of a specific example, namely a simple random

Section 11.2.1

Figure 11.9 A simple illustration using Metropolis algorithm to sample from a Gaussian distribution whose one standard-deviation contour is shown by the ellipse. The proposal distribution is an isotropic Gaussian distribution whose standard deviation is 0.2. Steps that are accepted are shown as green lines, and rejected steps are shown in red. A total of 150 candidate samples are generated, of which 43 are rejected.



walk. Consider a state space z consisting of the integers, with probabilities

$$p(z^{(\tau+1)} = z^{(\tau)}) = 0.5 \quad (11.34)$$

$$p(z^{(\tau+1)} = z^{(\tau)} + 1) = 0.25 \quad (11.35)$$

$$p(z^{(\tau+1)} = z^{(\tau)} - 1) = 0.25 \quad (11.36)$$

where $z^{(\tau)}$ denotes the state at step τ . If the initial state is $z^{(1)} = 0$, then by symmetry the expected state at time τ will also be zero $\mathbb{E}[z^{(\tau)}] = 0$, and similarly it is easily seen that $\mathbb{E}[(z^{(\tau)})^2] = \tau/2$. Thus after τ steps, the random walk has only travelled a distance that on average is proportional to the square root of τ . This square root dependence is typical of random walk behaviour and shows that random walks are very inefficient in exploring the state space. As we shall see, a central goal in designing Markov chain Monte Carlo methods is to avoid random walk behaviour.

11.2.1 Markov chains

Before discussing Markov chain Monte Carlo methods in more detail, it is useful to study some general properties of Markov chains in more detail. In particular, we ask under what circumstances will a Markov chain converge to the desired distribution. A first-order Markov chain is defined to be a series of random variables $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(M)}$ such that the following conditional independence property holds for $m \in \{1, \dots, M-1\}$

$$p(\mathbf{z}^{(m+1)} | \mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}) = p(\mathbf{z}^{(m+1)} | \mathbf{z}^{(m)}). \quad (11.37)$$

This of course can be represented as a directed graph in the form of a chain, an example of which is shown in Figure 8.38. We can then specify the Markov chain by giving the probability distribution for the initial variable $p(\mathbf{z}^{(0)})$ together with the

Exercise 11.10

conditional probabilities for subsequent variables in the form of *transition probabilities* $T_m(\mathbf{z}^{(m)}, \mathbf{z}^{(m+1)}) \equiv p(\mathbf{z}^{(m+1)} | \mathbf{z}^{(m)})$. A Markov chain is called *homogeneous* if the transition probabilities are the same for all m .

The marginal probability for a particular variable can be expressed in terms of the marginal probability for the previous variable in the chain in the form

$$p(\mathbf{z}^{(m+1)}) = \sum_{\mathbf{z}^{(m)}} p(\mathbf{z}^{(m+1)} | \mathbf{z}^{(m)}) p(\mathbf{z}^{(m)}). \quad (11.38)$$

A distribution is said to be invariant, or stationary, with respect to a Markov chain if each step in the chain leaves that distribution invariant. Thus, for a homogeneous Markov chain with transition probabilities $T(\mathbf{z}', \mathbf{z})$, the distribution $p^*(\mathbf{z})$ is invariant if

$$p^*(\mathbf{z}) = \sum_{\mathbf{z}'} T(\mathbf{z}', \mathbf{z}) p^*(\mathbf{z}'). \quad (11.39)$$

Note that a given Markov chain may have more than one invariant distribution. For instance, if the transition probabilities are given by the identity transformation, then any distribution will be invariant.

A sufficient (but not necessary) condition for ensuring that the required distribution $p(\mathbf{z})$ is invariant is to choose the transition probabilities to satisfy the property of *detailed balance*, defined by

$$p^*(\mathbf{z}) T(\mathbf{z}, \mathbf{z}') = p^*(\mathbf{z}') T(\mathbf{z}', \mathbf{z}) \quad (11.40)$$

for the particular distribution $p^*(\mathbf{z})$. It is easily seen that a transition probability that satisfies detailed balance with respect to a particular distribution will leave that distribution invariant, because

$$\sum_{\mathbf{z}'} p^*(\mathbf{z}') T(\mathbf{z}', \mathbf{z}) = \sum_{\mathbf{z}'} p^*(\mathbf{z}) T(\mathbf{z}, \mathbf{z}') = p^*(\mathbf{z}) \sum_{\mathbf{z}'} p(\mathbf{z}' | \mathbf{z}) = p^*(\mathbf{z}). \quad (11.41)$$

A Markov chain that respects detailed balance is said to be *reversible*.

Our goal is to use Markov chains to sample from a given distribution. We can achieve this if we set up a Markov chain such that the desired distribution is invariant. However, we must also require that for $m \rightarrow \infty$, the distribution $p(\mathbf{z}^{(m)})$ converges to the required invariant distribution $p^*(\mathbf{z})$, irrespective of the choice of initial distribution $p(\mathbf{z}^{(0)})$. This property is called *ergodicity*, and the invariant distribution is then called the *equilibrium* distribution. Clearly, an ergodic Markov chain can have only one equilibrium distribution. It can be shown that a homogeneous Markov chain will be ergodic, subject only to weak restrictions on the invariant distribution and the transition probabilities (Neal, 1993).

In practice we often construct the transition probabilities from a set of ‘base’ transitions B_1, \dots, B_K . This can be achieved through a mixture distribution of the form

$$T(\mathbf{z}', \mathbf{z}) = \sum_{k=1}^K \alpha_k B_k(\mathbf{z}', \mathbf{z}) \quad (11.42)$$

for some set of mixing coefficients $\alpha_1, \dots, \alpha_K$ satisfying $\alpha_k \geq 0$ and $\sum_k \alpha_k = 1$. Alternatively, the base transitions may be combined through successive application, so that

$$T(\mathbf{z}', \mathbf{z}) = \sum_{\mathbf{z}_1} \dots \sum_{\mathbf{z}_{n-1}} B_1(\mathbf{z}', \mathbf{z}_1) \dots B_{K-1}(\mathbf{z}_{K-2}, \mathbf{z}_{K-1}) B_K(\mathbf{z}_{K-1}, \mathbf{z}). \quad (11.43)$$

If a distribution is invariant with respect to each of the base transitions, then obviously it will also be invariant with respect to either of the $T(\mathbf{z}', \mathbf{z})$ given by (11.42) or (11.43). For the case of the mixture (11.42), if each of the base transitions satisfies detailed balance, then the mixture transition T will also satisfy detailed balance. This does not hold for the transition probability constructed using (11.43), although by symmetrizing the order of application of the base transitions, in the form $B_1, B_2, \dots, B_K, B_K, \dots, B_2, B_1$, detailed balance can be restored. A common example of the use of composite transition probabilities is where each base transition changes only a subset of the variables.

11.2.2 The Metropolis-Hastings algorithm

Earlier we introduced the basic Metropolis algorithm, without actually demonstrating that it samples from the required distribution. Before giving a proof, we first discuss a generalization, known as the *Metropolis-Hastings* algorithm (Hastings, 1970), to the case where the proposal distribution is no longer a symmetric function of its arguments. In particular at step τ of the algorithm, in which the current state is $\mathbf{z}^{(\tau)}$, we draw a sample \mathbf{z}^* from the distribution $q_k(\mathbf{z}^* | \mathbf{z}^{(\tau)})$ and then accept it with probability $A_k(\mathbf{z}^*, \mathbf{z}_\tau)$ where

$$A_k(\mathbf{z}^*, \mathbf{z}^{(\tau)}) = \min \left(1, \frac{\tilde{p}(\mathbf{z}^*) q_k(\mathbf{z}^{(\tau)} | \mathbf{z}^*)}{\tilde{p}(\mathbf{z}^{(\tau)}) q_k(\mathbf{z}^* | \mathbf{z}^{(\tau)})} \right). \quad (11.44)$$

Here k labels the members of the set of possible transitions being considered. Again, the evaluation of the acceptance criterion does not require knowledge of the normalizing constant Z_p in the probability distribution $p(\mathbf{z}) = \tilde{p}(\mathbf{z})/Z_p$. For a symmetric proposal distribution the Metropolis-Hastings criterion (11.44) reduces to the standard Metropolis criterion given by (11.33).

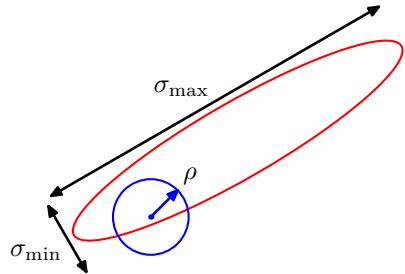
We can show that $p(\mathbf{z})$ is an invariant distribution of the Markov chain defined by the Metropolis-Hastings algorithm by showing that detailed balance, defined by (11.40), is satisfied. Using (11.44) we have

$$\begin{aligned} p(\mathbf{z}) q_k(\mathbf{z}' | \mathbf{z}') A_k(\mathbf{z}', \mathbf{z}) &= \min(p(\mathbf{z}) q_k(\mathbf{z}' | \mathbf{z}), p(\mathbf{z}') q_k(\mathbf{z}' | \mathbf{z})) \\ &= \min(p(\mathbf{z}') q_k(\mathbf{z}' | \mathbf{z}), p(\mathbf{z}) q_k(\mathbf{z} | \mathbf{z}')) \\ &= p(\mathbf{z}') q_k(\mathbf{z}' | \mathbf{z}) A_k(\mathbf{z}, \mathbf{z}') \end{aligned} \quad (11.45)$$

as required.

The specific choice of proposal distribution can have a marked effect on the performance of the algorithm. For continuous state spaces, a common choice is a Gaussian centred on the current state, leading to an important trade-off in determining the variance parameter of this distribution. If the variance is small, then the

Figure 11.10 Schematic illustration of the use of an isotropic Gaussian proposal distribution (blue circle) to sample from a correlated multivariate Gaussian distribution (red ellipse) having very different standard deviations in different directions, using the Metropolis-Hastings algorithm. In order to keep the rejection rate low, the scale ρ of the proposal distribution should be on the order of the smallest standard deviation σ_{\min} , which leads to random walk behaviour in which the number of steps separating states that are approximately independent is of order $(\sigma_{\max}/\sigma_{\min})^2$ where σ_{\max} is the largest standard deviation.



proportion of accepted transitions will be high, but progress through the state space takes the form of a slow random walk leading to long correlation times. However, if the variance parameter is large, then the rejection rate will be high because, in the kind of complex problems we are considering, many of the proposed steps will be to states for which the probability $p(\mathbf{z})$ is low. Consider a multivariate distribution $p(\mathbf{z})$ having strong correlations between the components of \mathbf{z} , as illustrated in Figure 11.10. The scale ρ of the proposal distribution should be as large as possible without incurring high rejection rates. This suggests that ρ should be of the same order as the smallest length scale σ_{\min} . The system then explores the distribution along the more extended direction by means of a random walk, and so the number of steps to arrive at a state that is more or less independent of the original state is of order $(\sigma_{\max}/\sigma_{\min})^2$. In fact in two dimensions, the increase in rejection rate as ρ increases is offset by the larger steps sizes of those transitions that are accepted, and more generally for a multivariate Gaussian the number of steps required to obtain independent samples scales like $(\sigma_{\max}/\sigma_2)^2$ where σ_2 is the second-smallest standard deviation (Neal, 1993). These details aside, it remains the case that if the length scales over which the distributions vary are very different in different directions, then the Metropolis Hastings algorithm can have very slow convergence.

11.3. Gibbs Sampling

Gibbs sampling (Geman and Geman, 1984) is a simple and widely applicable Markov chain Monte Carlo algorithm and can be seen as a special case of the Metropolis-Hastings algorithm.

Consider the distribution $p(\mathbf{z}) = p(z_1, \dots, z_M)$ from which we wish to sample, and suppose that we have chosen some initial state for the Markov chain. Each step of the Gibbs sampling procedure involves replacing the value of one of the variables by a value drawn from the distribution of that variable conditioned on the values of the remaining variables. Thus we replace z_i by a value drawn from the distribution $p(z_i | \mathbf{z}_{\setminus i})$, where z_i denotes the i^{th} component of \mathbf{z} , and $\mathbf{z}_{\setminus i}$ denotes z_1, \dots, z_M but with z_i omitted. This procedure is repeated either by cycling through the variables

in some particular order or by choosing the variable to be updated at each step at random from some distribution.

For example, suppose we have a distribution $p(z_1, z_2, z_3)$ over three variables, and at step τ of the algorithm we have selected values $z_1^{(\tau)}, z_2^{(\tau)}$ and $z_3^{(\tau)}$. We first replace $z_1^{(\tau)}$ by a new value $z_1^{(\tau+1)}$ obtained by sampling from the conditional distribution

$$p(z_1|z_2^{(\tau)}, z_3^{(\tau)}). \quad (11.46)$$

Next we replace $z_2^{(\tau)}$ by a value $z_2^{(\tau+1)}$ obtained by sampling from the conditional distribution

$$p(z_2|z_1^{(\tau+1)}, z_3^{(\tau)}) \quad (11.47)$$

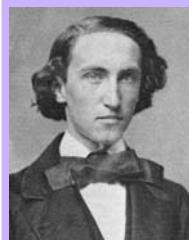
so that the new value for z_1 is used straight away in subsequent sampling steps. Then we update z_3 with a sample $z_3^{(\tau+1)}$ drawn from

$$p(z_3|z_1^{(\tau+1)}, z_2^{(\tau+1)}) \quad (11.48)$$

and so on, cycling through the three variables in turn.

Gibbs Sampling

1. Initialize $\{z_i : i = 1, \dots, M\}$
2. For $\tau = 1, \dots, T$:
 - Sample $z_1^{(\tau+1)} \sim p(z_1|z_2^{(\tau)}, z_3^{(\tau)}, \dots, z_M^{(\tau)})$.
 - Sample $z_2^{(\tau+1)} \sim p(z_2|z_1^{(\tau+1)}, z_3^{(\tau)}, \dots, z_M^{(\tau)})$.
 - \vdots
 - Sample $z_j^{(\tau+1)} \sim p(z_j|z_1^{(\tau+1)}, \dots, z_{j-1}^{(\tau+1)}, z_{j+1}^{(\tau)}, \dots, z_M^{(\tau)})$.
 - \vdots
 - Sample $z_M^{(\tau+1)} \sim p(z_M|z_1^{(\tau+1)}, z_2^{(\tau+1)}, \dots, z_{M-1}^{(\tau+1)})$.



Josiah Willard Gibbs
1839–1903

Gibbs spent almost his entire life living in a house built by his father in New Haven, Connecticut. In 1863, Gibbs was granted the first PhD in engineering in the United States, and in 1871 he was appointed to

the first chair of mathematical physics in the United States at Yale, a post for which he received no salary because at the time he had no publications. He developed the field of vector analysis and made contributions to crystallography and planetary orbits. His most famous work, entitled *On the Equilibrium of Heterogeneous Substances*, laid the foundations for the science of physical chemistry.

To show that this procedure samples from the required distribution, we first of all note that the distribution $p(\mathbf{z})$ is an invariant of each of the Gibbs sampling steps individually and hence of the whole Markov chain. This follows from the fact that when we sample from $p(z_i | \{\mathbf{z}_{\setminus i}\})$, the marginal distribution $p(\mathbf{z}_{\setminus i})$ is clearly invariant because the value of $\mathbf{z}_{\setminus i}$ is unchanged. Also, each step by definition samples from the correct conditional distribution $p(z_i | \mathbf{z}_{\setminus i})$. Because these conditional and marginal distributions together specify the joint distribution, we see that the joint distribution is itself invariant.

The second requirement to be satisfied in order that the Gibbs sampling procedure samples from the correct distribution is that it be ergodic. A sufficient condition for ergodicity is that none of the conditional distributions be anywhere zero. If this is the case, then any point in \mathbf{z} space can be reached from any other point in a finite number of steps involving one update of each of the component variables. If this requirement is not satisfied, so that some of the conditional distributions have zeros, then ergodicity, if it applies, must be proven explicitly.

The distribution of initial states must also be specified in order to complete the algorithm, although samples drawn after many iterations will effectively become independent of this distribution. Of course, successive samples from the Markov chain will be highly correlated, and so to obtain samples that are nearly independent it will be necessary to subsample the sequence.

We can obtain the Gibbs sampling procedure as a particular instance of the Metropolis-Hastings algorithm as follows. Consider a Metropolis-Hastings sampling step involving the variable z_k in which the remaining variables $\mathbf{z}_{\setminus k}$ remain fixed, and for which the transition probability from \mathbf{z} to \mathbf{z}^* is given by $q_k(\mathbf{z}^* | \mathbf{z}) = p(z_k^* | \mathbf{z}_{\setminus k})$. We note that $\mathbf{z}_{\setminus k}^* = \mathbf{z}_{\setminus k}$ because these components are unchanged by the sampling step. Also, $p(\mathbf{z}) = p(z_k | \mathbf{z}_{\setminus k})p(\mathbf{z}_{\setminus k})$. Thus the factor that determines the acceptance probability in the Metropolis-Hastings (11.44) is given by

$$A(\mathbf{z}^*, \mathbf{z}) = \frac{p(\mathbf{z}^*) q_k(\mathbf{z} | \mathbf{z}^*)}{p(\mathbf{z}) q_k(\mathbf{z}^* | \mathbf{z})} = \frac{p(z_k^* | \mathbf{z}_{\setminus k}^*) p(\mathbf{z}_{\setminus k}^*) p(z_k | \mathbf{z}_{\setminus k}^*)}{p(z_k | \mathbf{z}_{\setminus k}) p(\mathbf{z}_{\setminus k}) p(z_k^* | \mathbf{z}_{\setminus k})} = 1 \quad (11.49)$$

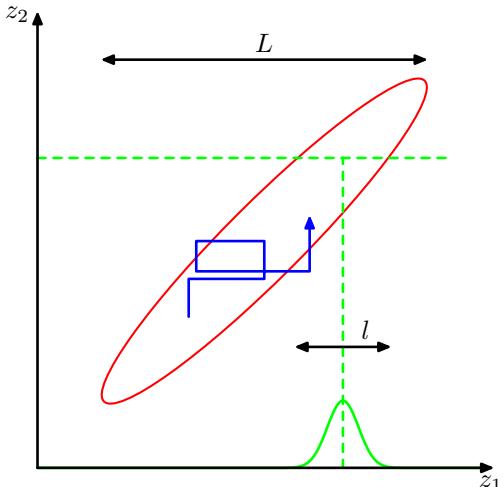
where we have used $\mathbf{z}_{\setminus k}^* = \mathbf{z}_{\setminus k}$. Thus the Metropolis-Hastings steps are always accepted.

As with the Metropolis algorithm, we can gain some insight into the behaviour of Gibbs sampling by investigating its application to a Gaussian distribution. Consider a correlated Gaussian in two variables, as illustrated in Figure 11.11, having conditional distributions of width l and marginal distributions of width L . The typical step size is governed by the conditional distributions and will be of order l . Because the state evolves according to a random walk, the number of steps needed to obtain independent samples from the distribution will be of order $(L/l)^2$. Of course if the Gaussian distribution were uncorrelated, then the Gibbs sampling procedure would be optimally efficient. For this simple problem, we could rotate the coordinate system in order to decorrelate the variables. However, in practical applications it will generally be infeasible to find such transformations.

One approach to reducing random walk behaviour in Gibbs sampling is called *over-relaxation* (Adler, 1981). In its original form, this applies to problems for which

Figure 11.11

Illustration of Gibbs sampling by alternate updates of two variables whose distribution is a correlated Gaussian. The step size is governed by the standard deviation of the conditional distribution (green curve), and is $O(l)$, leading to slow progress in the direction of elongation of the joint distribution (red ellipse). The number of steps needed to obtain an independent sample from the distribution is $O((L/l)^2)$.



the conditional distributions are Gaussian, which represents a more general class of distributions than the multivariate Gaussian because, for example, the non-Gaussian distribution $p(z, y) \propto \exp(-z^2 y^2)$ has Gaussian conditional distributions. At each step of the Gibbs sampling algorithm, the conditional distribution for a particular component z_i has some mean μ_i and some variance σ_i^2 . In the over-relaxation framework, the value of z_i is replaced with

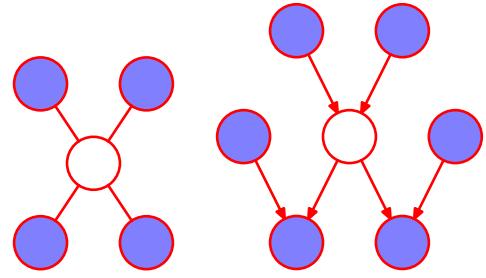
$$z'_i = \mu_i + \alpha(z_i - \mu_i) + \sigma_i(1 - \alpha_i^2)^{1/2}\nu \quad (11.50)$$

where ν is a Gaussian random variable with zero mean and unit variance, and α is a parameter such that $-1 < \alpha < 1$. For $\alpha = 0$, the method is equivalent to standard Gibbs sampling, and for $\alpha < 0$ the step is biased to the opposite side of the mean. This step leaves the desired distribution invariant because if z_i has mean μ_i and variance σ_i^2 , then so too does z'_i . The effect of over-relaxation is to encourage directed motion through state space when the variables are highly correlated. The framework of *ordered over-relaxation* (Neal, 1999) generalizes this approach to non-Gaussian distributions.

The practical applicability of Gibbs sampling depends on the ease with which samples can be drawn from the conditional distributions $p(z_k | \mathbf{z}_{\setminus k})$. In the case of probability distributions specified using graphical models, the conditional distributions for individual nodes depend only on the variables in the corresponding Markov blankets, as illustrated in Figure 11.12. For directed graphs, a wide choice of conditional distributions for the individual nodes conditioned on their parents will lead to conditional distributions for Gibbs sampling that are log concave. The adaptive rejection sampling methods discussed in Section 11.1.3 therefore provide a framework for Monte Carlo sampling from directed graphs with broad applicability.

If the graph is constructed using distributions from the exponential family, and if the parent-child relationships preserve conjugacy, then the full conditional distributions arising in Gibbs sampling will have the same functional form as the orig-

Figure 11.12 The Gibbs sampling method requires samples to be drawn from the conditional distribution of a variable conditioned on the remaining variables. For graphical models, this conditional distribution is a function only of the states of the nodes in the Markov blanket. For an undirected graph this comprises the set of neighbours, as shown on the left, while for a directed graph the Markov blanket comprises the parents, the children, and the co-parents, as shown on the right.



inal conditional distributions (conditioned on the parents) defining each node, and so standard sampling techniques can be employed. In general, the full conditional distributions will be of a complex form that does not permit the use of standard sampling algorithms. However, if these conditionals are log concave, then sampling can be done efficiently using adaptive rejection sampling (assuming the corresponding variable is a scalar).

If, at each stage of the Gibbs sampling algorithm, instead of drawing a sample from the corresponding conditional distribution, we make a point estimate of the variable given by the maximum of the conditional distribution, then we obtain the iterated conditional modes (ICM) algorithm discussed in Section 8.3.3. Thus ICM can be seen as a greedy approximation to Gibbs sampling.

Because the basic Gibbs sampling technique considers one variable at a time, there are strong dependencies between successive samples. At the opposite extreme, if we could draw samples directly from the joint distribution (an operation that we are supposing is intractable), then successive samples would be independent. We can hope to improve on the simple Gibbs sampler by adopting an intermediate strategy in which we sample successively from groups of variables rather than individual variables. This is achieved in the *blocking Gibbs* sampling algorithm by choosing blocks of variables, not necessarily disjoint, and then sampling jointly from the variables in each block in turn, conditioned on the remaining variables (Jensen *et al.*, 1995).

11.4. Slice Sampling

We have seen that one of the difficulties with the Metropolis algorithm is the sensitivity to step size. If this is too small, the result is slow decorrelation due to random walk behaviour, whereas if it is too large the result is inefficiency due to a high rejection rate. The technique of *slice sampling* (Neal, 2003) provides an adaptive step size that is automatically adjusted to match the characteristics of the distribution. Again it requires that we are able to evaluate the unnormalized distribution $\tilde{p}(\mathbf{z})$.

Consider first the univariate case. Slice sampling involves augmenting z with an additional variable u and then drawing samples from the joint (z, u) space. We shall see another example of this approach when we discuss hybrid Monte Carlo in Section 11.5. The goal is to sample uniformly from the area under the distribution

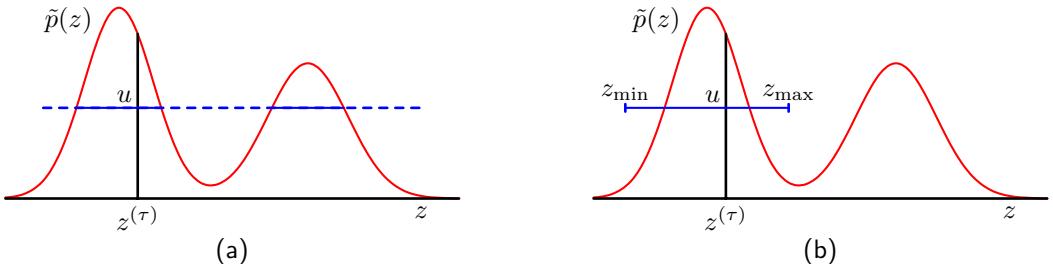


Figure 11.13 Illustration of slice sampling. (a) For a given value $z^{(\tau)}$, a value of u is chosen uniformly in the region $0 \leq u \leq \tilde{p}(z^{(\tau)})$, which then defines a ‘slice’ through the distribution, shown by the solid horizontal lines. (b) Because it is infeasible to sample directly from a slice, a new sample of z is drawn from a region $z_{\min} \leq z \leq z_{\max}$, which contains the previous value $z^{(\tau)}$.

given by

$$\hat{p}(z, u) = \begin{cases} 1/Z_p & \text{if } 0 \leq u \leq \tilde{p}(z) \\ 0 & \text{otherwise} \end{cases} \quad (11.51)$$

where $Z_p = \int \tilde{p}(z) dz$. The marginal distribution over z is given by

$$\int \hat{p}(z, u) du = \int_0^{\tilde{p}(z)} \frac{1}{Z_p} du = \frac{\tilde{p}(z)}{Z_p} = p(z) \quad (11.52)$$

and so we can sample from $p(z)$ by sampling from $\hat{p}(z, u)$ and then ignoring the u values. This can be achieved by alternately sampling z and u . Given the value of z we evaluate $\tilde{p}(z)$ and then sample u uniformly in the range $0 \leq u \leq \tilde{p}(z)$, which is straightforward. Then we fix u and sample z uniformly from the ‘slice’ through the distribution defined by $\{z : \tilde{p}(z) > u\}$. This is illustrated in Figure 11.13(a).

In practice, it can be difficult to sample directly from a slice through the distribution and so instead we define a sampling scheme that leaves the uniform distribution under $\hat{p}(z, u)$ invariant, which can be achieved by ensuring that detailed balance is satisfied. Suppose the current value of z is denoted $z^{(\tau)}$ and that we have obtained a corresponding sample u . The next value of z is obtained by considering a region $z_{\min} \leq z \leq z_{\max}$ that contains $z^{(\tau)}$. It is in the choice of this region that the adaptation to the characteristic length scales of the distribution takes place. We want the region to encompass as much of the slice as possible so as to allow large moves in z space while having as little as possible of this region lying outside the slice, because this makes the sampling less efficient.

One approach to the choice of region involves starting with a region containing $z^{(\tau)}$ having some width w and then testing each of the end points to see if they lie within the slice. If either end point does not, then the region is extended in that direction by increments of value w until the end point lies outside the region. A candidate value z' is then chosen uniformly from this region, and if it lies within the slice, then it forms $z^{(\tau+1)}$. If it lies outside the slice, then the region is shrunk such that z' forms an end point and such that the region still contains $z^{(\tau)}$. Then another

candidate point is drawn uniformly from this reduced region and so on, until a value of z is found that lies within the slice.

Slice sampling can be applied to multivariate distributions by repeatedly sampling each variable in turn, in the manner of Gibbs sampling. This requires that we are able to compute, for each component z_i , a function that is proportional to $p(z_i | \mathbf{z}_{\setminus i})$.

11.5. The Hybrid Monte Carlo Algorithm

As we have already noted, one of the major limitations of the Metropolis algorithm is that it can exhibit random walk behaviour whereby the distance traversed through the state space grows only as the square root of the number of steps. The problem cannot be resolved simply by taking bigger steps as this leads to a high rejection rate.

In this section, we introduce a more sophisticated class of transitions based on an analogy with physical systems and that has the property of being able to make large changes to the system state while keeping the rejection probability small. It is applicable to distributions over continuous variables for which we can readily evaluate the gradient of the log probability with respect to the state variables. We will discuss the dynamical systems framework in Section 11.5.1, and then in Section 11.5.2 we explain how this may be combined with the Metropolis algorithm to yield the powerful hybrid Monte Carlo algorithm. A background in physics is not required as this section is self-contained and the key results are all derived from first principles.

11.5.1 Dynamical systems

The dynamical approach to stochastic sampling has its origins in algorithms for simulating the behaviour of physical systems evolving under Hamiltonian dynamics. In a Markov chain Monte Carlo simulation, the goal is to sample from a given probability distribution $p(\mathbf{z})$. The framework of *Hamiltonian dynamics* is exploited by casting the probabilistic simulation in the form of a Hamiltonian system. In order to remain in keeping with the literature in this area, we make use of the relevant dynamical systems terminology where appropriate, which will be defined as we go along.

The dynamics that we consider corresponds to the evolution of the state variable $\mathbf{z} = \{z_i\}$ under continuous time, which we denote by τ . Classical dynamics is described by Newton's second law of motion in which the acceleration of an object is proportional to the applied force, corresponding to a second-order differential equation over time. We can decompose a second-order equation into two coupled first-order equations by introducing intermediate *momentum* variables \mathbf{r} , corresponding to the rate of change of the state variables \mathbf{z} , having components

$$r_i = \frac{dz_i}{d\tau} \quad (11.53)$$

where the z_i can be regarded as *position* variables in this dynamics perspective. Thus

for each position variable there is a corresponding momentum variable, and the joint space of position and momentum variables is called *phase space*.

Without loss of generality, we can write the probability distribution $p(\mathbf{z})$ in the form

$$p(\mathbf{z}) = \frac{1}{Z_p} \exp(-E(\mathbf{z})) \quad (11.54)$$

where $E(\mathbf{z})$ is interpreted as the *potential energy* of the system when in state \mathbf{z} . The system acceleration is the rate of change of momentum and is given by the applied *force*, which itself is the negative gradient of the potential energy

$$\frac{d\mathbf{r}_i}{d\tau} = -\frac{\partial E(\mathbf{z})}{\partial z_i}. \quad (11.55)$$

It is convenient to reformulate this dynamical system using the Hamiltonian framework. To do this, we first define the *kinetic energy* by

$$K(\mathbf{r}) = \frac{1}{2}\|\mathbf{r}\|^2 = \frac{1}{2}\sum_i r_i^2. \quad (11.56)$$

The total energy of the system is then the sum of its potential and kinetic energies

$$H(\mathbf{z}, \mathbf{r}) = E(\mathbf{z}) + K(\mathbf{r}) \quad (11.57)$$

where H is the *Hamiltonian* function. Using (11.53), (11.55), (11.56), and (11.57), we can now express the dynamics of the system in terms of the Hamiltonian equations given by

$$\frac{dz_i}{d\tau} = \frac{\partial H}{\partial r_i} \quad (11.58)$$

$$\frac{dr_i}{d\tau} = -\frac{\partial H}{\partial z_i}. \quad (11.59)$$

Exercise 11.15

William Hamilton 1805–1865

William Rowan Hamilton was an Irish mathematician and physicist, and child prodigy, who was appointed Professor of Astronomy at Trinity College, Dublin, in 1827, before he had even graduated. One of Hamilton's most important contributions was a new formulation of dynamics, which played a significant role in the later development of quantum mechanics.



His other great achievement was the development of *quaternions*, which generalize the concept of complex numbers by introducing three distinct square roots of minus one, which satisfy $i^2 = j^2 = k^2 = ijk = -1$. It is said that these equations occurred to him while walking along the Royal Canal in Dublin with his wife, on 16 October 1843, and he promptly carved the equations into the side of Broome bridge. Although there is no longer any evidence of the carving, there is now a stone plaque on the bridge commemorating the discovery and displaying the quaternion equations.

During the evolution of this dynamical system, the value of the Hamiltonian H is constant, as is easily seen by differentiation

$$\begin{aligned}\frac{dH}{d\tau} &= \sum_i \left\{ \frac{\partial H}{\partial z_i} \frac{dz_i}{d\tau} + \frac{\partial H}{\partial r_i} \frac{dr_i}{d\tau} \right\} \\ &= \sum_i \left\{ \frac{\partial H}{\partial z_i} \frac{\partial H}{\partial r_i} - \frac{\partial H}{\partial r_i} \frac{\partial H}{\partial z_i} \right\} = 0.\end{aligned}\quad (11.60)$$

A second important property of Hamiltonian dynamical systems, known as *Liouville's Theorem*, is that they preserve volume in phase space. In other words, if we consider a region within the space of variables (\mathbf{z}, \mathbf{r}) , then as this region evolves under the equations of Hamiltonian dynamics, its shape may change but its volume will not. This can be seen by noting that the flow field (rate of change of location in phase space) is given by

$$\mathbf{V} = \left(\frac{d\mathbf{z}}{d\tau}, \frac{d\mathbf{r}}{d\tau} \right) \quad (11.61)$$

and that the divergence of this field vanishes

$$\begin{aligned}\text{div } \mathbf{V} &= \sum_i \left\{ \frac{\partial}{\partial z_i} \frac{dz_i}{d\tau} + \frac{\partial}{\partial r_i} \frac{dr_i}{d\tau} \right\} \\ &= \sum_i \left\{ -\frac{\partial}{\partial z_i} \frac{\partial H}{\partial r_i} + \frac{\partial}{\partial r_i} \frac{\partial H}{\partial z_i} \right\} = 0.\end{aligned}\quad (11.62)$$

Now consider the joint distribution over phase space whose total energy is the Hamiltonian, i.e., the distribution given by

$$p(\mathbf{z}, \mathbf{r}) = \frac{1}{Z_H} \exp(-H(\mathbf{z}, \mathbf{r})). \quad (11.63)$$

Using the two results of conservation of volume and conservation of H , it follows that the Hamiltonian dynamics will leave $p(\mathbf{z}, \mathbf{r})$ invariant. This can be seen by considering a small region of phase space over which H is approximately constant. If we follow the evolution of the Hamiltonian equations for a finite time, then the volume of this region will remain unchanged as will the value of H in this region, and hence the probability density, which is a function only of H , will also be unchanged.

Although H is invariant, the values of \mathbf{z} and \mathbf{r} will vary, and so by integrating the Hamiltonian dynamics over a finite time duration it becomes possible to make large changes to \mathbf{z} in a systematic way that avoids random walk behaviour.

Evolution under the Hamiltonian dynamics will not, however, sample ergodically from $p(\mathbf{z}, \mathbf{r})$ because the value of H is constant. In order to arrive at an ergodic sampling scheme, we can introduce additional moves in phase space that change the value of H while also leaving the distribution $p(\mathbf{z}, \mathbf{r})$ invariant. The simplest way to achieve this is to replace the value of \mathbf{r} with one drawn from its distribution conditioned on \mathbf{z} . This can be regarded as a Gibbs sampling step, and hence from

Section 11.3 we see that this also leaves the desired distribution invariant. Noting that \mathbf{z} and \mathbf{r} are independent in the distribution $p(\mathbf{z}, \mathbf{r})$, we see that the conditional distribution $p(\mathbf{r}|\mathbf{z})$ is a Gaussian from which it is straightforward to sample.

In a practical application of this approach, we have to address the problem of performing a numerical integration of the Hamiltonian equations. This will necessarily introduce numerical errors and so we should devise a scheme that minimizes the impact of such errors. In fact, it turns out that integration schemes can be devised for which Liouville's theorem still holds exactly. This property will be important in the hybrid Monte Carlo algorithm, which is discussed in Section 11.5.2. One scheme for achieving this is called the *leapfrog* discretization and involves alternately updating discrete-time approximations $\hat{\mathbf{z}}$ and $\hat{\mathbf{r}}$ to the position and momentum variables using

$$\hat{r}_i(\tau + \epsilon/2) = \hat{r}_i(\tau) - \frac{\epsilon}{2} \frac{\partial E}{\partial z_i}(\hat{\mathbf{z}}(\tau)) \quad (11.64)$$

$$\hat{z}_i(\tau + \epsilon) = \hat{z}_i(\tau) + \epsilon \hat{r}_i(\tau + \epsilon/2) \quad (11.65)$$

$$\hat{r}_i(\tau + \epsilon) = \hat{r}_i(\tau + \epsilon/2) - \frac{\epsilon}{2} \frac{\partial E}{\partial z_i}(\hat{\mathbf{z}}(\tau + \epsilon)). \quad (11.66)$$

We see that this takes the form of a half-step update of the momentum variables with step size $\epsilon/2$, followed by a full-step update of the position variables with step size ϵ , followed by a second half-step update of the momentum variables. If several leapfrog steps are applied in succession, it can be seen that half-step updates to the momentum variables can be combined into full-step updates with step size ϵ . The successive updates to position and momentum variables then leapfrog over each other. In order to advance the dynamics by a time interval τ , we need to take τ/ϵ steps. The error involved in the discretized approximation to the continuous time dynamics will go to zero, assuming a smooth function $E(\mathbf{z})$, in the limit $\epsilon \rightarrow 0$. However, for a nonzero ϵ as used in practice, some residual error will remain. We shall see in Section 11.5.2 how the effects of such errors can be eliminated in the hybrid Monte Carlo algorithm.

In summary then, the Hamiltonian dynamical approach involves alternating between a series of leapfrog updates and a resampling of the momentum variables from their marginal distribution.

Note that the Hamiltonian dynamics method, unlike the basic Metropolis algorithm, is able to make use of information about the gradient of the log probability distribution as well as about the distribution itself. An analogous situation is familiar from the domain of function optimization. In most cases where gradient information is available, it is highly advantageous to make use of it. Informally, this follows from the fact that in a space of dimension D , the additional computational cost of evaluating a gradient compared with evaluating the function itself will typically be a fixed factor independent of D , whereas the D -dimensional gradient vector conveys D pieces of information compared with the one piece of information given by the function itself.

Exercise 11.16

11.5.2 Hybrid Monte Carlo

As we discussed in the previous section, for a nonzero step size ϵ , the discretization of the leapfrog algorithm will introduce errors into the integration of the Hamiltonian dynamical equations. *Hybrid Monte Carlo* (Duane *et al.*, 1987; Neal, 1996) combines Hamiltonian dynamics with the Metropolis algorithm and thereby removes any bias associated with the discretization.

Specifically, the algorithm uses a Markov chain consisting of alternate stochastic updates of the momentum variable \mathbf{r} and Hamiltonian dynamical updates using the leapfrog algorithm. After each application of the leapfrog algorithm, the resulting candidate state is accepted or rejected according to the Metropolis criterion based on the value of the Hamiltonian H . Thus if (\mathbf{z}, \mathbf{r}) is the initial state and $(\mathbf{z}^*, \mathbf{r}^*)$ is the state after the leapfrog integration, then this candidate state is accepted with probability

$$\min(1, \exp\{H(\mathbf{z}, \mathbf{r}) - H(\mathbf{z}^*, \mathbf{r}^*)\}). \quad (11.67)$$

If the leapfrog integration were to simulate the Hamiltonian dynamics perfectly, then every such candidate step would automatically be accepted because the value of H would be unchanged. Due to numerical errors, the value of H may sometimes decrease, and we would like the Metropolis criterion to remove any bias due to this effect and ensure that the resulting samples are indeed drawn from the required distribution. In order for this to be the case, we need to ensure that the update equations corresponding to the leapfrog integration satisfy detailed balance (11.40). This is easily achieved by modifying the leapfrog scheme as follows.

Before the start of each leapfrog integration sequence, we choose at random, with equal probability, whether to integrate forwards in time (using step size ϵ) or backwards in time (using step size $-\epsilon$). We first note that the leapfrog integration scheme (11.64), (11.65), and (11.66) is time-reversible, so that integration for L steps using step size $-\epsilon$ will exactly undo the effect of integration for L steps using step size ϵ . Next we show that the leapfrog integration preserves phase-space volume exactly. This follows from the fact that each step in the leapfrog scheme updates either a z_i variable or an r_i variable by an amount that is a function only of the other variable. As shown in Figure 11.14, this has the effect of shearing a region of phase space while not altering its volume.

Finally, we use these results to show that detailed balance holds. Consider a small region \mathcal{R} of phase space that, under a sequence of L leapfrog iterations of step size ϵ , maps to a region \mathcal{R}' . Using conservation of volume under the leapfrog iteration, we see that if \mathcal{R} has volume δV then so too will \mathcal{R}' . If we choose an initial point from the distribution (11.63) and then update it using L leapfrog interactions, the probability of the transition going from \mathcal{R} to \mathcal{R}' is given by

$$\frac{1}{Z_H} \exp(-H(\mathcal{R})) \delta V \frac{1}{2} \min\{1, \exp(-H(\mathcal{R}) + H(\mathcal{R}'))\}. \quad (11.68)$$

where the factor of $1/2$ arises from the probability of choosing to integrate with a positive step size rather than a negative one. Similarly, the probability of starting in

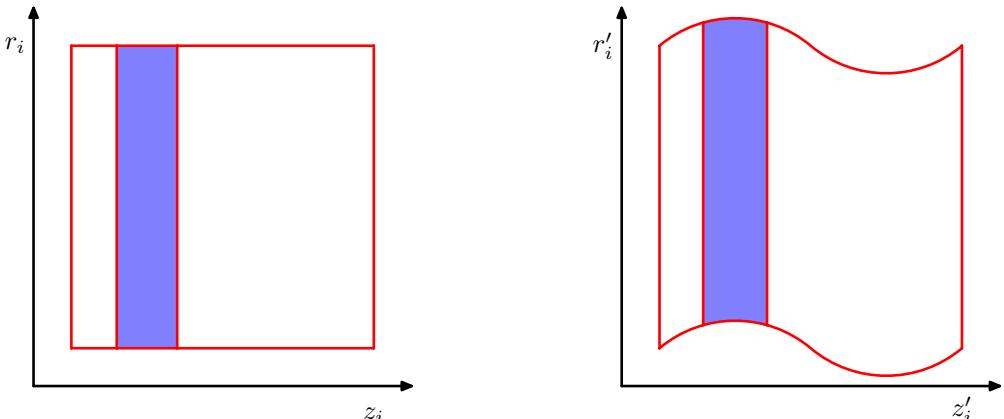


Figure 11.14 Each step of the leapfrog algorithm (11.64)–(11.66) modifies either a position variable z_i or a momentum variable r_i . Because the change to one variable is a function only of the other, any region in phase space will be sheared without change of volume.

region \mathcal{R}' and integrating backwards in time to end up in region \mathcal{R} is given by

$$\frac{1}{Z_H} \exp(-H(\mathcal{R}')) \delta V \frac{1}{2} \min \{1, \exp(-H(\mathcal{R}') + H(\mathcal{R}))\}. \quad (11.69)$$

It is easily seen that the two probabilities (11.68) and (11.69) are equal, and hence detailed balance holds. Note that this proof ignores any overlap between the regions \mathcal{R} and \mathcal{R}' but is easily generalized to allow for such overlap.

It is not difficult to construct examples for which the leapfrog algorithm returns to its starting position after a finite number of iterations. In such cases, the random replacement of the momentum values before each leapfrog integration will not be sufficient to ensure ergodicity because the position variables will never be updated. Such phenomena are easily avoided by choosing the magnitude of the step size at random from some small interval, before each leapfrog integration.

We can gain some insight into the behaviour of the hybrid Monte Carlo algorithm by considering its application to a multivariate Gaussian. For convenience, consider a Gaussian distribution $p(\mathbf{z})$ with independent components, for which the Hamiltonian is given by

$$H(\mathbf{z}, \mathbf{r}) = \frac{1}{2} \sum_i \frac{1}{\sigma_i^2} z_i^2 + \frac{1}{2} \sum_i r_i^2. \quad (11.70)$$

Our conclusions will be equally valid for a Gaussian distribution having correlated components because the hybrid Monte Carlo algorithm exhibits rotational isotropy. During the leapfrog integration, each pair of phase-space variables z_i, r_i evolves independently. However, the acceptance or rejection of the candidate point is based on the value of H , which depends on the values of all of the variables. Thus, a significant integration error in any one of the variables could lead to a high probability of rejection. In order that the discrete leapfrog integration be a reasonably

Exercise 11.17

good approximation to the true continuous-time dynamics, it is necessary for the leapfrog integration scale ϵ to be smaller than the shortest length-scale over which the potential is varying significantly. This is governed by the smallest value of σ_i , which we denote by σ_{\min} . Recall that the goal of the leapfrog integration in hybrid Monte Carlo is to move a substantial distance through phase space to a new state that is relatively independent of the initial state and still achieve a high probability of acceptance. In order to achieve this, the leapfrog integration must be continued for a number of iterations of order $\sigma_{\max}/\sigma_{\min}$.

By contrast, consider the behaviour of a simple Metropolis algorithm with an isotropic Gaussian proposal distribution of variance s^2 , considered earlier. In order to avoid high rejection rates, the value of s must be of order σ_{\min} . The exploration of state space then proceeds by a random walk and takes of order $(\sigma_{\max}/\sigma_{\min})^2$ steps to arrive at a roughly independent state.

11.6. Estimating the Partition Function

As we have seen, most of the sampling algorithms considered in this chapter require only the functional form of the probability distribution up to a multiplicative constant. Thus if we write

$$p_E(\mathbf{z}) = \frac{1}{Z_E} \exp(-E(\mathbf{z})) \quad (11.71)$$

then the value of the normalization constant Z_E , also known as the partition function, is not needed in order to draw samples from $p(\mathbf{z})$. However, knowledge of the value of Z_E can be useful for Bayesian model comparison since it represents the model evidence (i.e., the probability of the observed data given the model), and so it is of interest to consider how its value might be obtained. We assume that direct evaluation by summing, or integrating, the function $\exp(-E(\mathbf{z}))$ over the state space of \mathbf{z} is intractable.

For model comparison, it is actually the ratio of the partition functions for two models that is required. Multiplication of this ratio by the ratio of prior probabilities gives the ratio of posterior probabilities, which can then be used for model selection or model averaging.

One way to estimate a ratio of partition functions is to use importance sampling from a distribution with energy function $G(\mathbf{z})$

$$\begin{aligned} \frac{Z_E}{Z_G} &= \frac{\sum_{\mathbf{z}} \exp(-E(\mathbf{z}))}{\sum_{\mathbf{z}} \exp(-G(\mathbf{z}))} \\ &= \frac{\sum_{\mathbf{z}} \exp(-E(\mathbf{z}) + G(\mathbf{z})) \exp(-G(\mathbf{z}))}{\sum_{\mathbf{z}} \exp(-G(\mathbf{z}))} \\ &= \mathbb{E}_{G(\mathbf{z})}[\exp(-E + G)] \\ &\simeq \sum_l \exp(-E(\mathbf{z}^{(l)}) + G(\mathbf{z}^{(l)})) \end{aligned} \quad (11.72)$$

where $\{\mathbf{z}^{(l)}\}$ are samples drawn from the distribution defined by $p_G(\mathbf{z})$. If the distribution p_G is one for which the partition function can be evaluated analytically, for example a Gaussian, then the absolute value of Z_E can be obtained.

This approach will only yield accurate results if the importance sampling distribution p_G is closely matched to the distribution p_E , so that the ratio p_E/p_G does not have wide variations. In practice, suitable analytically specified importance sampling distributions cannot readily be found for the kinds of complex models considered in this book.

An alternative approach is therefore to use the samples obtained from a Markov chain to define the importance-sampling distribution. If the transition probability for the Markov chain is given by $T(\mathbf{z}, \mathbf{z}')$, and the sample set is given by $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}$, then the sampling distribution can be written as

$$\frac{1}{Z_G} \exp(-G(\mathbf{z})) = \sum_{l=1}^L T(\mathbf{z}^{(l)}, \mathbf{z}) \quad (11.73)$$

which can be used directly in (11.72).

Methods for estimating the ratio of two partition functions require for their success that the two corresponding distributions be reasonably closely matched. This is especially problematic if we wish to find the absolute value of the partition function for a complex distribution because it is only for relatively simple distributions that the partition function can be evaluated directly, and so attempting to estimate the ratio of partition functions directly is unlikely to be successful. This problem can be tackled using a technique known as *chaining* (Neal, 1993; Barber and Bishop, 1997), which involves introducing a succession of intermediate distributions p_2, \dots, p_{M-1} that interpolate between a simple distribution $p_1(\mathbf{z})$ for which we can evaluate the normalization coefficient Z_1 and the desired complex distribution $p_M(\mathbf{z})$. We then have

$$\frac{Z_M}{Z_1} = \frac{Z_2}{Z_1} \frac{Z_3}{Z_2} \cdots \frac{Z_M}{Z_{M-1}} \quad (11.74)$$

in which the intermediate ratios can be determined using Monte Carlo methods as discussed above. One way to construct such a sequence of intermediate systems is to use an energy function containing a continuous parameter $0 \leq \alpha \leq 1$ that interpolates between the two distributions

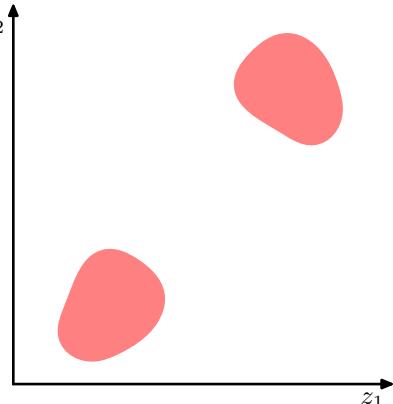
$$E_\alpha(\mathbf{z}) = (1 - \alpha)E_1(\mathbf{z}) + \alpha E_M(\mathbf{z}). \quad (11.75)$$

If the intermediate ratios in (11.74) are to be found using Monte Carlo, it may be more efficient to use a single Markov chain run than to restart the Markov chain for each ratio. In this case, the Markov chain is run initially for the system p_1 and then after some suitable number of steps moves on to the next distribution in the sequence. Note, however, that the system must remain close to the equilibrium distribution at each stage.

Exercises

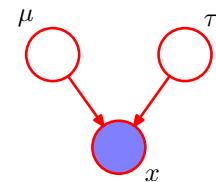
- 11.1** (*) **WWW** Show that the finite sample estimator \hat{f} defined by (11.2) has mean equal to $\mathbb{E}[f]$ and variance given by (11.3).
- 11.2** (*) Suppose that z is a random variable with uniform distribution over $(0, 1)$ and that we transform z using $y = h^{-1}(z)$ where $h(y)$ is given by (11.6). Show that y has the distribution $p(y)$.
- 11.3** (*) Given a random variable z that is uniformly distributed over $(0, 1)$, find a transformation $y = f(z)$ such that y has a Cauchy distribution given by (11.8).
- 11.4** (**) Suppose that z_1 and z_2 are uniformly distributed over the unit circle, as shown in Figure 11.3, and that we make the change of variables given by (11.10) and (11.11). Show that (y_1, y_2) will be distributed according to (11.12).
- 11.5** (*) **WWW** Let \mathbf{z} be a D -dimensional random variable having a Gaussian distribution with zero mean and unit covariance matrix, and suppose that the positive definite symmetric matrix Σ has the Cholesky decomposition $\Sigma = \mathbf{L}\mathbf{L}^T$ where \mathbf{L} is a lower-triangular matrix (i.e., one with zeros above the leading diagonal). Show that the variable $\mathbf{y} = \boldsymbol{\mu} + \mathbf{L}\mathbf{z}$ has a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance Σ . This provides a technique for generating samples from a general multivariate Gaussian using samples from a univariate Gaussian having zero mean and unit variance.
- 11.6** (**) **WWW** In this exercise, we show more carefully that rejection sampling does indeed draw samples from the desired distribution $p(\mathbf{z})$. Suppose the proposal distribution is $q(\mathbf{z})$ and show that the probability of a sample value \mathbf{z} being accepted is given by $\tilde{p}(\mathbf{z})/kq(\mathbf{z})$ where \tilde{p} is any unnormalized distribution that is proportional to $p(\mathbf{z})$, and the constant k is set to the smallest value that ensures $kq(\mathbf{z}) \geq \tilde{p}(\mathbf{z})$ for all values of \mathbf{z} . Note that the probability of drawing a value \mathbf{z} is given by the probability of drawing that value from $q(\mathbf{z})$ times the probability of accepting that value given that it has been drawn. Make use of this, along with the sum and product rules of probability, to write down the normalized form for the distribution over \mathbf{z} , and show that it equals $p(\mathbf{z})$.
- 11.7** (*) Suppose that z has a uniform distribution over the interval $[0, 1]$. Show that the variable $y = b \tan z + c$ has a Cauchy distribution given by (11.16).
- 11.8** (**) Determine expressions for the coefficients k_i in the envelope distribution (11.17) for adaptive rejection sampling using the requirements of continuity and normalization.
- 11.9** (**) By making use of the technique discussed in Section 11.1.1 for sampling from a single exponential distribution, devise an algorithm for sampling from the piecewise exponential distribution defined by (11.17).
- 11.10** (*) Show that the simple random walk over the integers defined by (11.34), (11.35), and (11.36) has the property that $\mathbb{E}[(z^{(\tau)})^2] = \mathbb{E}[(z^{(\tau-1)})^2] + 1/2$ and hence by induction that $\mathbb{E}[(z^{(\tau)})^2] = \tau/2$.

Figure 11.15 A probability distribution over two variables z_1 and z_2 that is uniform over the shaded regions and that is zero everywhere else.



- 11.11** (**) **www** Show that the Gibbs sampling algorithm, discussed in Section 11.3, satisfies detailed balance as defined by (11.40).
- 11.12** (*) Consider the distribution shown in Figure 11.15. Discuss whether the standard Gibbs sampling procedure for this distribution is ergodic, and therefore whether it would sample correctly from this distribution
- 11.13** (**) Consider the simple 3-node graph shown in Figure 11.16 in which the observed node x is given by a Gaussian distribution $\mathcal{N}(x|\mu, \tau^{-1})$ with mean μ and precision τ . Suppose that the marginal distributions over the mean and precision are given by $\mathcal{N}(\mu|\mu_0, s_0)$ and $\text{Gam}(\tau|a, b)$, where $\text{Gam}(\cdot|a, b)$ denotes a gamma distribution. Write down expressions for the conditional distributions $p(\mu|x, \tau)$ and $p(\tau|x, \mu)$ that would be required in order to apply Gibbs sampling to the posterior distribution $p(\mu, \tau|x)$.
- 11.14** (*) Verify that the over-relaxation update (11.50), in which z_i has mean μ_i and variance σ_i^2 , and where ν has zero mean and unit variance, gives a value z'_i with mean μ_i and variance σ_i^2 .
- 11.15** (*) **www** Using (11.56) and (11.57), show that the Hamiltonian equation (11.58) is equivalent to (11.53). Similarly, using (11.57) show that (11.59) is equivalent to (11.55).
- 11.16** (*) By making use of (11.56), (11.57), and (11.63), show that the conditional distribution $p(\mathbf{r}|z)$ is a Gaussian.

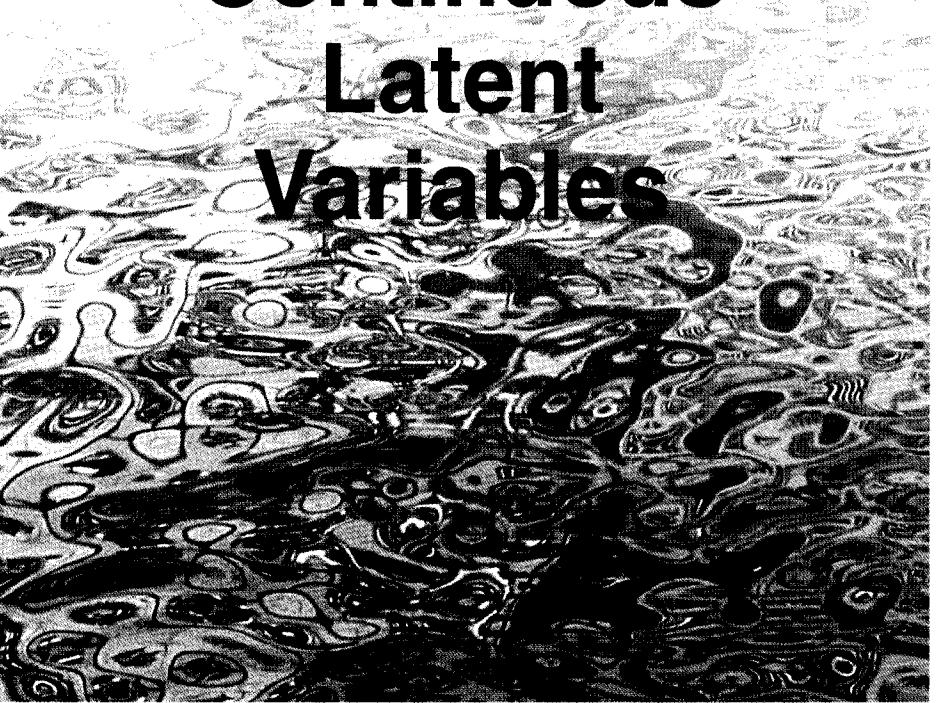
Figure 11.16 A graph involving an observed Gaussian variable x with prior distributions over its mean μ and precision τ .



11.17 (*) **www** Verify that the two probabilities (11.68) and (11.69) are equal, and hence that detailed balance holds for the hybrid Monte Carlo algorithm.

12

Continuous Latent Variables



Appendix A

In Chapter 9, we discussed probabilistic models having discrete latent variables, such as the mixture of Gaussians. We now explore models in which some, or all, of the latent variables are continuous. An important motivation for such models is that many data sets have the property that the data points all lie close to a manifold of much lower dimensionality than that of the original data space. To see why this might arise, consider an artificial data set constructed by taking one of the off-line digits, represented by a 64×64 pixel grey-level image, and embedding it in a larger image of size 100×100 by padding with pixels having the value zero (corresponding to white pixels) in which the location and orientation of the digit is varied at random, as illustrated in Figure 12.1. Each of the resulting images is represented by a point in the $100 \times 100 = 10,000$ -dimensional data space. However, across a data set of such images, there are only three *degrees of freedom* of variability, corresponding to the vertical and horizontal translations and the rotations. The data points will therefore live on a subspace of the data space whose *intrinsic dimensionality* is three. Note

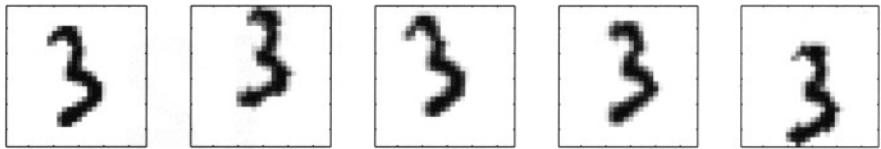


Figure 12.1 A synthetic data set obtained by taking one of the off-line digit images and creating multiple copies in each of which the digit has undergone a random displacement and rotation within some larger image field. The resulting images each have $100 \times 100 = 10,000$ pixels.

that the manifold will be nonlinear because, for instance, if we translate the digit past a particular pixel, that pixel value will go from zero (white) to one (black) and back to zero again, which is clearly a nonlinear function of the digit position. In this example, the translation and rotation parameters are latent variables because we observe only the image vectors and are not told which values of the translation or rotation variables were used to create them.

For real digit image data, there will be a further degree of freedom arising from scaling. Moreover there will be multiple additional degrees of freedom associated with more complex deformations due to the variability in an individual's writing as well as the differences in writing styles between individuals. Nevertheless, the number of such degrees of freedom will be small compared to the dimensionality of the data set.

Appendix A

Another example is provided by the oil flow data set, in which (for a given geometrical configuration of the gas, water, and oil phases) there are only two degrees of freedom of variability corresponding to the fraction of oil in the pipe and the fraction of water (the fraction of gas then being determined). Although the data space comprises 12 measurements, a data set of points will lie close to a two-dimensional manifold embedded within this space. In this case, the manifold comprises several distinct segments corresponding to different flow regimes, each such segment being a (noisy) continuous two-dimensional manifold. If our goal is data compression, or density modelling, then there can be benefits in exploiting this manifold structure.

In practice, the data points will not be confined precisely to a smooth low-dimensional manifold, and we can interpret the departures of data points from the manifold as ‘noise’. This leads naturally to a generative view of such models in which we first select a point within the manifold according to some latent variable distribution and then generate an observed data point by adding noise, drawn from some conditional distribution of the data variables given the latent variables.

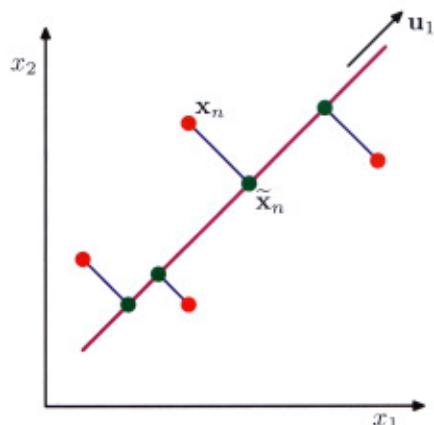
Section 8.1.4

The simplest continuous latent variable model assumes Gaussian distributions for both the latent and observed variables and makes use of a linear-Gaussian dependence of the observed variables on the state of the latent variables. This leads to a probabilistic formulation of the well-known technique of principal component analysis (PCA), as well as to a related model called factor analysis.

Section 12.1

In this chapter we will begin with a standard, nonprobabilistic treatment of PCA, and then we show how PCA arises naturally as the maximum likelihood solution to

Figure 12.2 Principal component analysis seeks a space of lower dimensionality, known as the principal subspace and denoted by the magenta line, such that the orthogonal projection of the data points (red dots) onto this subspace maximizes the variance of the projected points (green dots). An alternative definition of PCA is based on minimizing the sum-of-squares of the projection errors, indicated by the blue lines.



Section 12.2

a particular form of linear-Gaussian latent variable model. This probabilistic reformulation brings many advantages, such as the use of EM for parameter estimation, principled extensions to mixtures of PCA models, and Bayesian formulations that allow the number of principal components to be determined automatically from the data. Finally, we discuss briefly several generalizations of the latent variable concept that go beyond the linear-Gaussian assumption including non-Gaussian latent variables, which leads to the framework of *independent component analysis*, as well as models having a nonlinear relationship between latent and observed variables.

Section 12.4

12.1. Principal Component Analysis

Principal component analysis, or PCA, is a technique that is widely used for applications such as dimensionality reduction, lossy data compression, feature extraction, and data visualization (Jolliffe, 2002). It is also known as the *Karhunen-Loëve* transform.

There are two commonly used definitions of PCA that give rise to the same algorithm. PCA can be defined as the orthogonal projection of the data onto a lower dimensional linear space, known as the *principal subspace*, such that the variance of the projected data is maximized (Hotelling, 1933). Equivalently, it can be defined as the linear projection that minimizes the average projection cost, defined as the mean squared distance between the data points and their projections (Pearson, 1901). The process of orthogonal projection is illustrated in Figure 12.2. We consider each of these definitions in turn.

12.1.1 Maximum variance formulation

Consider a data set of observations $\{\mathbf{x}_n\}$ where $n = 1, \dots, N$, and \mathbf{x}_n is a Euclidean variable with dimensionality D . Our goal is to project the data onto a space having dimensionality $M < D$ while maximizing the variance of the projected data. For the moment, we shall assume that the value of M is given. Later in this

chapter, we shall consider techniques to determine an appropriate value of M from the data.

To begin with, consider the projection onto a one-dimensional space ($M = 1$). We can define the direction of this space using a D -dimensional vector \mathbf{u}_1 , which for convenience (and without loss of generality) we shall choose to be a unit vector so that $\mathbf{u}_1^T \mathbf{u}_1 = 1$ (note that we are only interested in the direction defined by \mathbf{u}_1 , not in the magnitude of \mathbf{u}_1 itself). Each data point \mathbf{x}_n is then projected onto a scalar value $\mathbf{u}_1^T \mathbf{x}_n$. The mean of the projected data is $\mathbf{u}_1^T \bar{\mathbf{x}}$ where $\bar{\mathbf{x}}$ is the sample set mean given by

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \quad (12.1)$$

and the variance of the projected data is given by

$$\frac{1}{N} \sum_{n=1}^N \{ \mathbf{u}_1^T \mathbf{x}_n - \mathbf{u}_1^T \bar{\mathbf{x}} \}^2 = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 \quad (12.2)$$

where \mathbf{S} is the data covariance matrix defined by

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T. \quad (12.3)$$

We now maximize the projected variance $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$ with respect to \mathbf{u}_1 . Clearly, this has to be a constrained maximization to prevent $\|\mathbf{u}_1\| \rightarrow \infty$. The appropriate constraint comes from the normalization condition $\mathbf{u}_1^T \mathbf{u}_1 = 1$. To enforce this constraint, we introduce a Lagrange multiplier that we shall denote by λ_1 , and then make an unconstrained maximization of

$$\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1). \quad (12.4)$$

By setting the derivative with respect to \mathbf{u}_1 equal to zero, we see that this quantity will have a stationary point when

$$\mathbf{S} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1 \quad (12.5)$$

which says that \mathbf{u}_1 must be an eigenvector of \mathbf{S} . If we left-multiply by \mathbf{u}_1^T and make use of $\mathbf{u}_1^T \mathbf{u}_1 = 1$, we see that the variance is given by

$$\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \lambda_1 \quad (12.6)$$

and so the variance will be a maximum when we set \mathbf{u}_1 equal to the eigenvector having the largest eigenvalue λ_1 . This eigenvector is known as the first principal component.

We can define additional principal components in an incremental fashion by choosing each new direction to be that which maximizes the projected variance

amongst all possible directions orthogonal to those already considered. If we consider the general case of an M -dimensional projection space, the optimal linear projection for which the variance of the projected data is maximized is now defined by the M eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_M$ of the data covariance matrix \mathbf{S} corresponding to the M largest eigenvalues $\lambda_1, \dots, \lambda_M$. This is easily shown using proof by induction.

Exercise 12.1

Section 12.2.2

Appendix C

To summarize, principal component analysis involves evaluating the mean $\bar{\mathbf{x}}$ and the covariance matrix \mathbf{S} of the data set and then finding the M eigenvectors of \mathbf{S} corresponding to the M largest eigenvalues. Algorithms for finding eigenvectors and eigenvalues, as well as additional theorems related to eigenvector decomposition, can be found in Golub and Van Loan (1996). Note that the computational cost of computing the full eigenvector decomposition for a matrix of size $D \times D$ is $O(D^3)$. If we plan to project our data onto the first M principal components, then we only need to find the first M eigenvalues and eigenvectors. This can be done with more efficient techniques, such as the *power method* (Golub and Van Loan, 1996), that scale like $O(MD^2)$, or alternatively we can make use of the EM algorithm.

12.1.2 Minimum-error formulation

We now discuss an alternative formulation of PCA based on projection error minimization. To do this, we introduce a complete orthonormal set of D -dimensional basis vectors $\{\mathbf{u}_i\}$ where $i = 1, \dots, D$ that satisfy

$$\mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}. \quad (12.7)$$

Because this basis is complete, each data point can be represented exactly by a linear combination of the basis vectors

$$\mathbf{x}_n = \sum_{i=1}^D \alpha_{ni} \mathbf{u}_i \quad (12.8)$$

where the coefficients α_{ni} will be different for different data points. This simply corresponds to a rotation of the coordinate system to a new system defined by the $\{\mathbf{u}_i\}$, and the original D components $\{x_{n1}, \dots, x_{nD}\}$ are replaced by an equivalent set $\{\alpha_{n1}, \dots, \alpha_{nD}\}$. Taking the inner product with \mathbf{u}_j , and making use of the orthonormality property, we obtain $\alpha_{nj} = \mathbf{x}_n^T \mathbf{u}_j$, and so without loss of generality we can write

$$\mathbf{x}_n = \sum_{i=1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i. \quad (12.9)$$

Our goal, however, is to approximate this data point using a representation involving a restricted number $M < D$ of variables corresponding to a projection onto a lower-dimensional subspace. The M -dimensional linear subspace can be represented, without loss of generality, by the first M of the basis vectors, and so we approximate each data point \mathbf{x}_n by

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^M z_{ni} \mathbf{u}_i + \sum_{i=M+1}^D b_i \mathbf{u}_i \quad (12.10)$$

where the $\{z_{ni}\}$ depend on the particular data point, whereas the $\{b_i\}$ are constants that are the same for all data points. We are free to choose the $\{\mathbf{u}_i\}$, the $\{z_{ni}\}$, and the $\{b_i\}$ so as to minimize the distortion introduced by the reduction in dimensionality. As our distortion measure, we shall use the squared distance between the original data point \mathbf{x}_n and its approximation $\tilde{\mathbf{x}}_n$, averaged over the data set, so that our goal is to minimize

$$J = \frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2. \quad (12.11)$$

Consider first of all the minimization with respect to the quantities $\{z_{ni}\}$. Substituting for $\tilde{\mathbf{x}}_n$, setting the derivative with respect to z_{nj} to zero, and making use of the orthonormality conditions, we obtain

$$z_{nj} = \mathbf{x}_n^T \mathbf{u}_j \quad (12.12)$$

where $j = 1, \dots, M$. Similarly, setting the derivative of J with respect to b_i to zero, and again making use of the orthonormality relations, gives

$$b_j = \bar{\mathbf{x}}^T \mathbf{u}_j \quad (12.13)$$

where $j = M+1, \dots, D$. If we substitute for z_{ni} and b_i , and make use of the general expansion (12.9), we obtain

$$\mathbf{x}_n - \tilde{\mathbf{x}}_n = \sum_{i=M+1}^D \{(\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i\} \mathbf{u}_i \quad (12.14)$$

from which we see that the displacement vector from \mathbf{x}_n to $\tilde{\mathbf{x}}_n$ lies in the space orthogonal to the principal subspace, because it is a linear combination of $\{\mathbf{u}_i\}$ for $i = M+1, \dots, D$, as illustrated in Figure 12.2. This is to be expected because the projected points $\tilde{\mathbf{x}}_n$ must lie within the principal subspace, but we can move them freely within that subspace, and so the minimum error is given by the orthogonal projection.

We therefore obtain an expression for the distortion measure J as a function purely of the $\{\mathbf{u}_i\}$ in the form

$$J = \frac{1}{N} \sum_{n=1}^N \sum_{i=M+1}^D (\mathbf{x}_n^T \mathbf{u}_i - \bar{\mathbf{x}}^T \mathbf{u}_i)^2 = \sum_{i=M+1}^D \mathbf{u}_i^T \mathbf{S} \mathbf{u}_i. \quad (12.15)$$

There remains the task of minimizing J with respect to the $\{\mathbf{u}_i\}$, which must be a constrained minimization otherwise we will obtain the vacuous result $\mathbf{u}_i = 0$. The constraints arise from the orthonormality conditions and, as we shall see, the solution will be expressed in terms of the eigenvector expansion of the covariance matrix. Before considering a formal solution, let us try to obtain some intuition about the result by considering the case of a two-dimensional data space $D = 2$ and a one-dimensional principal subspace $M = 1$. We have to choose a direction \mathbf{u}_2 so as to

minimize $J = \mathbf{u}_2^T \mathbf{S} \mathbf{u}_2$, subject to the normalization constraint $\mathbf{u}_2^T \mathbf{u}_2 = 1$. Using a Lagrange multiplier λ_2 to enforce the constraint, we consider the minimization of

$$\tilde{J} = \mathbf{u}_2^T \mathbf{S} \mathbf{u}_2 + \lambda_2 (1 - \mathbf{u}_2^T \mathbf{u}_2). \quad (12.16)$$

Setting the derivative with respect to \mathbf{u}_2 to zero, we obtain $\mathbf{S} \mathbf{u}_2 = \lambda_2 \mathbf{u}_2$ so that \mathbf{u}_2 is an eigenvector of \mathbf{S} with eigenvalue λ_2 . Thus any eigenvector will define a stationary point of the distortion measure. To find the value of J at the minimum, we back-substitute the solution for \mathbf{u}_2 into the distortion measure to give $J = \lambda_2$. We therefore obtain the minimum value of J by choosing \mathbf{u}_2 to be the eigenvector corresponding to the smaller of the two eigenvalues. Thus we should choose the principal subspace to be aligned with the eigenvector having the *larger* eigenvalue. This result accords with our intuition that, in order to minimize the average squared projection distance, we should choose the principal component subspace to pass through the mean of the data points and to be aligned with the directions of maximum variance. For the case when the eigenvalues are equal, any choice of principal direction will give rise to the same value of J .

Exercise 12.2

The general solution to the minimization of J for arbitrary D and arbitrary $M < D$ is obtained by choosing the $\{\mathbf{u}_i\}$ to be eigenvectors of the covariance matrix given by

$$\mathbf{S} \mathbf{u}_i = \lambda_i \mathbf{u}_i \quad (12.17)$$

where $i = 1, \dots, D$, and as usual the eigenvectors $\{\mathbf{u}_i\}$ are chosen to be orthonormal. The corresponding value of the distortion measure is then given by

$$J = \sum_{i=M+1}^D \lambda_i \quad (12.18)$$

which is simply the sum of the eigenvalues of those eigenvectors that are orthogonal to the principal subspace. We therefore obtain the minimum value of J by selecting these eigenvectors to be those having the $D - M$ smallest eigenvalues, and hence the eigenvectors defining the principal subspace are those corresponding to the M largest eigenvalues.

Although we have considered $M < D$, the PCA analysis still holds if $M = D$, in which case there is no dimensionality reduction but simply a rotation of the coordinate axes to align with principal components.

Finally, it is worth noting that there exists a closely related linear dimensionality reduction technique called *canonical correlation analysis*, or CCA (Hotelling, 1936; Bach and Jordan, 2002). Whereas PCA works with a single random variable, CCA considers two (or more) variables and tries to find a corresponding pair of linear subspaces that have high cross-correlation, so that each component within one of the subspaces is correlated with a single component from the other subspace. Its solution can be expressed in terms of a generalized eigenvector problem.

12.1.3 Applications of PCA

We can illustrate the use of PCA for data compression by considering the off-line digits data set. Because each eigenvector of the covariance matrix is a vector

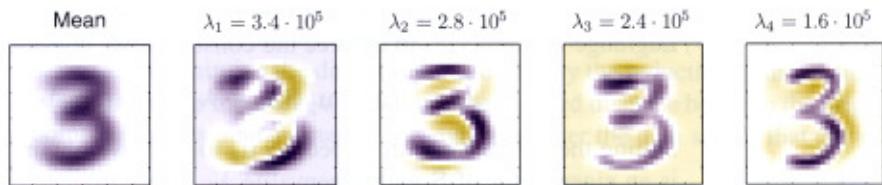


Figure 12.3 The mean vector $\bar{\mathbf{x}}$ along with the first four PCA eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_4$ for the off-line digits data set, together with the corresponding eigenvalues.

in the original D -dimensional space, we can represent the eigenvectors as images of the same size as the data points. The first five eigenvectors, along with the corresponding eigenvalues, are shown in Figure 12.3. A plot of the complete spectrum of eigenvalues, sorted into decreasing order, is shown in Figure 12.4(a). The distortion measure J associated with choosing a particular value of M is given by the sum of the eigenvalues from $M + 1$ up to D and is plotted for different values of M in Figure 12.4(b).

If we substitute (12.12) and (12.13) into (12.10), we can write the PCA approximation to a data vector \mathbf{x}_n in the form

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^M (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i + \sum_{i=M+1}^D (\bar{\mathbf{x}}^T \mathbf{u}_i) \mathbf{u}_i \quad (12.19)$$

$$= \bar{\mathbf{x}} + \sum_{i=1}^M (\mathbf{x}_n^T \mathbf{u}_i - \bar{\mathbf{x}}^T \mathbf{u}_i) \mathbf{u}_i \quad (12.20)$$

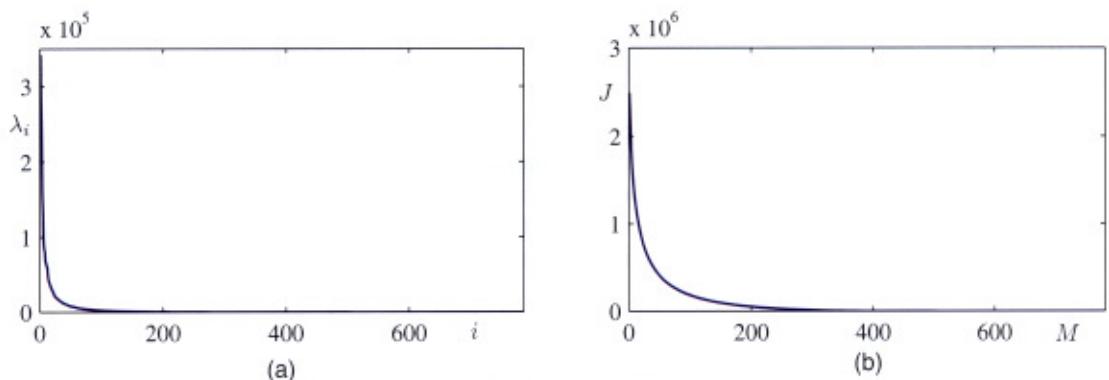


Figure 12.4 (a) Plot of the eigenvalue spectrum for the off-line digits data set. (b) Plot of the sum of the discarded eigenvalues, which represents the sum-of-squares distortion J introduced by projecting the data onto a principal component subspace of dimensionality M .

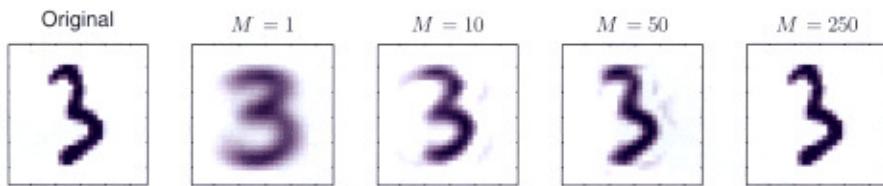


Figure 12.5 An original example from the off-line digits data set together with its PCA reconstructions obtained by retaining M principal components for various values of M . As M increases the reconstruction becomes more accurate and would become perfect when $M = D = 28 \times 28 = 784$.

where we have made use of the relation

$$\bar{\mathbf{x}} = \sum_{i=1}^D (\bar{\mathbf{x}}^T \mathbf{u}_i) \mathbf{u}_i \quad (12.21)$$

which follows from the completeness of the $\{\mathbf{u}_i\}$. This represents a compression of the data set, because for each data point we have replaced the D -dimensional vector \mathbf{x}_n with an M -dimensional vector having components $(\mathbf{x}_n^T \mathbf{u}_i - \bar{\mathbf{x}}^T \mathbf{u}_i)$. The smaller the value of M , the greater the degree of compression. Examples of PCA reconstructions of data points for the digits data set are shown in Figure 12.5.

Another application of principal component analysis is to data pre-processing. In this case, the goal is not dimensionality reduction but rather the transformation of a data set in order to standardize certain of its properties. This can be important in allowing subsequent pattern recognition algorithms to be applied successfully to the data set. Typically, it is done when the original variables are measured in various different units or have significantly different variability. For instance in the Old Faithful data set, the time between eruptions is typically an order of magnitude greater than the duration of an eruption. When we applied the K -means algorithm to this data set, we first made a separate linear re-scaling of the individual variables such that each variable had zero mean and unit variance. This is known as *standardizing* the data, and the covariance matrix for the standardized data has components

$$\rho_{ij} = \frac{1}{N} \sum_{n=1}^N \frac{(x_{ni} - \bar{x}_i)}{\sigma_i} \frac{(x_{nj} - \bar{x}_j)}{\sigma_j} \quad (12.22)$$

where σ_i is the variance of x_i . This is known as the *correlation* matrix of the original data and has the property that if two components x_i and x_j of the data are perfectly correlated, then $\rho_{ij} = 1$, and if they are uncorrelated, then $\rho_{ij} = 0$.

However, using PCA we can make a more substantial normalization of the data to give it zero mean and unit covariance, so that different variables become decorrelated. To do this, we first write the eigenvector equation (12.17) in the form

$$\mathbf{S}\mathbf{U} = \mathbf{U}\mathbf{L} \quad (12.23)$$

Appendix A

Section 9.1

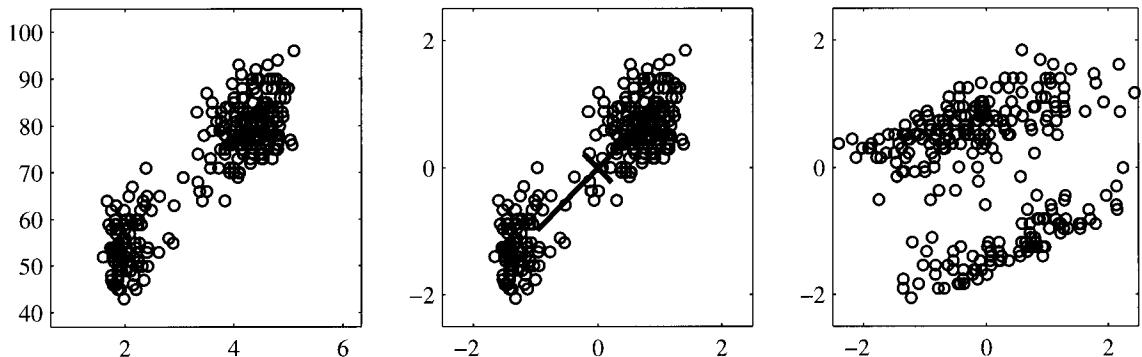


Figure 12.6 Illustration of the effects of linear pre-processing applied to the Old Faithful data set. The plot on the left shows the original data. The centre plot shows the result of standardizing the individual variables to zero mean and unit variance. Also shown are the principal axes of this normalized data set, plotted over the range $\pm \lambda_i^{1/2}$. The plot on the right shows the result of whitening of the data to give it zero mean and unit covariance.

where \mathbf{L} is a $D \times D$ diagonal matrix with elements λ_i , and \mathbf{U} is a $D \times D$ orthogonal matrix with columns given by \mathbf{u}_i . Then we define, for each data point \mathbf{x}_n , a transformed value given by

$$\mathbf{y}_n = \mathbf{L}^{-1/2} \mathbf{U}^T (\mathbf{x}_n - \bar{\mathbf{x}}) \quad (12.24)$$

where $\bar{\mathbf{x}}$ is the sample mean defined by (12.1). Clearly, the set $\{\mathbf{y}_n\}$ has zero mean, and its covariance is given by the identity matrix because

$$\begin{aligned} \frac{1}{N} \sum_{n=1}^N \mathbf{y}_n \mathbf{y}_n^T &= \frac{1}{N} \sum_{n=1}^N \mathbf{L}^{-1/2} \mathbf{U}^T (\mathbf{x}_n - \bar{\mathbf{x}}) (\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{U} \mathbf{L}^{-1/2} \\ &= \mathbf{L}^{-1/2} \mathbf{U}^T \mathbf{S} \mathbf{U} \mathbf{L}^{-1/2} = \mathbf{L}^{-1/2} \mathbf{L} \mathbf{L}^{-1/2} = \mathbf{I}. \end{aligned} \quad (12.25)$$

This operation is known as *whitening* or *sphereing* the data and is illustrated for the Old Faithful data set in Figure 12.6.

Appendix A

It is interesting to compare PCA with the Fisher linear discriminant which was discussed in Section 4.1.4. Both methods can be viewed as techniques for linear dimensionality reduction. However, PCA is unsupervised and depends only on the values \mathbf{x}_n whereas Fisher linear discriminant also uses class-label information. This difference is highlighted by the example in Figure 12.7.

Another common application of principal component analysis is to data visualization. Here each data point is projected onto a two-dimensional ($M = 2$) principal subspace, so that a data point \mathbf{x}_n is plotted at Cartesian coordinates given by $\mathbf{x}_n^T \mathbf{u}_1$ and $\mathbf{x}_n^T \mathbf{u}_2$, where \mathbf{u}_1 and \mathbf{u}_2 are the eigenvectors corresponding to the largest and second largest eigenvalues. An example of such a plot, for the oil flow data set, is shown in Figure 12.8.

Figure 12.7 A comparison of principal component analysis with Fisher's linear discriminant for linear dimensionality reduction. Here the data in two dimensions, belonging to two classes shown in red and blue, is to be projected onto a single dimension. PCA chooses the direction of maximum variance, shown by the magenta curve, which leads to strong class overlap, whereas the Fisher linear discriminant takes account of the class labels and leads to a projection onto the green curve giving much better class separation.

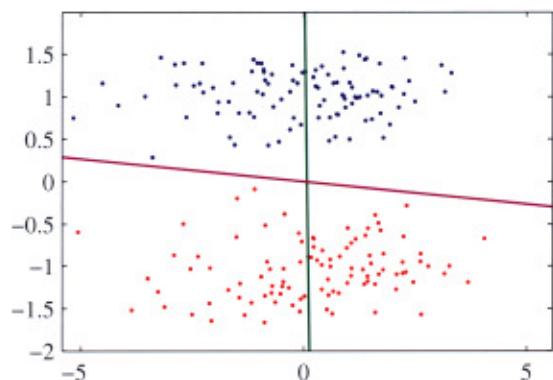
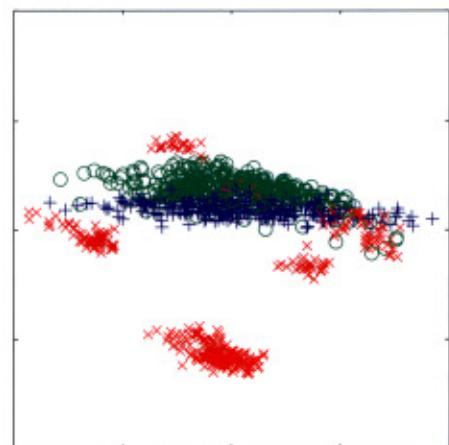


Figure 12.8 Visualization of the oil flow data set obtained by projecting the data onto the first two principal components. The red, blue, and green points correspond to the 'laminar', 'homogeneous', and 'annular' flow configurations respectively.



12.1.4 PCA for high-dimensional data

In some applications of principal component analysis, the number of data points is smaller than the dimensionality of the data space. For example, we might want to apply PCA to a data set of a few hundred images, each of which corresponds to a vector in a space of potentially several million dimensions (corresponding to three colour values for each of the pixels in the image). Note that in a D -dimensional space a set of N points, where $N < D$, defines a linear subspace whose dimensionality is at most $N - 1$, and so there is little point in applying PCA for values of M that are greater than $N - 1$. Indeed, if we perform PCA we will find that at least $D - N + 1$ of the eigenvalues are zero, corresponding to eigenvectors along whose directions the data set has zero variance. Furthermore, typical algorithms for finding the eigenvectors of a $D \times D$ matrix have a computational cost that scales like $O(D^3)$, and so for applications such as the image example, a direct application of PCA will be computationally infeasible.

We can resolve this problem as follows. First, let us define \mathbf{X} to be the $(N \times D)$ -

dimensional centred data matrix, whose n^{th} row is given by $(\mathbf{x}_n - \bar{\mathbf{x}})^T$. The covariance matrix (12.3) can then be written as $\mathbf{S} = N^{-1}\mathbf{X}^T\mathbf{X}$, and the corresponding eigenvector equation becomes

$$\frac{1}{N}\mathbf{X}^T\mathbf{X}\mathbf{u}_i = \lambda_i\mathbf{u}_i. \quad (12.26)$$

Now pre-multiply both sides by \mathbf{X} to give

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T(\mathbf{X}\mathbf{u}_i) = \lambda_i(\mathbf{X}\mathbf{u}_i). \quad (12.27)$$

If we now define $\mathbf{v}_i = \mathbf{X}\mathbf{u}_i$, we obtain

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T\mathbf{v}_i = \lambda_i\mathbf{v}_i \quad (12.28)$$

which is an eigenvector equation for the $N \times N$ matrix $N^{-1}\mathbf{X}\mathbf{X}^T$. We see that this has the same $N - 1$ eigenvalues as the original covariance matrix (which itself has an additional $D - N + 1$ eigenvalues of value zero). Thus we can solve the eigenvector problem in spaces of lower dimensionality with computational cost $O(N^3)$ instead of $O(D^3)$. In order to determine the eigenvectors, we multiply both sides of (12.28) by \mathbf{X}^T to give

$$\left(\frac{1}{N}\mathbf{X}^T\mathbf{X}\right)(\mathbf{X}^T\mathbf{v}_i) = \lambda_i(\mathbf{X}^T\mathbf{v}_i) \quad (12.29)$$

from which we see that $(\mathbf{X}^T\mathbf{v}_i)$ is an eigenvector of \mathbf{S} with eigenvalue λ_i . Note, however, that these eigenvectors need not be normalized. To determine the appropriate normalization, we re-scale $\mathbf{u}_i \propto \mathbf{X}^T\mathbf{v}_i$ by a constant such that $\|\mathbf{u}_i\| = 1$, which, assuming \mathbf{v}_i has been normalized to unit length, gives

$$\mathbf{u}_i = \frac{1}{(N\lambda_i)^{1/2}}\mathbf{X}^T\mathbf{v}_i. \quad (12.30)$$

In summary, to apply this approach we first evaluate $\mathbf{X}\mathbf{X}^T$ and then find its eigenvectors and eigenvalues and then compute the eigenvectors in the original data space using (12.30).

12.2. Probabilistic PCA

The formulation of PCA discussed in the previous section was based on a linear projection of the data onto a subspace of lower dimensionality than the original data space. We now show that PCA can also be expressed as the maximum likelihood solution of a probabilistic latent variable model. This reformulation of PCA, known as *probabilistic PCA*, brings several advantages compared with conventional PCA:

- Probabilistic PCA represents a constrained form of the Gaussian distribution in which the number of free parameters can be restricted while still allowing the model to capture the dominant correlations in a data set.

- We can derive an EM algorithm for PCA that is computationally efficient in situations where only a few leading eigenvectors are required and that avoids having to evaluate the data covariance matrix as an intermediate step.

Section 12.2.2

- The combination of a probabilistic model and EM allows us to deal with missing values in the data set.
- Mixtures of probabilistic PCA models can be formulated in a principled way and trained using the EM algorithm.
- Probabilistic PCA forms the basis for a Bayesian treatment of PCA in which the dimensionality of the principal subspace can be found automatically from the data.

Section 12.2.3

- The existence of a likelihood function allows direct comparison with other probabilistic density models. By contrast, conventional PCA will assign a low reconstruction cost to data points that are close to the principal subspace even if they lie arbitrarily far from the training data.
- Probabilistic PCA can be used to model class-conditional densities and hence be applied to classification problems.
- The probabilistic PCA model can be run generatively to provide samples from the distribution.

This formulation of PCA as a probabilistic model was proposed independently by Tipping and Bishop (1997, 1999b) and by Roweis (1998). As we shall see later, it is closely related to *factor analysis* (Basilevsky, 1994).

Section 8.1.4

Probabilistic PCA is a simple example of the linear-Gaussian framework, in which all of the marginal and conditional distributions are Gaussian. We can formulate probabilistic PCA by first introducing an explicit latent variable \mathbf{z} corresponding to the principal-component subspace. Next we define a Gaussian prior distribution $p(\mathbf{z})$ over the latent variable, together with a Gaussian conditional distribution $p(\mathbf{x}|\mathbf{z})$ for the observed variable \mathbf{x} conditioned on the value of the latent variable. Specifically, the prior distribution over \mathbf{z} is given by a zero-mean unit-covariance Gaussian

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}). \quad (12.31)$$

Similarly, the conditional distribution of the observed variable \mathbf{x} , conditioned on the value of the latent variable \mathbf{z} , is again Gaussian, of the form

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2 \mathbf{I}) \quad (12.32)$$

in which the mean of \mathbf{x} is a general linear function of \mathbf{z} governed by the $D \times M$ matrix \mathbf{W} and the D -dimensional vector $\boldsymbol{\mu}$. Note that this factorizes with respect to the elements of \mathbf{x} , in other words this is an example of the naive Bayes model. As we shall see shortly, the columns of \mathbf{W} span a linear subspace within the data space that corresponds to the principal subspace. The other parameter in this model is the scalar σ^2 governing the variance of the conditional distribution. Note that there is no

Section 8.2.2

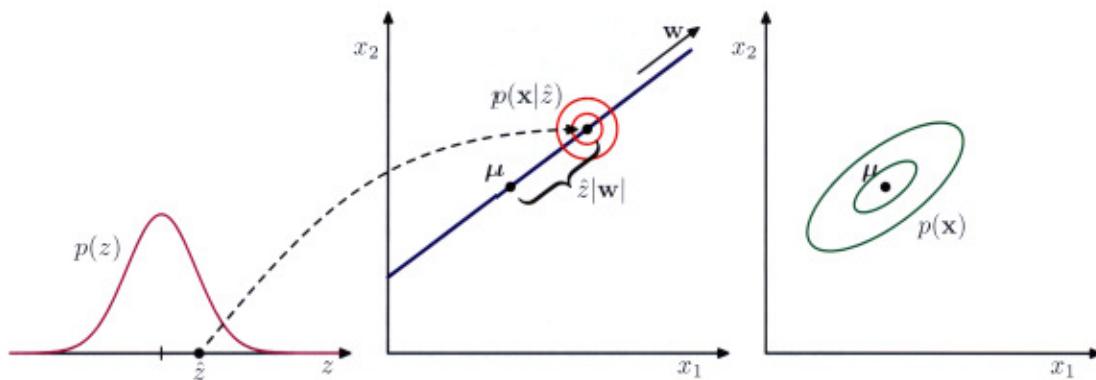


Figure 12.9 An illustration of the generative view of the probabilistic PCA model for a two-dimensional data space and a one-dimensional latent space. An observed data point \mathbf{x} is generated by first drawing a value \hat{z} for the latent variable from its prior distribution $p(z)$ and then drawing a value for \mathbf{x} from an isotropic Gaussian distribution (illustrated by the red circles) having mean $w\hat{z} + \mu$ and covariance $\sigma^2\mathbf{I}$. The green ellipses show the density contours for the marginal distribution $p(\mathbf{x})$.

loss of generality in assuming a zero mean, unit covariance Gaussian for the latent distribution $p(\mathbf{z})$ because a more general Gaussian distribution would give rise to an equivalent probabilistic model.

Exercise 12.4

We can view the probabilistic PCA model from a generative viewpoint in which a sampled value of the observed variable is obtained by first choosing a value for the latent variable and then sampling the observed variable conditioned on this latent value. Specifically, the D -dimensional observed variable \mathbf{x} is defined by a linear transformation of the M -dimensional latent variable \mathbf{z} plus additive Gaussian ‘noise’, so that

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \quad (12.33)$$

where \mathbf{z} is an M -dimensional Gaussian latent variable, and $\boldsymbol{\epsilon}$ is a D -dimensional zero-mean Gaussian-distributed noise variable with covariance $\sigma^2\mathbf{I}$. This generative process is illustrated in Figure 12.9. Note that this framework is based on a mapping from latent space to data space, in contrast to the more conventional view of PCA discussed above. The reverse mapping, from data space to the latent space, will be obtained shortly using Bayes’ theorem.

Suppose we wish to determine the values of the parameters \mathbf{W} , $\boldsymbol{\mu}$ and σ^2 using maximum likelihood. To write down the likelihood function, we need an expression for the marginal distribution $p(\mathbf{x})$ of the observed variable. This is expressed, from the sum and product rules of probability, in the form

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z}) d\mathbf{z}. \quad (12.34)$$

Exercise 12.7

Because this corresponds to a linear-Gaussian model, this marginal distribution is again Gaussian, and is given by

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{C}) \quad (12.35)$$

where the $D \times D$ covariance matrix \mathbf{C} is defined by

$$\mathbf{C} = \mathbf{WW}^T + \sigma^2 \mathbf{I}. \quad (12.36)$$

This result can also be derived more directly by noting that the predictive distribution will be Gaussian and then evaluating its mean and covariance using (12.33). This gives

$$\mathbb{E}[\mathbf{x}] = \mathbb{E}[\mathbf{Wz} + \boldsymbol{\mu} + \boldsymbol{\epsilon}] = \boldsymbol{\mu} \quad (12.37)$$

$$\begin{aligned} \text{cov}[\mathbf{x}] &= \mathbb{E}[(\mathbf{Wz} + \boldsymbol{\epsilon})(\mathbf{Wz} + \boldsymbol{\epsilon})^T] \\ &= \mathbb{E}[\mathbf{Wzz}^T \mathbf{W}^T] + \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T] = \mathbf{WW}^T + \sigma^2 \mathbf{I} \end{aligned} \quad (12.38)$$

where we have used the fact that \mathbf{z} and $\boldsymbol{\epsilon}$ are independent random variables and hence are uncorrelated.

Intuitively, we can think of the distribution $p(\mathbf{x})$ as being defined by taking an isotropic Gaussian ‘spray can’ and moving it across the principal subspace spraying Gaussian ink with density determined by σ^2 and weighted by the prior distribution. The accumulated ink density gives rise to a ‘pancake’ shaped distribution representing the marginal density $p(\mathbf{x})$.

The predictive distribution $p(\mathbf{x})$ is governed by the parameters $\boldsymbol{\mu}$, \mathbf{W} , and σ^2 . However, there is redundancy in this parameterization corresponding to rotations of the latent space coordinates. To see this, consider a matrix $\widetilde{\mathbf{W}} = \mathbf{WR}$ where \mathbf{R} is an orthogonal matrix. Using the orthogonality property $\mathbf{RR}^T = \mathbf{I}$, we see that the quantity $\widetilde{\mathbf{W}}\widetilde{\mathbf{W}}^T$ that appears in the covariance matrix \mathbf{C} takes the form

$$\widetilde{\mathbf{W}}\widetilde{\mathbf{W}}^T = \mathbf{WRR}^T \mathbf{W}^T = \mathbf{WW}^T \quad (12.39)$$

and hence is independent of \mathbf{R} . Thus there is a whole family of matrices $\widetilde{\mathbf{W}}$ all of which give rise to the same predictive distribution. This invariance can be understood in terms of rotations within the latent space. We shall return to a discussion of the number of independent parameters in this model later.

When we evaluate the predictive distribution, we require \mathbf{C}^{-1} , which involves the inversion of a $D \times D$ matrix. The computation required to do this can be reduced by making use of the matrix inversion identity (C.7) to give

$$\mathbf{C}^{-1} = \sigma^{-2} \mathbf{M}^{-1} \mathbf{W}^T \quad (12.40)$$

where the $M \times M$ matrix \mathbf{M} is defined by

$$\mathbf{M} = \mathbf{W}^T \mathbf{W} + \sigma^2 \mathbf{I}. \quad (12.41)$$

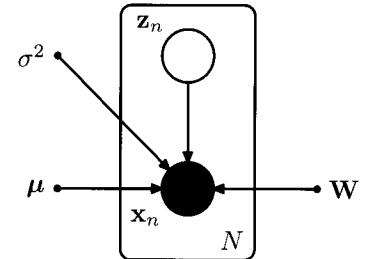
Because we invert \mathbf{M} rather than inverting \mathbf{C} directly, the cost of evaluating \mathbf{C}^{-1} is reduced from $O(D^3)$ to $O(M^3)$.

As well as the predictive distribution $p(\mathbf{x})$, we will also require the posterior distribution $p(\mathbf{z}|\mathbf{x})$, which can again be written down directly using the result (2.116) for linear-Gaussian models to give

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z} | \mathbf{M}^{-1} \mathbf{W}^T (\mathbf{x} - \boldsymbol{\mu}), \sigma^{-2} \mathbf{M}). \quad (12.42)$$

Note that the posterior mean depends on \mathbf{x} , whereas the posterior covariance is independent of \mathbf{x} .

Figure 12.10 The probabilistic PCA model for a data set of N observations of \mathbf{x} can be expressed as a directed graph in which each observation \mathbf{x}_n is associated with a value \mathbf{z}_n of the latent variable.



12.2.1 Maximum likelihood PCA

We next consider the determination of the model parameters using maximum likelihood. Given a data set $\mathbf{X} = \{\mathbf{x}_n\}$ of observed data points, the probabilistic PCA model can be expressed as a directed graph, as shown in Figure 12.10. The corresponding log likelihood function is given, from (12.35), by

$$\begin{aligned}\ln p(\mathbf{X}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2) &= \sum_{n=1}^N \ln p(\mathbf{x}_n|\mathbf{W}, \boldsymbol{\mu}, \sigma^2) \\ &= -\frac{ND}{2} \ln(2\pi) - \frac{N}{2} \ln |\mathbf{C}| - \frac{1}{2} \sum_{n=1}^N (\mathbf{x}_n - \boldsymbol{\mu})^T \mathbf{C}^{-1} (\mathbf{x}_n - \boldsymbol{\mu}).\end{aligned}\quad (12.43)$$

Setting the derivative of the log likelihood with respect to $\boldsymbol{\mu}$ equal to zero gives the expected result $\boldsymbol{\mu} = \bar{\mathbf{x}}$ where $\bar{\mathbf{x}}$ is the data mean defined by (12.1). Back-substituting we can then write the log likelihood function in the form

$$\ln p(\mathbf{X}|\mathbf{W}, \boldsymbol{\mu}, \sigma^2) = -\frac{N}{2} \{ D \ln(2\pi) + \ln |\mathbf{C}| + \text{Tr}(\mathbf{C}^{-1} \mathbf{S}) \} \quad (12.44)$$

where \mathbf{S} is the data covariance matrix defined by (12.3). Because the log likelihood is a quadratic function of $\boldsymbol{\mu}$, this solution represents the unique maximum, as can be confirmed by computing second derivatives.

Maximization with respect to \mathbf{W} and σ^2 is more complex but nonetheless has an exact closed-form solution. It was shown by Tipping and Bishop (1999b) that all of the stationary points of the log likelihood function can be written as

$$\mathbf{W}_{\text{ML}} = \mathbf{U}_M (\mathbf{L}_M - \sigma^2 \mathbf{I})^{1/2} \mathbf{R} \quad (12.45)$$

where \mathbf{U}_M is a $D \times M$ matrix whose columns are given by any subset (of size M) of the eigenvectors of the data covariance matrix \mathbf{S} , the $M \times M$ diagonal matrix \mathbf{L}_M has elements given by the corresponding eigenvalues λ_i , and \mathbf{R} is an arbitrary $M \times M$ orthogonal matrix.

Furthermore, Tipping and Bishop (1999b) showed that the *maximum* of the likelihood function is obtained when the M eigenvectors are chosen to be those whose eigenvalues are the M largest (all other solutions being saddle points). A similar result was conjectured independently by Roweis (1998), although no proof was given.

Again, we shall assume that the eigenvectors have been arranged in order of decreasing values of the corresponding eigenvalues, so that the M principal eigenvectors are $\mathbf{u}_1, \dots, \mathbf{u}_M$. In this case, the columns of \mathbf{W} define the principal subspace of standard PCA. The corresponding maximum likelihood solution for σ^2 is then given by

$$\sigma_{\text{ML}}^2 = \frac{1}{D - M} \sum_{i=M+1}^D \lambda_i \quad (12.46)$$

so that σ_{ML}^2 is the average variance associated with the discarded dimensions.

Because \mathbf{R} is orthogonal, it can be interpreted as a rotation matrix in the $M \times M$ latent space. If we substitute the solution for \mathbf{W} into the expression for \mathbf{C} , and make use of the orthogonality property $\mathbf{R}\mathbf{R}^T = \mathbf{I}$, we see that \mathbf{C} is independent of \mathbf{R} . This simply says that the predictive density is unchanged by rotations in the latent space as discussed earlier. For the particular case of $\mathbf{R} = \mathbf{I}$, we see that the columns of \mathbf{W} are the principal component eigenvectors scaled by the variance parameters $\lambda_i - \sigma^2$. The interpretation of these scaling factors is clear once we recognize that for a convolution of independent Gaussian distributions (in this case the latent space distribution and the noise model) the variances are additive. Thus the variance λ_i in the direction of an eigenvector \mathbf{u}_i is composed of the sum of a contribution $\lambda_i - \sigma^2$ from the projection of the unit-variance latent space distribution into data space through the corresponding column of \mathbf{W} , plus an isotropic contribution of variance σ^2 which is added in all directions by the noise model.

It is worth taking a moment to study the form of the covariance matrix given by (12.36). Consider the variance of the predictive distribution along some direction specified by the unit vector \mathbf{v} , where $\mathbf{v}^T \mathbf{v} = 1$, which is given by $\mathbf{v}^T \mathbf{C} \mathbf{v}$. First suppose that \mathbf{v} is orthogonal to the principal subspace, in other words it is given by some linear combination of the discarded eigenvectors. Then $\mathbf{v}^T \mathbf{U} = 0$ and hence $\mathbf{v}^T \mathbf{C} \mathbf{v} = \sigma^2$. Thus the model predicts a noise variance orthogonal to the principal subspace, which, from (12.46), is just the average of the discarded eigenvalues. Now suppose that $\mathbf{v} = \mathbf{u}_i$ where \mathbf{u}_i is one of the retained eigenvectors defining the principal subspace. Then $\mathbf{v}^T \mathbf{C} \mathbf{v} = (\lambda_i - \sigma^2) + \sigma^2 = \lambda_i$. In other words, this model correctly captures the variance of the data along the principal axes, and approximates the variance in all remaining directions with a single average value σ^2 .

One way to construct the maximum likelihood density model would simply be to find the eigenvectors and eigenvalues of the data covariance matrix and then to evaluate \mathbf{W} and σ^2 using the results given above. In this case, we would choose $\mathbf{R} = \mathbf{I}$ for convenience. However, if the maximum likelihood solution is found by numerical optimization of the likelihood function, for instance using an algorithm such as conjugate gradients (Fletcher, 1987; Nocedal and Wright, 1999; Bishop and Nabney, 2008) or through the EM algorithm, then the resulting value of \mathbf{R} is essentially arbitrary. This implies that the columns of \mathbf{W} need not be orthogonal. If an orthogonal basis is required, the matrix \mathbf{W} can be post-processed appropriately (Golub and Van Loan, 1996). Alternatively, the EM algorithm can be modified in such a way as to yield orthonormal principal directions, sorted in descending order of the corresponding eigenvalues, directly (Ahn and Oh, 2003).

The rotational invariance in latent space represents a form of statistical nonidentifiability, analogous to that encountered for mixture models in the case of discrete latent variables. Here there is a continuum of parameters all of which lead to the same predictive density, in contrast to the discrete nonidentifiability associated with component re-labelling in the mixture setting.

If we consider the case of $M = D$, so that there is no reduction of dimensionality, then $\mathbf{U}_M = \mathbf{U}$ and $\mathbf{L}_M = \mathbf{L}$. Making use of the orthogonality properties $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ and $\mathbf{R}\mathbf{R}^T = \mathbf{I}$, we see that the covariance \mathbf{C} of the marginal distribution for \mathbf{x} becomes

$$\mathbf{C} = \mathbf{U}(\mathbf{L} - \sigma^2\mathbf{I})^{1/2}\mathbf{R}\mathbf{R}^T(\mathbf{L} - \sigma^2\mathbf{I})^{1/2}\mathbf{U}^T + \sigma^2\mathbf{I} = \mathbf{U}\mathbf{L}\mathbf{U}^T = \mathbf{S} \quad (12.47)$$

and so we obtain the standard maximum likelihood solution for an unconstrained Gaussian distribution in which the covariance matrix is given by the sample covariance.

Conventional PCA is generally formulated as a projection of points from the D -dimensional data space onto an M -dimensional linear subspace. Probabilistic PCA, however, is most naturally expressed as a mapping from the latent space into the data space via (12.33). For applications such as visualization and data compression, we can reverse this mapping using Bayes' theorem. Any point \mathbf{x} in data space can then be summarized by its posterior mean and covariance in latent space. From (12.42) the mean is given by

$$\mathbb{E}[\mathbf{z}|\mathbf{x}] = \mathbf{M}^{-1}\mathbf{W}_{ML}^T(\mathbf{x} - \bar{\mathbf{x}}) \quad (12.48)$$

where \mathbf{M} is given by (12.41). This projects to a point in data space given by

$$\mathbf{W}\mathbb{E}[\mathbf{z}|\mathbf{x}] + \boldsymbol{\mu}. \quad (12.49)$$

Section 3.3.1

Note that this takes the same form as the equations for regularized linear regression and is a consequence of maximizing the likelihood function for a linear Gaussian model. Similarly, the posterior covariance is given from (12.42) by $\sigma^2\mathbf{M}^{-1}$ and is independent of \mathbf{x} .

If we take the limit $\sigma^2 \rightarrow 0$, then the posterior mean reduces to

$$(\mathbf{W}_{ML}^T \mathbf{W}_{ML})^{-1} \mathbf{W}_{ML}^T (\mathbf{x} - \bar{\mathbf{x}}) \quad (12.50)$$

Exercise 12.11

which represents an orthogonal projection of the data point onto the latent space, and so we recover the standard PCA model. The posterior covariance in this limit is zero, however, and the density becomes singular. For $\sigma^2 > 0$, the latent projection is shifted towards the origin, relative to the orthogonal projection.

Exercise 12.12

Finally, we note that an important role for the probabilistic PCA model is in defining a multivariate Gaussian distribution in which the number of degrees of freedom, in other words the number of independent parameters, can be controlled whilst still allowing the model to capture the dominant correlations in the data. Recall that a general Gaussian distribution has $D(D + 1)/2$ independent parameters in its covariance matrix (plus another D parameters in its mean). Thus the number of parameters scales quadratically with D and can become excessive in spaces of high

Section 2.3

dimensionality. If we restrict the covariance matrix to be diagonal, then it has only D independent parameters, and so the number of parameters now grows linearly with dimensionality. However, it now treats the variables as if they were independent and hence can no longer express any correlations between them. Probabilistic PCA provides an elegant compromise in which the M most significant correlations can be captured while still ensuring that the total number of parameters grows only linearly with D . We can see this by evaluating the number of degrees of freedom in the PPCA model as follows. The covariance matrix \mathbf{C} depends on the parameters \mathbf{W} , which has size $D \times M$, and σ^2 , giving a total parameter count of $DM + 1$. However, we have seen that there is some redundancy in this parameterization associated with rotations of the coordinate system in the latent space. The orthogonal matrix \mathbf{R} that expresses these rotations has size $M \times M$. In the first column of this matrix there are $M - 1$ independent parameters, because the column vector must be normalized to unit length. In the second column there are $M - 2$ independent parameters, because the column must be normalized and also must be orthogonal to the previous column, and so on. Summing this arithmetic series, we see that \mathbf{R} has a total of $M(M - 1)/2$ independent parameters. Thus the number of degrees of freedom in the covariance matrix \mathbf{C} is given by

$$DM + 1 - M(M - 1)/2. \quad (12.51)$$

The number of independent parameters in this model therefore only grows linearly with D , for fixed M . If we take $M = D - 1$, then we recover the standard result for a full covariance Gaussian. In this case, the variance along $D - 1$ linearly independent directions is controlled by the columns of \mathbf{W} , and the variance along the remaining direction is given by σ^2 . If $M = 0$, the model is equivalent to the isotropic covariance case.

Exercise 12.14

12.2.2 EM algorithm for PCA

As we have seen, the probabilistic PCA model can be expressed in terms of a marginalization over a continuous latent space \mathbf{z} in which for each data point \mathbf{x}_n , there is a corresponding latent variable \mathbf{z}_n . We can therefore make use of the EM algorithm to find maximum likelihood estimates of the model parameters. This may seem rather pointless because we have already obtained an exact closed-form solution for the maximum likelihood parameter values. However, in spaces of high dimensionality, there may be computational advantages in using an iterative EM procedure rather than working directly with the sample covariance matrix. This EM procedure can also be extended to the factor analysis model, for which there is no closed-form solution. Finally, it allows missing data to be handled in a principled way.

Section 12.2.4

Section 9.4

We can derive the EM algorithm for probabilistic PCA by following the general framework for EM. Thus we write down the complete-data log likelihood and take its expectation with respect to the posterior distribution of the latent distribution evaluated using ‘old’ parameter values. Maximization of this expected complete-data log likelihood then yields the ‘new’ parameter values. Because the data points

are assumed independent, the complete-data log likelihood function takes the form

$$\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2) = \sum_{n=1}^N \{\ln p(\mathbf{x}_n|\mathbf{z}_n) + \ln p(\mathbf{z}_n)\} \quad (12.52)$$

where the n^{th} row of the matrix \mathbf{Z} is given by \mathbf{z}_n . We already know that the exact maximum likelihood solution for $\boldsymbol{\mu}$ is given by the sample mean $\bar{\mathbf{x}}$ defined by (12.1), and it is convenient to substitute for $\boldsymbol{\mu}$ at this stage. Making use of the expressions (12.31) and (12.32) for the latent and conditional distributions, respectively, and taking the expectation with respect to the posterior distribution over the latent variables, we obtain

$$\begin{aligned} \mathbb{E}[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2)] &= -\sum_{n=1}^N \left\{ \frac{D}{2} \ln(2\pi\sigma^2) + \frac{1}{2} \text{Tr}(\mathbb{E}[\mathbf{z}_n \mathbf{z}_n^T]) \right. \\ &\quad + \frac{1}{2\sigma^2} \|\mathbf{x}_n - \boldsymbol{\mu}\|^2 - \frac{1}{\sigma^2} \mathbb{E}[\mathbf{z}_n]^T \mathbf{W}^T (\mathbf{x}_n - \boldsymbol{\mu}) \\ &\quad \left. + \frac{1}{2\sigma^2} \text{Tr}(\mathbb{E}[\mathbf{z}_n \mathbf{z}_n^T] \mathbf{W}^T \mathbf{W}) \right\}. \end{aligned} \quad (12.53)$$

Note that this depends on the posterior distribution only through the sufficient statistics of the Gaussian. Thus in the E step, we use the old parameter values to evaluate

$$\mathbb{E}[\mathbf{z}_n] = \mathbf{M}^{-1} \mathbf{W}^T (\mathbf{x}_n - \bar{\mathbf{x}}) \quad (12.54)$$

$$\mathbb{E}[\mathbf{z}_n \mathbf{z}_n^T] = \sigma^2 \mathbf{M}^{-1} + \mathbb{E}[\mathbf{z}_n] \mathbb{E}[\mathbf{z}_n]^T \quad (12.55)$$

which follow directly from the posterior distribution (12.42) together with the standard result $\mathbb{E}[\mathbf{z}_n \mathbf{z}_n^T] = \text{cov}[\mathbf{z}_n] + \mathbb{E}[\mathbf{z}_n] \mathbb{E}[\mathbf{z}_n]^T$. Here \mathbf{M} is defined by (12.41).

In the M step, we maximize with respect to \mathbf{W} and σ^2 , keeping the posterior statistics fixed. Maximization with respect to σ^2 is straightforward. For the maximization with respect to \mathbf{W} we make use of (C.24), and obtain the M-step equations

$$\mathbf{W}_{\text{new}} = \left[\sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}}) \mathbb{E}[\mathbf{z}_n]^T \right] \left[\sum_{n=1}^N \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^T] \right]^{-1} \quad (12.56)$$

$$\begin{aligned} \sigma_{\text{new}}^2 &= \frac{1}{ND} \sum_{n=1}^N \left\{ \|\mathbf{x}_n - \bar{\mathbf{x}}\|^2 - 2\mathbb{E}[\mathbf{z}_n]^T \mathbf{W}_{\text{new}}^T (\mathbf{x}_n - \bar{\mathbf{x}}) \right. \\ &\quad \left. + \text{Tr}(\mathbb{E}[\mathbf{z}_n \mathbf{z}_n^T] \mathbf{W}_{\text{new}}^T \mathbf{W}_{\text{new}}) \right\}. \end{aligned} \quad (12.57)$$

The EM algorithm for probabilistic PCA proceeds by initializing the parameters and then alternately computing the sufficient statistics of the latent space posterior distribution using (12.54) and (12.55) in the E step and revising the parameter values using (12.56) and (12.57) in the M step.

One of the benefits of the EM algorithm for PCA is computational efficiency for large-scale applications (Roweis, 1998). Unlike conventional PCA based on an

Exercise 12.15

eigenvector decomposition of the sample covariance matrix, the EM approach is iterative and so might appear to be less attractive. However, each cycle of the EM algorithm can be computationally much more efficient than conventional PCA in spaces of high dimensionality. To see this, we note that the eigendecomposition of the covariance matrix requires $O(D^3)$ computation. Often we are interested only in the first M eigenvectors and their corresponding eigenvalues, in which case we can use algorithms that are $O(MD^2)$. However, the evaluation of the covariance matrix itself takes $O(ND^2)$ computations, where N is the number of data points. Algorithms such as the snapshot method (Sirovich, 1987), which assume that the eigenvectors are linear combinations of the data vectors, avoid direct evaluation of the covariance matrix but are $O(N^3)$ and hence unsuited to large data sets. The EM algorithm described here also does not construct the covariance matrix explicitly. Instead, the most computationally demanding steps are those involving sums over the data set that are $O(NDM)$. For large D , and $M \ll D$, this can be a significant saving compared to $O(ND^2)$ and can offset the iterative nature of the EM algorithm.

Note that this EM algorithm can be implemented in an on-line form in which each D -dimensional data point is read in and processed and then discarded before the next data point is considered. To see this, note that the quantities evaluated in the E step (an M -dimensional vector and an $M \times M$ matrix) can be computed for each data point separately, and in the M step we need to accumulate sums over data points, which we can do incrementally. This approach can be advantageous if both N and D are large.

Because we now have a fully probabilistic model for PCA, we can deal with missing data, provided that it is missing at random, by marginalizing over the distribution of the unobserved variables. Again these missing values can be treated using the EM algorithm. We give an example of the use of this approach for data visualization in Figure 12.11.

Another elegant feature of the EM approach is that we can take the limit $\sigma^2 \rightarrow 0$, corresponding to standard PCA, and still obtain a valid EM-like algorithm (Roweis, 1998). From (12.55), we see that the only quantity we need to compute in the E step is $\mathbb{E}[\mathbf{z}_n]$. Furthermore, the M step is simplified because $\mathbf{M} = \mathbf{W}^T \mathbf{W}$. To emphasize the simplicity of the algorithm, let us define $\tilde{\mathbf{X}}$ to be a matrix of size $N \times D$ whose n^{th} row is given by the vector $\mathbf{x}_n - \bar{\mathbf{x}}$ and similarly define Ω to be a matrix of size $D \times M$ whose n^{th} row is given by the vector $\mathbb{E}[\mathbf{z}_n]$. The E step (12.54) of the EM algorithm for PCA then becomes

$$\Omega = (\mathbf{W}_{\text{old}}^T \mathbf{W}_{\text{old}})^{-1} \mathbf{W}_{\text{old}}^T \tilde{\mathbf{X}} \quad (12.58)$$

and the M step (12.56) takes the form

$$\mathbf{W}_{\text{new}} = \tilde{\mathbf{X}}^T \Omega^T (\Omega \Omega^T)^{-1}. \quad (12.59)$$

Again these can be implemented in an on-line form. These equations have a simple interpretation as follows. From our earlier discussion, we see that the E step involves an orthogonal projection of the data points onto the current estimate for the principal subspace. Correspondingly, the M step represents a re-estimation of the principal

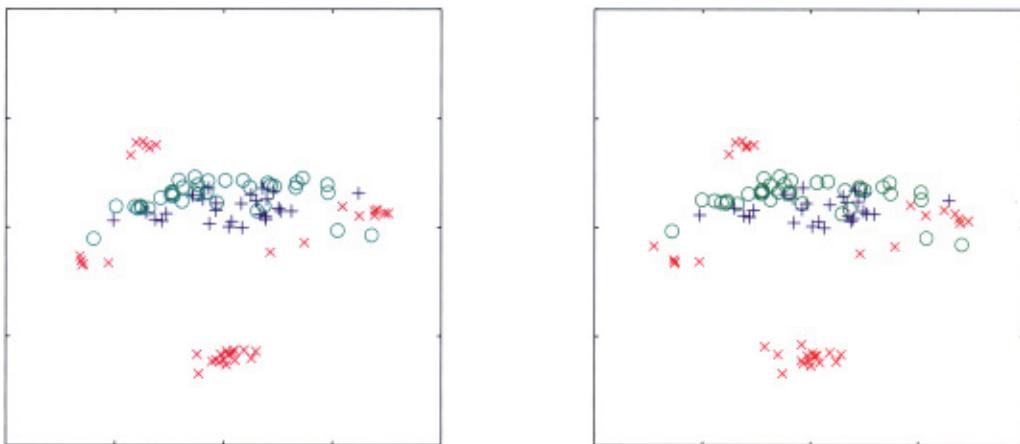


Figure 12.11 Probabilistic PCA visualization of a portion of the oil flow data set for the first 100 data points. The left-hand plot shows the posterior mean projections of the data points on the principal subspace. The right-hand plot is obtained by first randomly omitting 30% of the variable values and then using EM to handle the missing values. Note that each data point then has at least one missing measurement but that the plot is very similar to the one obtained without missing values.

Exercise 12.17 subspace to minimize the squared reconstruction error in which the projections are fixed.

We can give a simple physical analogy for this EM algorithm, which is easily visualized for $D = 2$ and $M = 1$. Consider a collection of data points in two dimensions, and let the one-dimensional principal subspace be represented by a solid rod. Now attach each data point to the rod via a spring obeying Hooke's law (stored energy is proportional to the square of the spring's length). In the E step, we keep the rod fixed and allow the attachment points to slide up and down the rod so as to minimize the energy. This causes each attachment point (independently) to position itself at the orthogonal projection of the corresponding data point onto the rod. In the M step, we keep the attachment points fixed and then release the rod and allow it to move to the minimum energy position. The E and M steps are then repeated until a suitable convergence criterion is satisfied, as is illustrated in Figure 12.12.

12.2.3 Bayesian PCA

So far in our discussion of PCA, we have assumed that the value M for the dimensionality of the principal subspace is given. In practice, we must choose a suitable value according to the application. For visualization, we generally choose $M = 2$, whereas for other applications the appropriate choice for M may be less clear. One approach is to plot the eigenvalue spectrum for the data set, analogous to the example in Figure 12.4 for the off-line digits data set, and look to see if the eigenvalues naturally form two groups comprising a set of small values separated by a significant gap from a set of relatively large values, indicating a natural choice for M . In practice, such a gap is often not seen.

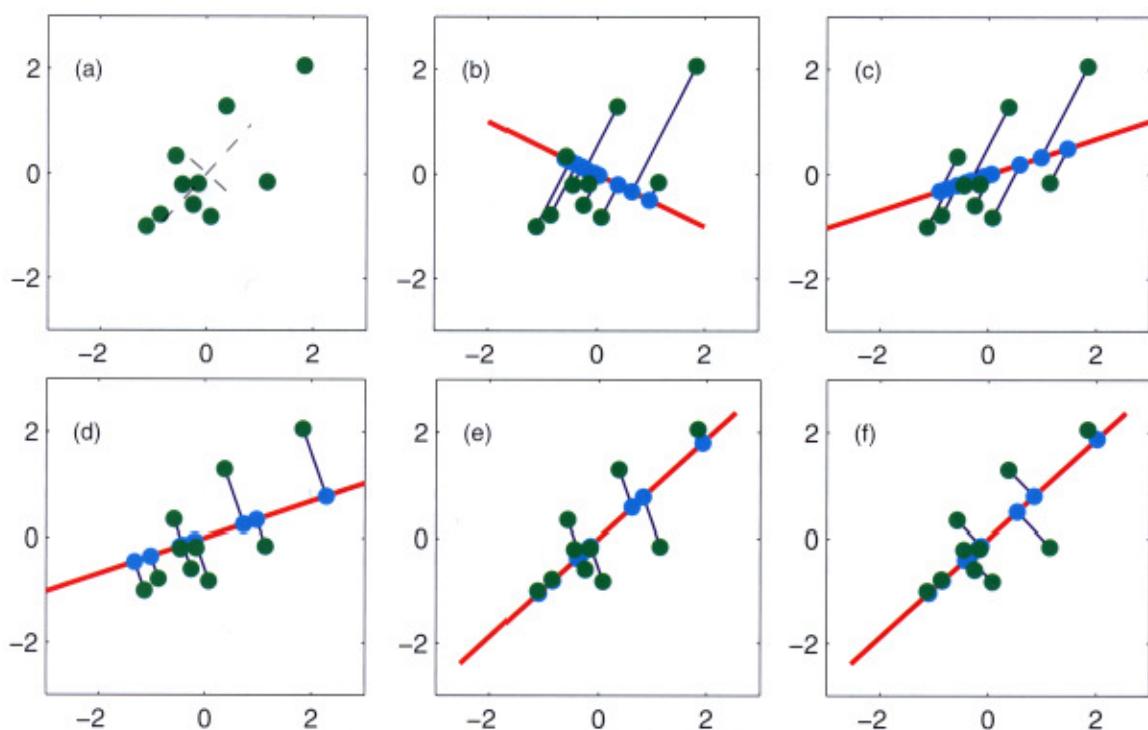


Figure 12.12 Synthetic data illustrating the EM algorithm for PCA defined by (12.58) and (12.59). (a) A data set X with the data points shown in green, together with the true principal components (shown as eigenvectors scaled by the square roots of the eigenvalues). (b) Initial configuration of the principal subspace defined by W , shown in red, together with the projections of the latent points Z into the data space, given by ZW^T , shown in cyan. (c) After one M step, the latent space has been updated with Z held fixed. (d) After the successive E step, the values of Z have been updated, giving orthogonal projections, with W held fixed. (e) After the second M step. (f) After the second E step.

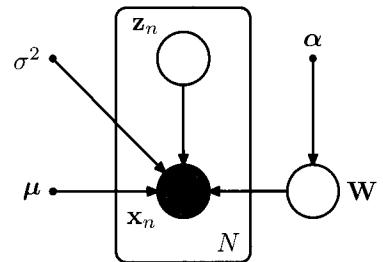
Section 1.3

Because the probabilistic PCA model has a well-defined likelihood function, we could employ cross-validation to determine the value of dimensionality by selecting the largest log likelihood on a validation data set. Such an approach, however, can become computationally costly, particularly if we consider a probabilistic mixture of PCA models (Tipping and Bishop, 1999a) in which we seek to determine the appropriate dimensionality separately for each component in the mixture.

Given that we have a probabilistic formulation of PCA, it seems natural to seek a Bayesian approach to model selection. To do this, we need to marginalize out the model parameters μ , W , and σ^2 with respect to appropriate prior distributions. This can be done by using a variational framework to approximate the analytically intractable marginalizations (Bishop, 1999b). The marginal likelihood values, given by the variational lower bound, can then be compared for a range of different values of M and the value giving the largest marginal likelihood selected.

Here we consider a simpler approach introduced by based on the *evidence ap-*

Figure 12.13 Probabilistic graphical model for Bayesian PCA in which the distribution over the parameter matrix \mathbf{W} is governed by a vector α of hyperparameters.



proximation, which is appropriate when the number of data points is relatively large and the corresponding posterior distribution is tightly peaked (Bishop, 1999a). It involves a specific choice of prior over \mathbf{W} that allows surplus dimensions in the principal subspace to be pruned out of the model. This corresponds to an example of *automatic relevance determination*, or *ARD*, discussed in Section 7.2.2. Specifically, we define an independent Gaussian prior over each column of \mathbf{W} , which represent the vectors defining the principal subspace. Each such Gaussian has an independent variance governed by a precision hyperparameter α_i so that

$$p(\mathbf{W}|\alpha) = \prod_{i=1}^M \left(\frac{\alpha_i}{2\pi} \right)^{D/2} \exp \left\{ -\frac{1}{2} \alpha_i \mathbf{w}_i^\top \mathbf{w}_i \right\} \quad (12.60)$$

where \mathbf{w}_i is the i^{th} column of \mathbf{W} . The resulting model can be represented using the directed graph shown in Figure 12.13.

The values for α_i will be found iteratively by maximizing the marginal likelihood function in which \mathbf{W} has been integrated out. As a result of this optimization, some of the α_i may be driven to infinity, with the corresponding parameters vector \mathbf{w}_i being driven to zero (the posterior distribution becomes a delta function at the origin) giving a sparse solution. The effective dimensionality of the principal subspace is then determined by the number of finite α_i values, and the corresponding vectors \mathbf{w}_i can be thought of as ‘relevant’ for modelling the data distribution. In this way, the Bayesian approach is automatically making the trade-off between improving the fit to the data, by using a larger number of vectors \mathbf{w}_i with their corresponding eigenvalues λ_i each tuned to the data, and reducing the complexity of the model by suppressing some of the \mathbf{w}_i vectors. The origins of this sparsity were discussed earlier in the context of relevance vector machines.

Section 7.2

The values of α_i are re-estimated during training by maximizing the log marginal likelihood given by

$$p(\mathbf{X}|\alpha, \mu, \sigma^2) = \int p(\mathbf{X}|\mathbf{W}, \mu, \sigma^2)p(\mathbf{W}|\alpha) d\mathbf{W} \quad (12.61)$$

where the log of $p(\mathbf{X}|\mathbf{W}, \mu, \sigma^2)$ is given by (12.43). Note that for simplicity we also treat μ and σ^2 as parameters to be estimated, rather than defining priors over these parameters.

Section 4.4

Section 3.5.3

Because this integration is intractable, we make use of the Laplace approximation. If we assume that the posterior distribution is sharply peaked, as will occur for sufficiently large data sets, then the re-estimation equations obtained by maximizing the marginal likelihood with respect to α_i take the simple form

$$\alpha_i^{\text{new}} = \frac{D}{\mathbf{w}_i^T \mathbf{w}_i} \quad (12.62)$$

which follows from (3.98), noting that the dimensionality of \mathbf{w}_i is D . These re-estimations are interleaved with the EM algorithm updates for determining \mathbf{W} and σ^2 . The E-step equations are again given by (12.54) and (12.55). Similarly, the M-step equation for σ^2 is again given by (12.57). The only change is to the M-step equation for \mathbf{W} , which is modified to give

$$\mathbf{W}_{\text{new}} = \left[\sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}}) \mathbb{E}[\mathbf{z}_n]^T \right] \left[\sum_{n=1}^N \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^T] + \sigma^2 \mathbf{A} \right]^{-1} \quad (12.63)$$

where $\mathbf{A} = \text{diag}(\alpha_i)$. The value of μ is given by the sample mean, as before.

If we choose $M = D - 1$ then, if all α_i values are finite, the model represents a full-covariance Gaussian, while if all the α_i go to infinity the model is equivalent to an isotropic Gaussian, and so the model can encompass all permissible values for the effective dimensionality of the principal subspace. It is also possible to consider smaller values of M , which will save on computational cost but which will limit the maximum dimensionality of the subspace. A comparison of the results of this algorithm with standard probabilistic PCA is shown in Figure 12.14.

Bayesian PCA provides an opportunity to illustrate the Gibbs sampling algorithm discussed in Section 11.3. Figure 12.15 shows an example of the samples from the hyperparameters $\ln \alpha_i$ for a data set in $D = 4$ dimensions in which the dimensionality of the latent space is $M = 3$ but in which the data set is generated from a probabilistic PCA model having one direction of high variance, with the remaining directions comprising low variance noise. This result shows clearly the presence of three distinct modes in the posterior distribution. At each step of the iteration, one of the hyperparameters has a small value and the remaining two have large values, so that two of the three latent variables are suppressed. During the course of the Gibbs sampling, the solution makes sharp transitions between the three modes.

The model described here involves a prior only over the matrix \mathbf{W} . A fully Bayesian treatment of PCA, including priors over μ , σ^2 , and α , and solved using variational methods, is described in Bishop (1999b). For a discussion of various Bayesian approaches to determining the appropriate dimensionality for a PCA model, see Minka (2001c).

12.2.4 Factor analysis

Factor analysis is a linear-Gaussian latent variable model that is closely related to probabilistic PCA. Its definition differs from that of probabilistic PCA only in that the conditional distribution of the observed variable \mathbf{x} given the latent variable \mathbf{z} is

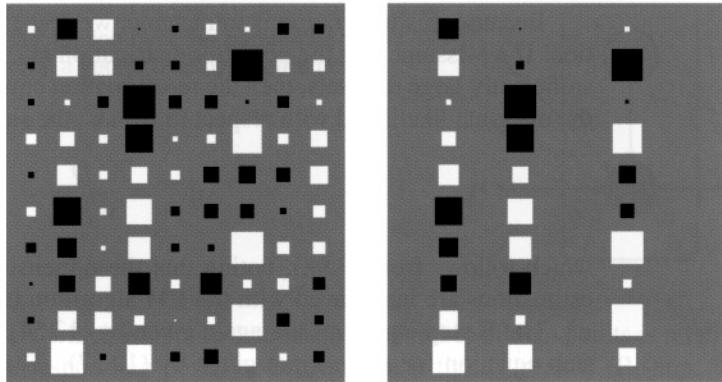


Figure 12.14 ‘Hinton’ diagrams of the matrix \mathbf{W} in which each element of the matrix is depicted as a square (white for positive and black for negative values) whose area is proportional to the magnitude of that element. The synthetic data set comprises 300 data points in $D = 10$ dimensions sampled from a Gaussian distribution having standard deviation 1.0 in 3 directions and standard deviation 0.5 in the remaining 7 directions for a data set in $D = 10$ dimensions having $M = 3$ directions with larger variance than the remaining 7 directions. The left-hand plot shows the result from maximum likelihood probabilistic PCA, and the left-hand plot shows the corresponding result from Bayesian PCA. We see how the Bayesian model is able to discover the appropriate dimensionality by suppressing the 6 surplus degrees of freedom.

taken to have a diagonal rather than an isotropic covariance so that

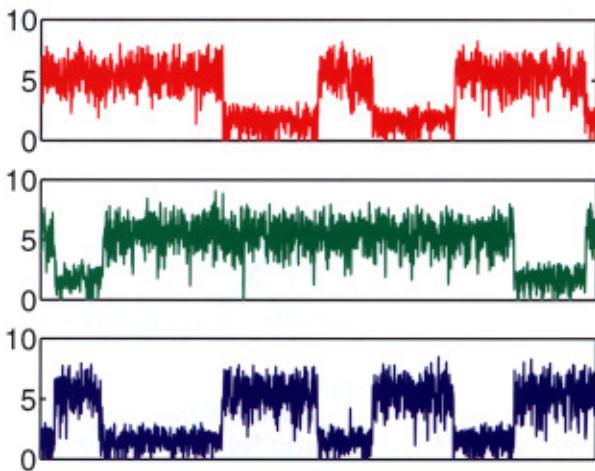
$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \quad (12.64)$$

where $\boldsymbol{\Psi}$ is a $D \times D$ diagonal matrix. Note that the factor analysis model, in common with probabilistic PCA, assumes that the observed variables x_1, \dots, x_D are independent, given the latent variable \mathbf{z} . In essence, the factor analysis model is explaining the observed covariance structure of the data by representing the independent variance associated with each coordinate in the matrix $\boldsymbol{\Psi}$ and capturing the covariance between variables in the matrix \mathbf{W} . In the factor analysis literature, the columns of \mathbf{W} , which capture the correlations between observed variables, are called *factor loadings*, and the diagonal elements of $\boldsymbol{\Psi}$, which represent the independent noise variances for each of the variables, are called *uniquenesses*.

The origins of factor analysis are as old as those of PCA, and discussions of factor analysis can be found in the books by Everitt (1984), Bartholomew (1987), and Basilevsky (1994). Links between factor analysis and PCA were investigated by Lawley (1953) and Anderson (1963) who showed that at stationary points of the likelihood function, for a factor analysis model with $\boldsymbol{\Psi} = \sigma^2 \mathbf{I}$, the columns of \mathbf{W} are scaled eigenvectors of the sample covariance matrix, and σ^2 is the average of the discarded eigenvalues. Later, Tipping and Bishop (1999b) showed that the maximum of the log likelihood function occurs when the eigenvectors comprising \mathbf{W} are chosen to be the principal eigenvectors.

Making use of (2.115), we see that the marginal distribution for the observed

Figure 12.15 Gibbs sampling for Bayesian PCA showing plots of $\ln \alpha_i$ versus iteration number for three α values, showing transitions between the three modes of the posterior distribution.



variable is given by $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{C})$ where now

$$\mathbf{C} = \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi}. \quad (12.65)$$

Exercise 12.19

As with probabilistic PCA, this model is invariant to rotations in the latent space.

Historically, factor analysis has been the subject of controversy when attempts have been made to place an interpretation on the individual factors (the coordinates in \mathbf{z} -space), which has proven problematic due to the nonidentifiability of factor analysis associated with rotations in this space. From our perspective, however, we shall view factor analysis as a form of latent variable density model, in which the form of the latent space is of interest but not the particular choice of coordinates used to describe it. If we wish to remove the degeneracy associated with latent space rotations, we must consider non-Gaussian latent variable distributions, giving rise to independent component analysis (ICA) models.

We can determine the parameters $\boldsymbol{\mu}$, \mathbf{W} , and $\boldsymbol{\Psi}$ in the factor analysis model by maximum likelihood. The solution for $\boldsymbol{\mu}$ is again given by the sample mean. However, unlike probabilistic PCA, there is no longer a closed-form maximum likelihood solution for \mathbf{W} , which must therefore be found iteratively. Because factor analysis is a latent variable model, this can be done using an EM algorithm (Rubin and Thayer, 1982) that is analogous to the one used for probabilistic PCA. Specifically, the E-step equations are given by

$$\mathbb{E}[\mathbf{z}_n] = \mathbf{G}\mathbf{W}^T\boldsymbol{\Psi}^{-1}(\mathbf{x}_n - \bar{\mathbf{x}}) \quad (12.66)$$

$$\mathbb{E}[\mathbf{z}_n \mathbf{z}_n^T] = \mathbf{G} + \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^T \quad (12.67)$$

where we have defined

$$\mathbf{G} = (\mathbf{I} + \mathbf{W}^T\boldsymbol{\Psi}^{-1}\mathbf{W})^{-1}. \quad (12.68)$$

Note that this is expressed in a form that involves inversion of matrices of size $M \times M$ rather than $D \times D$ (except for the $D \times D$ diagonal matrix $\boldsymbol{\Psi}$ whose inverse is trivial

Section 12.4

Exercise 12.21

to compute in $O(D)$ steps), which is convenient because often $M \ll D$. Similarly, the M-step equations take the form

$$\mathbf{W}^{\text{new}} = \left[\sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}}) \mathbb{E}[\mathbf{z}_n]^T \right] \left[\sum_{n=1}^N \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^T] \right]^{-1} \quad (12.69)$$

$$\boldsymbol{\Psi}^{\text{new}} = \text{diag} \left\{ \mathbf{S} - \mathbf{W}_{\text{new}} \frac{1}{N} \sum_{n=1}^N \mathbb{E}[\mathbf{z}_n] (\mathbf{x}_n - \bar{\mathbf{x}})^T \right\} \quad (12.70)$$

where the ‘diag’ operator sets all of the nondiagonal elements of a matrix to zero. A Bayesian treatment of the factor analysis model can be obtained by a straightforward application of the techniques discussed in this book.

Exercise 12.25

Another difference between probabilistic PCA and factor analysis concerns their different behaviour under transformations of the data set. For PCA and probabilistic PCA, if we rotate the coordinate system in data space, then we obtain exactly the same fit to the data but with the \mathbf{W} matrix transformed by the corresponding rotation matrix. However, for factor analysis, the analogous property is that if we make a component-wise re-scaling of the data vectors, then this is absorbed into a corresponding re-scaling of the elements of $\boldsymbol{\Psi}$.

12.3. Kernel PCA

In Chapter 6, we saw how the technique of kernel substitution allows us to take an algorithm expressed in terms of scalar products of the form $\mathbf{x}^T \mathbf{x}'$ and generalize that algorithm by replacing the scalar products with a nonlinear kernel. Here we apply this technique of kernel substitution to principal component analysis, thereby obtaining a nonlinear generalization called *kernel PCA* (Schölkopf *et al.*, 1998).

Consider a data set $\{\mathbf{x}_n\}$ of observations, where $n = 1, \dots, N$, in a space of dimensionality D . In order to keep the notation uncluttered, we shall assume that we have already subtracted the sample mean from each of the vectors \mathbf{x}_n , so that $\sum_n \mathbf{x}_n = \mathbf{0}$. The first step is to express conventional PCA in such a form that the data vectors $\{\mathbf{x}_n\}$ appear only in the form of the scalar products $\mathbf{x}_n^T \mathbf{x}_m$. Recall that the principal components are defined by the eigenvectors \mathbf{u}_i of the covariance matrix

$$\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i \quad (12.71)$$

where $i = 1, \dots, D$. Here the $D \times D$ sample covariance matrix \mathbf{S} is defined by

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T, \quad (12.72)$$

and the eigenvectors are normalized such that $\mathbf{u}_i^T \mathbf{u}_i = 1$.

Now consider a nonlinear transformation $\phi(\mathbf{x})$ into an M -dimensional feature space, so that each data point \mathbf{x}_n is thereby projected onto a point $\phi(\mathbf{x}_n)$. We can

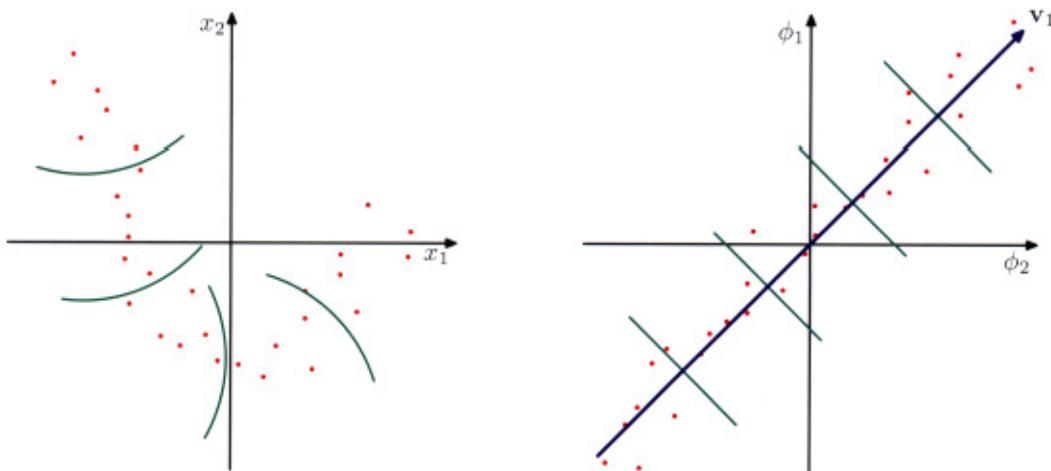


Figure 12.16 Schematic illustration of kernel PCA. A data set in the original data space (left-hand plot) is projected by a nonlinear transformation $\phi(\mathbf{x})$ into a feature space (right-hand plot). By performing PCA in the feature space, we obtain the principal components, of which the first is shown in blue and is denoted by the vector \mathbf{v}_1 . The green lines in feature space indicate the linear projections onto the first principal component, which correspond to nonlinear projections in the original data space. Note that in general it is not possible to represent the nonlinear principal component by a vector in \mathbf{x} space.

now perform standard PCA in the feature space, which implicitly defines a nonlinear principal component model in the original data space, as illustrated in Figure 12.16.

For the moment, let us assume that the projected data set also has zero mean, so that $\sum_n \phi(\mathbf{x}_n) = 0$. We shall return to this point shortly. The $M \times M$ sample covariance matrix in feature space is given by

$$\mathbf{C} = \frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^T \quad (12.73)$$

and its eigenvector expansion is defined by

$$\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i \quad (12.74)$$

$i = 1, \dots, M$. Our goal is to solve this eigenvalue problem without having to work explicitly in the feature space. From the definition of \mathbf{C} , the eigenvector equations tells us that \mathbf{v}_i satisfies

$$\frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \{\phi(\mathbf{x}_n)^T \mathbf{v}_i\} = \lambda_i \mathbf{v}_i \quad (12.75)$$

and so we see that (provided $\lambda_i > 0$) the vector \mathbf{v}_i is given by a linear combination of the $\phi(\mathbf{x}_n)$ and so can be written in the form

$$\mathbf{v}_i = \sum_{n=1}^N a_{in} \phi(\mathbf{x}_n). \quad (12.76)$$

Substituting this expansion back into the eigenvector equation, we obtain

$$\frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^T \sum_{m=1}^N a_{im} \phi(\mathbf{x}_m) = \lambda_i \sum_{n=1}^N a_{in} \phi(\mathbf{x}_n). \quad (12.77)$$

The key step is now to express this in terms of the kernel function $k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$, which we do by multiplying both sides by $\phi(\mathbf{x}_l)^T$ to give

$$\frac{1}{N} \sum_{n=1}^N k(\mathbf{x}_l, \mathbf{x}_n) \sum_{m=1}^m a_{im} k(\mathbf{x}_n, \mathbf{x}_m) = \lambda_i \sum_{n=1}^N a_{in} k(\mathbf{x}_l, \mathbf{x}_n). \quad (12.78)$$

This can be written in matrix notation as

$$\mathbf{K}^2 \mathbf{a}_i = \lambda_i N \mathbf{K} \mathbf{a}_i \quad (12.79)$$

where \mathbf{a}_i is an N -dimensional column vector with elements a_{ni} for $n = 1, \dots, N$. We can find solutions for \mathbf{a}_i by solving the following eigenvalue problem

$$\mathbf{K} \mathbf{a}_i = \lambda_i N \mathbf{a}_i \quad (12.80)$$

in which we have removed a factor of \mathbf{K} from both sides of (12.79). Note that the solutions of (12.79) and (12.80) differ only by eigenvectors of \mathbf{K} having zero eigenvalues that do not affect the principal components projection.

Exercise 12.26

The normalization condition for the coefficients \mathbf{a}_i is obtained by requiring that the eigenvectors in feature space be normalized. Using (12.76) and (12.80), we have

$$1 = \mathbf{v}_i^T \mathbf{v}_i = \sum_{n=1}^N \sum_{m=1}^N a_{in} a_{im} \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) = \mathbf{a}_i^T \mathbf{K} \mathbf{a}_i = \lambda_i N \mathbf{a}_i^T \mathbf{a}_i. \quad (12.81)$$

Having solved the eigenvector problem, the resulting principal component projections can then also be cast in terms of the kernel function so that, using (12.76), the projection of a point \mathbf{x} onto eigenvector i is given by

$$y_i(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{v}_i = \sum_{n=1}^N a_{in} \phi(\mathbf{x})^T \phi(\mathbf{x}_n) = \sum_{n=1}^N a_{in} k(\mathbf{x}, \mathbf{x}_n) \quad (12.82)$$

and so again is expressed in terms of the kernel function.

In the original D -dimensional \mathbf{x} space there are D orthogonal eigenvectors and hence we can find at most D linear principal components. The dimensionality M of the feature space, however, can be much larger than D (even infinite), and thus we can find a number of nonlinear principal components that can exceed D . Note, however, that the number of nonzero eigenvalues cannot exceed the number N of data points, because (even if $M > N$) the covariance matrix in feature space has rank at most equal to N . This is reflected in the fact that kernel PCA involves the eigenvector expansion of the $N \times N$ matrix \mathbf{K} .

So far we have assumed that the projected data set given by $\phi(\mathbf{x}_n)$ has zero mean, which in general will not be the case. We cannot simply compute and then subtract off the mean, since we wish to avoid working directly in feature space, and so again, we formulate the algorithm purely in terms of the kernel function. The projected data points after centralizing, denoted $\tilde{\phi}(\mathbf{x}_n)$, are given by

$$\tilde{\phi}(\mathbf{x}_n) = \phi(\mathbf{x}_n) - \frac{1}{N} \sum_{l=1}^N \phi(\mathbf{x}_l) \quad (12.83)$$

and the corresponding elements of the Gram matrix are given by

$$\begin{aligned} \tilde{K}_{nm} &= \tilde{\phi}(\mathbf{x}_n)^T \tilde{\phi}(\mathbf{x}_m) \\ &= \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) - \frac{1}{N} \sum_{l=1}^N \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_l) \\ &\quad - \frac{1}{N} \sum_{l=1}^N \phi(\mathbf{x}_l)^T \phi(\mathbf{x}_m) + \frac{1}{N^2} \sum_{j=1}^N \sum_{l=1}^N \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_l) \\ &= k(\mathbf{x}_n, \mathbf{x}_m) - \frac{1}{N} \sum_{l=1}^N k(\mathbf{x}_l, \mathbf{x}_m) \\ &\quad - \frac{1}{N} \sum_{l=1}^N k(\mathbf{x}_n, \mathbf{x}_l) + \frac{1}{N^2} \sum_{j=1}^N \sum_{l=1}^N k(\mathbf{x}_j, \mathbf{x}_l). \end{aligned} \quad (12.84)$$

This can be expressed in matrix notation as

$$\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N \mathbf{K} - \mathbf{K} \mathbf{1}_N + \mathbf{1}_N \mathbf{K} \mathbf{1}_N \quad (12.85)$$

where $\mathbf{1}_N$ denotes the $N \times N$ matrix in which every element takes the value $1/N$. Thus we can evaluate $\tilde{\mathbf{K}}$ using only the kernel function and then use $\tilde{\mathbf{K}}$ to determine the eigenvalues and eigenvectors. Note that the standard PCA algorithm is recovered as a special case if we use a linear kernel $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$. Figure 12.17 shows an example of kernel PCA applied to a synthetic data set (Schölkopf *et al.*, 1998). Here a ‘Gaussian’ kernel of the form

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 0.1) \quad (12.86)$$

is applied to a synthetic data set. The lines correspond to contours along which the projection onto the corresponding principal component, defined by

$$\phi(\mathbf{x})^T \mathbf{v}_i = \sum_{n=1}^N a_{in} k(\mathbf{x}, \mathbf{x}_n) \quad (12.87)$$

is constant.

Exercise 12.27

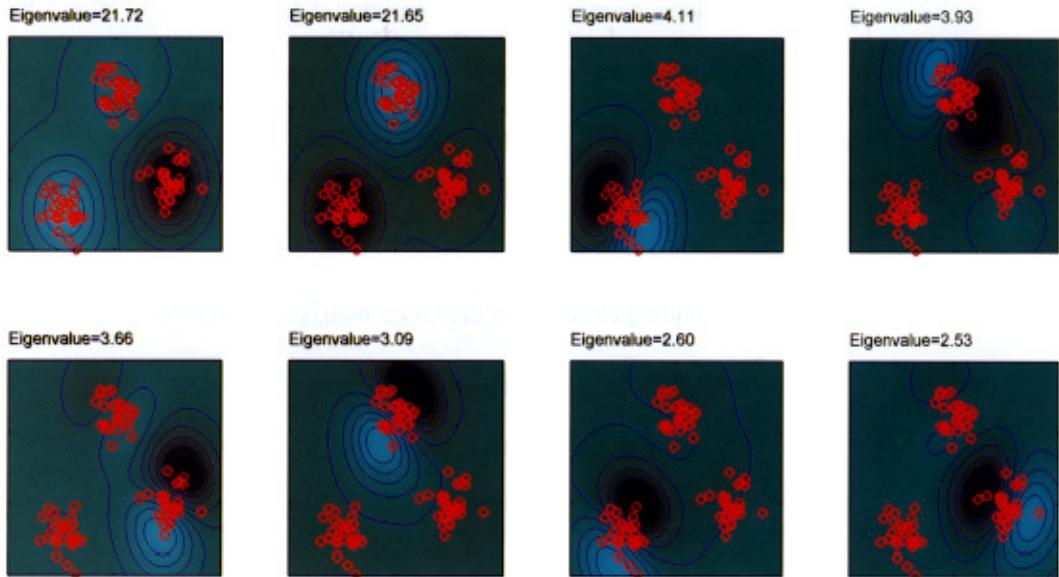


Figure 12.17 Example of kernel PCA, with a Gaussian kernel applied to a synthetic data set in two dimensions, showing the first eight eigenfunctions along with their eigenvalues. The contours are lines along which the projection onto the corresponding principal component is constant. Note how the first two eigenvectors separate the three clusters, the next three eigenvectors split each of the cluster into halves, and the following three eigenvectors again split the clusters into halves along directions orthogonal to the previous splits.

One obvious disadvantage of kernel PCA is that it involves finding the eigenvectors of the $N \times N$ matrix $\tilde{\mathbf{K}}$ rather than the $D \times D$ matrix \mathbf{S} of conventional linear PCA, and so in practice for large data sets approximations are often used.

Finally, we note that in standard linear PCA, we often retain some reduced number $L < D$ of eigenvectors and then approximate a data vector \mathbf{x}_n by its projection $\hat{\mathbf{x}}_n$ onto the L -dimensional principal subspace, defined by

$$\hat{\mathbf{x}}_n = \sum_{i=1}^L (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i. \quad (12.88)$$

In kernel PCA, this will in general not be possible. To see this, note that the mapping $\phi(\mathbf{x})$ maps the D -dimensional \mathbf{x} space into a D -dimensional *manifold* in the M -dimensional feature space ϕ . The vector \mathbf{x} is known as the *pre-image* of the corresponding point $\phi(\mathbf{x})$. However, the projection of points in feature space onto the linear PCA subspace in that space will typically not lie on the nonlinear D -dimensional manifold and so will not have a corresponding pre-image in data space. Techniques have therefore been proposed for finding approximate pre-images (Bakir *et al.*, 2004).

12.4. Nonlinear Latent Variable Models

In this chapter, we have focussed on the simplest class of models having continuous latent variables, namely those based on linear-Gaussian distributions. As well as having great practical importance, these models are relatively easy to analyse and to fit to data and can also be used as components in more complex models. Here we consider briefly some generalizations of this framework to models that are either nonlinear or non-Gaussian, or both.

In fact, the issues of nonlinearity and non-Gaussianity are related because a general probability density can be obtained from a simple fixed reference density, such as a Gaussian, by making a nonlinear change of variables. This idea forms the basis of several practical latent variable models as we shall see shortly.

Exercise 12.28

12.4.1 Independent component analysis

We begin by considering models in which the observed variables are related linearly to the latent variables, but for which the latent distribution is non-Gaussian. An important class of such models, known as *independent component analysis*, or *ICA*, arises when we consider a distribution over the latent variables that factorizes, so that

$$p(\mathbf{z}) = \prod_{j=1}^M p(z_j). \quad (12.89)$$

To understand the role of such models, consider a situation in which two people are talking at the same time, and we record their voices using two microphones. If we ignore effects such as time delay and echoes, then the signals received by the microphones at any point in time will be given by linear combinations of the amplitudes of the two voices. The coefficients of this linear combination will be constant, and if we can infer their values from sample data, then we can invert the mixing process (assuming it is nonsingular) and thereby obtain two clean signals each of which contains the voice of just one person. This is an example of a problem called *blind source separation* in which ‘blind’ refers to the fact that we are given only the mixed data, and neither the original sources nor the mixing coefficients are observed (Cardoso, 1998).

This type of problem is sometimes addressed using the following approach (MacKay, 2003) in which we ignore the temporal nature of the signals and treat the successive samples as i.i.d. We consider a generative model in which there are two latent variables corresponding to the unobserved speech signal amplitudes, and there are two observed variables given by the signal values at the microphones. The latent variables have a joint distribution that factorizes as above, and the observed variables are given by a linear combination of the latent variables. There is no need to include a noise distribution because the number of latent variables equals the number of observed variables, and therefore the marginal distribution of the observed variables will not in general be singular, so the observed variables are simply deterministic linear combinations of the latent variables. Given a data set of observations, the

likelihood function for this model is a function of the coefficients in the linear combination. The log likelihood can be maximized using gradient-based optimization giving rise to a particular version of independent component analysis.

The success of this approach requires that the latent variables have non-Gaussian distributions. To see this, recall that in probabilistic PCA (and in factor analysis) the latent-space distribution is given by a zero-mean isotropic Gaussian. The model therefore cannot distinguish between two different choices for the latent variables where these differ simply by a rotation in latent space. This can be verified directly by noting that the marginal density (12.35), and hence the likelihood function, is unchanged if we make the transformation $\mathbf{W} \rightarrow \mathbf{WR}$ where \mathbf{R} is an orthogonal matrix satisfying $\mathbf{RR}^T = \mathbf{I}$, because the matrix \mathbf{C} given by (12.36) is itself invariant. Extending the model to allow more general Gaussian latent distributions does not change this conclusion because, as we have seen, such a model is equivalent to the zero-mean isotropic Gaussian latent variable model.

Another way to see why a Gaussian latent variable distribution in a linear model is insufficient to find independent components is to note that the principal components represent a rotation of the coordinate system in data space such as to diagonalize the covariance matrix, so that the data distribution in the new coordinates is then uncorrelated. Although zero correlation is a necessary condition for independence it is not, however, sufficient. In practice, a common choice for the latent-variable distribution is given by

$$p(z_j) = \frac{1}{\pi \cosh(z_j)} = \frac{1}{\pi(e^{z_j} + e^{-z_j})} \quad (12.90)$$

which has heavy tails compared to a Gaussian, reflecting the observation that many real-world distributions also exhibit this property.

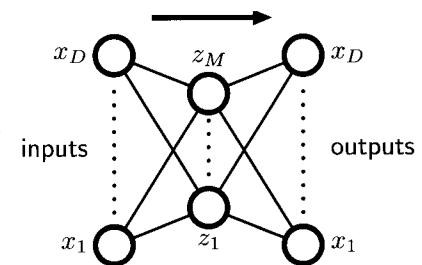
The original ICA model (Bell and Sejnowski, 1995) was based on the optimization of an objective function defined by information maximization. One advantage of a probabilistic latent variable formulation is that it helps to motivate and formulate generalizations of basic ICA. For instance, *independent factor analysis* (Attias, 1999a) considers a model in which the number of latent and observed variables can differ, the observed variables are noisy, and the individual latent variables have flexible distributions modelled by mixtures of Gaussians. The log likelihood for this model is maximized using EM, and the reconstruction of the latent variables is approximated using a variational approach. Many other types of model have been considered, and there is now a huge literature on ICA and its applications (Jutten and Herault, 1991; Comon *et al.*, 1991; Amari *et al.*, 1996; Pearlmutter and Parra, 1997; Hyvärinen and Oja, 1997; Hinton *et al.*, 2001; Miskin and MacKay, 2001; Hojen-Sorensen *et al.*, 2002; Choudrey and Roberts, 2003; Chan *et al.*, 2003; Stone, 2004).

12.4.2 Autoassociative neural networks

In Chapter 5 we considered neural networks in the context of supervised learning, where the role of the network is to predict the output variables given values

Exercise 12.29

Figure 12.18 An autoassociative multilayer perceptron having two layers of weights. Such a network is trained to map input vectors onto themselves by minimization of a sum-of-squares error. Even with nonlinear units in the hidden layer, such a network is equivalent to linear principal component analysis. Links representing bias parameters have been omitted for clarity.



for the input variables. However, neural networks have also been applied to unsupervised learning where they have been used for dimensionality reduction. This is achieved by using a network having the same number of outputs as inputs, and optimizing the weights so as to minimize some measure of the reconstruction error between inputs and outputs with respect to a set of training data.

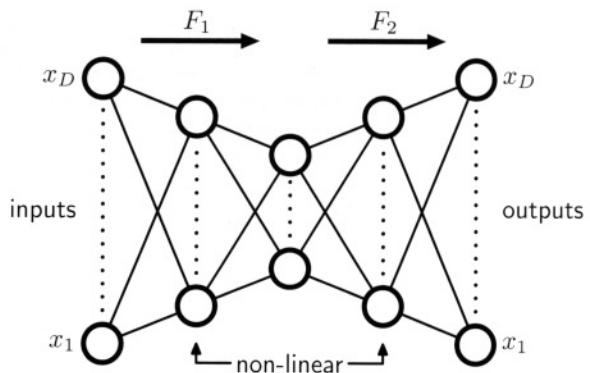
Consider first a multilayer perceptron of the form shown in Figure 12.18, having D inputs, D output units and M hidden units, with $M < D$. The targets used to train the network are simply the input vectors themselves, so that the network is attempting to map each input vector onto itself. Such a network is said to form an *autoassociative* mapping. Since the number of hidden units is smaller than the number of inputs, a perfect reconstruction of all input vectors is not in general possible. We therefore determine the network parameters \mathbf{w} by minimizing an error function which captures the degree of mismatch between the input vectors and their reconstructions. In particular, we shall choose a sum-of-squares error of the form

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{x}_n\|^2. \quad (12.91)$$

If the hidden units have linear activation functions, then it can be shown that the error function has a unique global minimum, and that at this minimum the network performs a projection onto the M -dimensional subspace which is spanned by the first M principal components of the data (Bourlard and Kamp, 1988; Baldi and Hornik, 1989). Thus, the vectors of weights which lead into the hidden units in Figure 12.18 form a basis set which spans the principal subspace. Note, however, that these vectors need not be orthogonal or normalized. This result is unsurprising, since both principal component analysis and the neural network are using linear dimensionality reduction and are minimizing the same sum-of-squares error function.

It might be thought that the limitations of a linear dimensionality reduction could be overcome by using nonlinear (sigmoidal) activation functions for the hidden units in the network in Figure 12.18. However, even with nonlinear hidden units, the minimum error solution is again given by the projection onto the principal component subspace (Bourlard and Kamp, 1988). There is therefore no advantage in using two-layer neural networks to perform dimensionality reduction. Standard techniques for principal component analysis (based on singular value decomposition) are guaranteed to give the correct solution in finite time, and they also generate an ordered set of eigenvalues with corresponding orthonormal eigenvectors.

Figure 12.19 Addition of extra hidden layers of nonlinear units gives an autoassociative network which can perform a nonlinear dimensionality reduction.



The situation is different, however, if additional hidden layers are permitted in the network. Consider the four-layer autoassociative network shown in Figure 12.19. Again the output units are linear, and the M units in the second hidden layer can also be linear, however, the first and third hidden layers have sigmoidal nonlinear activation functions. The network is again trained by minimization of the error function (12.91). We can view this network as two successive functional mappings \mathbf{F}_1 and \mathbf{F}_2 , as indicated in Figure 12.19. The first mapping \mathbf{F}_1 projects the original D -dimensional data onto an M -dimensional subspace \mathcal{S} defined by the activations of the units in the second hidden layer. Because of the presence of the first hidden layer of nonlinear units, this mapping is very general, and in particular is not restricted to being linear. Similarly, the second half of the network defines an arbitrary functional mapping from the M -dimensional space back into the original D -dimensional input space. This has a simple geometrical interpretation, as indicated for the case $D = 3$ and $M = 2$ in Figure 12.20.

Such a network effectively performs a nonlinear principal component analysis.

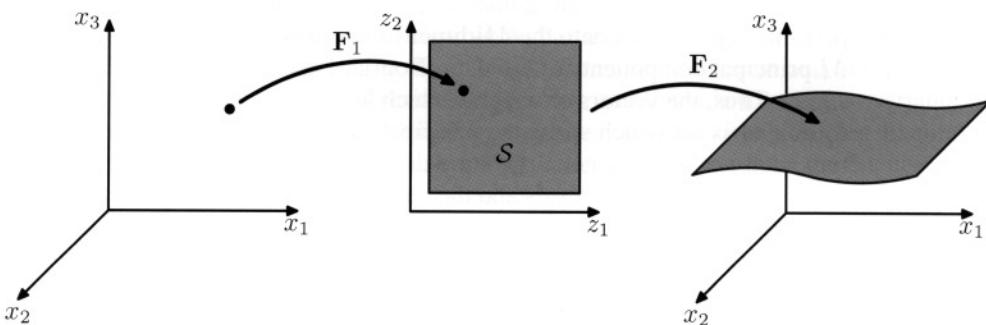


Figure 12.20 Geometrical interpretation of the mappings performed by the network in Figure 12.19 for the case of $D = 3$ inputs and $M = 2$ units in the middle hidden layer. The function \mathbf{F}_1 maps from an M -dimensional space \mathcal{S} into a D -dimensional space and therefore defines the way in which the space \mathcal{S} is embedded within the original x -space. Since the mapping \mathbf{F}_2 can be nonlinear, the embedding of \mathcal{S} can be nonplanar, as indicated in the figure. The mapping \mathbf{F}_1 then defines a projection of points in the original D -dimensional space into the M -dimensional subspace \mathcal{S} .

It has the advantage of not being limited to linear transformations, although it contains standard principal component analysis as a special case. However, training the network now involves a nonlinear optimization problem, since the error function (12.91) is no longer a quadratic function of the network parameters. Computationally intensive nonlinear optimization techniques must be used, and there is the risk of finding a suboptimal local minimum of the error function. Also, the dimensionality of the subspace must be specified before training the network.

12.4.3 Modelling nonlinear manifolds

As we have already noted, many natural sources of data correspond to low-dimensional, possibly noisy, nonlinear manifolds embedded within the higher dimensional observed data space. Capturing this property explicitly can lead to improved density modelling compared with more general methods. Here we consider briefly a range of techniques that attempt to do this.

One way to model the nonlinear structure is through a combination of linear models, so that we make a piece-wise linear approximation to the manifold. This can be obtained, for instance, by using a clustering technique such as K -means based on Euclidean distance to partition the data set into local groups with standard PCA applied to each group. A better approach is to use the reconstruction error for cluster assignment (Kambhatla and Leen, 1997; Hinton *et al.*, 1997) as then a common cost function is being optimized in each stage. However, these approaches still suffer from limitations due to the absence of an overall density model. By using probabilistic PCA it is straightforward to define a fully probabilistic model simply by considering a mixture distribution in which the components are probabilistic PCA models (Tipping and Bishop, 1999a). Such a model has both discrete latent variables, corresponding to the discrete mixture, as well as continuous latent variables, and the likelihood function can be maximized using the EM algorithm. A fully Bayesian treatment, based on variational inference (Bishop and Winn, 2000), allows the number of components in the mixture, as well as the effective dimensionalities of the individual models, to be inferred from the data. There are many variants of this model in which parameters such as the \mathbf{W} matrix or the noise variances are tied across components in the mixture, or in which the isotropic noise distributions are replaced by diagonal ones, giving rise to a mixture of factor analysers (Ghahramani and Hinton, 1996a; Ghahramani and Beal, 2000). The mixture of probabilistic PCA models can also be extended hierarchically to produce an interactive data visualization algorithm (Bishop and Tipping, 1998).

An alternative to considering a mixture of linear models is to consider a single nonlinear model. Recall that conventional PCA finds a linear subspace that passes close to the data in a least-squares sense. This concept can be extended to one-dimensional nonlinear surfaces in the form of *principal curves* (Hastie and Stuetzle, 1989). We can describe a curve in a D -dimensional data space using a vector-valued function $\mathbf{f}(\lambda)$, which is a vector each of whose elements is a function of the scalar λ . There are many possible ways to parameterize the curve, of which a natural choice is the arc length along the curve. For any given point $\hat{\mathbf{x}}$ in data space, we can find the point on the curve that is closest in Euclidean distance. We denote this point by

$\lambda = g_{\mathbf{f}}(\mathbf{x})$ because it depends on the particular curve $\mathbf{f}(\lambda)$. For a continuous data density $p(\mathbf{x})$, a principal curve is defined as one for which every point on the curve is the mean of all those points in data space that project to it, so that

$$\mathbb{E} [\mathbf{x}|g_{\mathbf{f}}(\mathbf{x}) = \lambda] = \mathbf{f}(\lambda). \quad (12.92)$$

For a given continuous density, there can be many principal curves. In practice, we are interested in finite data sets, and we also wish to restrict attention to smooth curves. Hastie and Stuetzle (1989) propose a two-stage iterative procedure for finding such principal curves, somewhat reminiscent of the EM algorithm for PCA. The curve is initialized using the first principal component, and then the algorithm alternates between a data projection step and curve re-estimation step. In the projection step, each data point is assigned to a value of λ corresponding to the closest point on the curve. Then in the re-estimation step, each point on the curve is given by a weighted average of those points that project to nearby points on the curve, with points closest on the curve given the greatest weight. In the case where the subspace is constrained to be linear, the procedure converges to the first principal component and is equivalent to the power method for finding the largest eigenvector of the covariance matrix. Principal curves can be generalized to multidimensional manifolds called *principal surfaces* although these have found limited use due to the difficulty of data smoothing in higher dimensions even for two-dimensional manifolds.

PCA is often used to project a data set onto a lower-dimensional space, for example two dimensional, for the purposes of visualization. Another linear technique with a similar aim is *multidimensional scaling*, or *MDS* (Cox and Cox, 2000). It finds a low-dimensional projection of the data such as to preserve, as closely as possible, the pairwise distances between data points, and involves finding the eigenvectors of the distance matrix. In the case where the distances are Euclidean, it gives equivalent results to PCA. The MDS concept can be extended to a wide variety of data types specified in terms of a similarity matrix, giving *nonmetric MDS*.

Two other nonprobabilistic methods for dimensionality reduction and data visualization are worthy of mention. *Locally linear embedding*, or *LLE* (Roweis and Saul, 2000) first computes the set of coefficients that best reconstructs each data point from its neighbours. These coefficients are arranged to be invariant to rotations, translations, and scalings of that data point and its neighbours, and hence they characterize the local geometrical properties of the neighbourhood. LLE then maps the high-dimensional data points down to a lower dimensional space while preserving these neighbourhood coefficients. If the local neighbourhood for a particular data point can be considered linear, then the transformation can be achieved using a combination of translation, rotation, and scaling, such as to preserve the angles formed between the data points and their neighbours. Because the weights are invariant to these transformations, we expect the same weight values to reconstruct the data points in the low-dimensional space as in the high-dimensional data space. In spite of the nonlinearity, the optimization for LLE does not exhibit local minima.

In *isometric feature mapping*, or *isomap* (Tenenbaum *et al.*, 2000), the goal is to project the data to a lower-dimensional space using MDS, but where the dissimilarities are defined in terms of the *geodesic distances* measured along the mani-

fold. For instance, if two points lie on a circle, then the geodesic is the arc-length distance measured around the circumference of the circle not the straight line distance measured along the chord connecting them. The algorithm first defines the neighbourhood for each data point, either by finding the K nearest neighbours or by finding all points within a sphere of radius ϵ . A graph is then constructed by linking all neighbouring points and labelling them with their Euclidean distance. The geodesic distance between any pair of points is then approximated by the sum of the arc lengths along the shortest path connecting them (which itself is found using standard algorithms). Finally, metric MDS is applied to the geodesic distance matrix to find the low-dimensional projection.

Our focus in this chapter has been on models for which the observed variables are continuous. We can also consider models having continuous latent variables together with discrete observed variables, giving rise to *latent trait* models (Bartholomew, 1987). In this case, the marginalization over the continuous latent variables, even for a linear relationship between latent and observed variables, cannot be performed analytically, and so more sophisticated techniques are required. Tipping (1999) uses variational inference in a model with a two-dimensional latent space, allowing a binary data set to be visualized analogously to the use of PCA to visualize continuous data. Note that this model is the dual of the Bayesian logistic regression problem discussed in Section 4.5. In the case of logistic regression we have N observations of the feature vector ϕ_n which are parameterized by a single parameter vector w , whereas in the latent space visualization model there is a single latent space variable x (analogous to ϕ) and N copies of the latent variable w_n . A generalization of probabilistic latent variable models to general exponential family distributions is described in Collins *et al.* (2002).

We have already noted that an arbitrary distribution can be formed by taking a Gaussian random variable and transforming it through a suitable nonlinearity. This is exploited in a general latent variable model called a *density network* (MacKay, 1995; MacKay and Gibbs, 1999) in which the nonlinear function is governed by a multilayered neural network. If the network has enough hidden units, it can approximate a given nonlinear function to any desired accuracy. The downside of having such a flexible model is that the marginalization over the latent variables, required in order to obtain the likelihood function, is no longer analytically tractable. Instead, the likelihood is approximated using Monte Carlo techniques by drawing samples from the Gaussian prior. The marginalization over the latent variables then becomes a simple sum with one term for each sample. However, because a large number of sample points may be required in order to give an accurate representation of the marginal, this procedure can be computationally costly.

If we consider more restricted forms for the nonlinear function, and make an appropriate choice of the latent variable distribution, then we can construct a latent variable model that is both nonlinear and efficient to train. The *generative topographic mapping*, or *GTM* (Bishop *et al.*, 1996; Bishop *et al.*, 1997a; Bishop *et al.*, 1998b) uses a latent distribution that is defined by a finite regular grid of delta functions over the (typically two-dimensional) latent space. Marginalization over the latent space then simply involves summing over the contributions from each of the grid locations.

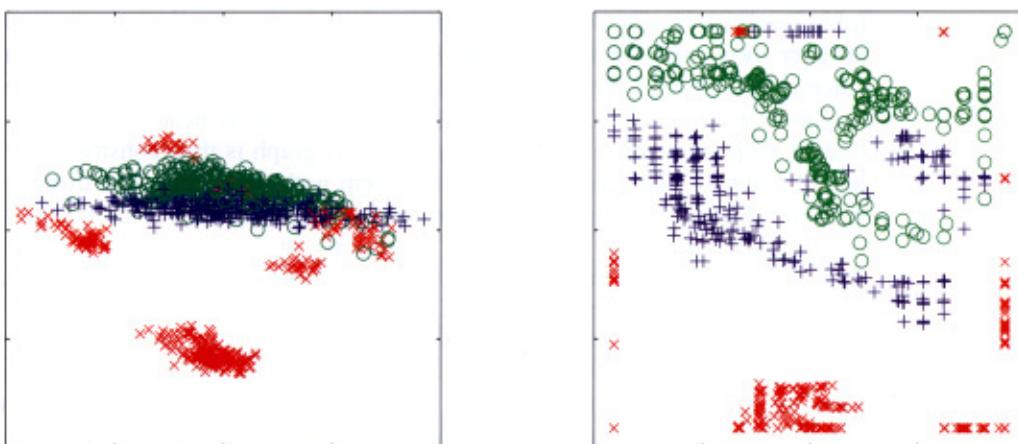


Figure 12.21 Plot of the oil flow data set visualized using PCA on the left and GTM on the right. For the GTM model, each data point is plotted at the mean of its posterior distribution in latent space. The nonlinearity of the GTM model allows the separation between the groups of data points to be seen more clearly.

Chapter 3

Section 1.4

The nonlinear mapping is given by a linear regression model that allows for general nonlinearity while being a linear function of the adaptive parameters. Note that the usual limitation of linear regression models arising from the curse of dimensionality does not arise in the context of the GTM since the manifold generally has two dimensions irrespective of the dimensionality of the data space. A consequence of these two choices is that the likelihood function can be expressed analytically in closed form and can be optimized efficiently using the EM algorithm. The resulting GTM model fits a two-dimensional nonlinear manifold to the data set, and by evaluating the posterior distribution over latent space for the data points, they can be projected back to the latent space for visualization purposes. Figure 12.21 shows a comparison of the oil data set visualized with linear PCA and with the nonlinear GTM.

The GTM can be seen as a probabilistic version of an earlier model called the *self organizing map*, or *SOM* (Kohonen, 1982; Kohonen, 1995), which also represents a two-dimensional nonlinear manifold as a regular array of discrete points. The SOM is somewhat reminiscent of the *K*-means algorithm in that data points are assigned to nearby prototype vectors that are then subsequently updated. Initially, the prototypes are distributed at random, and during the training process they ‘self organize’ so as to approximate a smooth manifold. Unlike *K*-means, however, the SOM is not optimizing any well-defined cost function (Erwin *et al.*, 1992) making it difficult to set the parameters of the model and to assess convergence. There is also no guarantee that the ‘self-organization’ will take place as this is dependent on the choice of appropriate parameter values for any particular data set.

By contrast, GTM optimizes the log likelihood function, and the resulting model defines a probability density in data space. In fact, it corresponds to a constrained mixture of Gaussians in which the components share a common variance, and the means are constrained to lie on a smooth two-dimensional manifold. This proba-

Section 6.4

bilistic foundation also makes it very straightforward to define generalizations of GTM (Bishop *et al.*, 1998a) such as a Bayesian treatment, dealing with missing values, a principled extension to discrete variables, the use of Gaussian processes to define the manifold, or a hierarchical GTM model (Tino and Nabney, 2002).

Because the manifold in GTM is defined as a continuous surface, not just at the prototype vectors as in the SOM, it is possible to compute the *magnification factors* corresponding to the local expansions and compressions of the manifold needed to fit the data set (Bishop *et al.*, 1997b) as well as the *directional curvatures* of the manifold (Tino *et al.*, 2001). These can be visualized along with the projected data and provide additional insight into the model.

Exercises

Appendix E

- 12.1** (**) **www** In this exercise, we use proof by induction to show that the linear projection onto an M -dimensional subspace that maximizes the variance of the projected data is defined by the M eigenvectors of the data covariance matrix \mathbf{S} , given by (12.3), corresponding to the M largest eigenvalues. In Section 12.1, this result was proven for the case of $M = 1$. Now suppose the result holds for some general value of M and show that it consequently holds for dimensionality $M + 1$. To do this, first set the derivative of the variance of the projected data with respect to a vector \mathbf{u}_{M+1} defining the new direction in data space equal to zero. This should be done subject to the constraints that \mathbf{u}_{M+1} be orthogonal to the existing vectors $\mathbf{u}_1, \dots, \mathbf{u}_M$, and also that it be normalized to unit length. Use Lagrange multipliers to enforce these constraints. Then make use of the orthonormality properties of the vectors $\mathbf{u}_1, \dots, \mathbf{u}_M$ to show that the new vector \mathbf{u}_{M+1} is an eigenvector of \mathbf{S} . Finally, show that the variance is maximized if the eigenvector is chosen to be the one corresponding to eigenvector λ_{M+1} where the eigenvalues have been ordered in decreasing value.
- 12.2** (**) Show that the minimum value of the PCA distortion measure J given by (12.15) with respect to the \mathbf{u}_i , subject to the orthonormality constraints (12.7), is obtained when the \mathbf{u}_i are eigenvectors of the data covariance matrix \mathbf{S} . To do this, introduce a matrix \mathbf{H} of Lagrange multipliers, one for each constraint, so that the modified distortion measure, in matrix notation reads

$$\tilde{J} = \text{Tr} \left\{ \widehat{\mathbf{U}}^T \mathbf{S} \widehat{\mathbf{U}} \right\} + \text{Tr} \left\{ \mathbf{H} (\mathbf{I} - \widehat{\mathbf{U}}^T \widehat{\mathbf{U}}) \right\} \quad (12.93)$$

where $\widehat{\mathbf{U}}$ is a matrix of dimension $D \times (D - M)$ whose columns are given by \mathbf{u}_i . Now minimize \tilde{J} with respect to $\widehat{\mathbf{U}}$ and show that the solution satisfies $\mathbf{S} \widehat{\mathbf{U}} = \widehat{\mathbf{U}} \mathbf{H}$. Clearly, one possible solution is that the columns of $\widehat{\mathbf{U}}$ are eigenvectors of \mathbf{S} , in which case \mathbf{H} is a diagonal matrix containing the corresponding eigenvalues. To obtain the general solution, show that \mathbf{H} can be assumed to be a symmetric matrix, and by using its eigenvector expansion show that the general solution to $\mathbf{S} \widehat{\mathbf{U}} = \widehat{\mathbf{U}} \mathbf{H}$ gives the same value for \tilde{J} as the specific solution in which the columns of $\widehat{\mathbf{U}}$ are

the eigenvectors of \mathbf{S} . Because these solutions are all equivalent, it is convenient to choose the eigenvector solution.

- 12.3** (*) Verify that the eigenvectors defined by (12.30) are normalized to unit length, assuming that the eigenvectors \mathbf{v}_i have unit length.
- 12.4** (*) **www** Suppose we replace the zero-mean, unit-covariance latent space distribution (12.31) in the probabilistic PCA model by a general Gaussian distribution of the form $\mathcal{N}(\mathbf{z}|\mathbf{m}, \Sigma)$. By redefining the parameters of the model, show that this leads to an identical model for the marginal distribution $p(\mathbf{x})$ over the observed variables for any valid choice of \mathbf{m} and Σ .
- 12.5** (**) Let \mathbf{x} be a D -dimensional random variable having a Gaussian distribution given by $\mathcal{N}(\mathbf{x}|\mu, \Sigma)$, and consider the M -dimensional random variable given by $\mathbf{y} = \mathbf{Ax} + \mathbf{b}$ where \mathbf{A} is an $M \times D$ matrix. Show that \mathbf{y} also has a Gaussian distribution, and find expressions for its mean and covariance. Discuss the form of this Gaussian distribution for $M < D$, for $M = D$, and for $M > D$.
- 12.6** (*) **www** Draw a directed probabilistic graph for the probabilistic PCA model described in Section 12.2 in which the components of the observed variable \mathbf{x} are shown explicitly as separate nodes. Hence verify that the probabilistic PCA model has the same independence structure as the naive Bayes model discussed in Section 8.2.2.
- 12.7** (**) By making use of the results (2.270) and (2.271) for the mean and covariance of a general distribution, derive the result (12.35) for the marginal distribution $p(\mathbf{x})$ in the probabilistic PCA model.
- 12.8** (**) **www** By making use of the result (2.116), show that the posterior distribution $p(\mathbf{z}|\mathbf{x})$ for the probabilistic PCA model is given by (12.42).
- 12.9** (*) Verify that maximizing the log likelihood (12.43) for the probabilistic PCA model with respect to the parameter μ gives the result $\mu_{\text{ML}} = \bar{\mathbf{x}}$ where $\bar{\mathbf{x}}$ is the mean of the data vectors.
- 12.10** (**) By evaluating the second derivatives of the log likelihood function (12.43) for the probabilistic PCA model with respect to the parameter μ , show that the stationary point $\mu_{\text{ML}} = \bar{\mathbf{x}}$ represents the unique maximum.
- 12.11** (**) **www** Show that in the limit $\sigma^2 \rightarrow 0$, the posterior mean for the probabilistic PCA model becomes an orthogonal projection onto the principal subspace, as in conventional PCA.
- 12.12** (**) For $\sigma^2 > 0$ show that the posterior mean in the probabilistic PCA model is shifted towards the origin relative to the orthogonal projection.
- 12.13** (**) Show that the optimal reconstruction of a data point under probabilistic PCA, according to the least squares projection cost of conventional PCA, is given by

$$\tilde{\mathbf{x}} = \mathbf{W}_{\text{ML}}(\mathbf{W}_{\text{ML}}^T \mathbf{W}_{\text{ML}})^{-1} \mathbf{M} \mathbb{E}[\mathbf{z}|\mathbf{x}]. \quad (12.94)$$

- 12.14** (*) The number of independent parameters in the covariance matrix for the probabilistic PCA model with an M -dimensional latent space and a D -dimensional data space is given by (12.51). Verify that in the case of $M = D - 1$, the number of independent parameters is the same as in a general covariance Gaussian, whereas for $M = 0$ it is the same as for a Gaussian with an isotropic covariance.
- 12.15** (**) **www** Derive the M-step equations (12.56) and (12.57) for the probabilistic PCA model by maximization of the expected complete-data log likelihood function given by (12.53).
- 12.16** (***) In Figure 12.11, we showed an application of probabilistic PCA to a data set in which some of the data values were missing at random. Derive the EM algorithm for maximizing the likelihood function for the probabilistic PCA model in this situation. Note that the $\{\mathbf{z}_n\}$, as well as the missing data values that are components of the vectors $\{\mathbf{x}_n\}$, are now latent variables. Show that in the special case in which all of the data values are observed, this reduces to the EM algorithm for probabilistic PCA derived in Section 12.2.2.
- 12.17** (**) **www** Let \mathbf{W} be a $D \times M$ matrix whose columns define a linear subspace of dimensionality M embedded within a data space of dimensionality D , and let $\boldsymbol{\mu}$ be a D -dimensional vector. Given a data set $\{\mathbf{x}_n\}$ where $n = 1, \dots, N$, we can approximate the data points using a linear mapping from a set of M -dimensional vectors $\{\mathbf{z}_n\}$, so that \mathbf{x}_n is approximated by $\mathbf{W}\mathbf{z}_n + \boldsymbol{\mu}$. The associated sum-of-squares reconstruction cost is given by

$$J = \sum_{n=1}^N \|\mathbf{x}_n - \boldsymbol{\mu} - \mathbf{W}\mathbf{z}_n\|^2. \quad (12.95)$$

First show that minimizing J with respect to $\boldsymbol{\mu}$ leads to an analogous expression with \mathbf{x}_n and \mathbf{z}_n replaced by zero-mean variables $\mathbf{x}_n - \bar{\mathbf{x}}$ and $\mathbf{z}_n - \bar{\mathbf{z}}$, respectively, where $\bar{\mathbf{x}}$ and $\bar{\mathbf{z}}$ denote sample means. Then show that minimizing J with respect to \mathbf{z}_n , where \mathbf{W} is kept fixed, gives rise to the PCA E step (12.58), and that minimizing J with respect to \mathbf{W} , where $\{\mathbf{z}_n\}$ is kept fixed, gives rise to the PCA M step (12.59).

- 12.18** (*) Derive an expression for the number of independent parameters in the factor analysis model described in Section 12.2.4.
- 12.19** (**) **www** Show that the factor analysis model described in Section 12.2.4 is invariant under rotations of the latent space coordinates.
- 12.20** (**) By considering second derivatives, show that the only stationary point of the log likelihood function for the factor analysis model discussed in Section 12.2.4 with respect to the parameter $\boldsymbol{\mu}$ is given by the sample mean defined by (12.1). Furthermore, show that this stationary point is a maximum.
- 12.21** (**) Derive the formulae (12.66) and (12.67) for the E step of the EM algorithm for factor analysis. Note that from the result of Exercise 12.20, the parameter $\boldsymbol{\mu}$ can be replaced by the sample mean $\bar{\mathbf{x}}$.

- 12.22** (**) Write down an expression for the expected complete-data log likelihood function for the factor analysis model, and hence derive the corresponding M step equations (12.69) and (12.70).
- 12.23** (*) **www** Draw a directed probabilistic graphical model representing a discrete mixture of probabilistic PCA models in which each PCA model has its own values of \mathbf{W} , $\boldsymbol{\mu}$, and σ^2 . Now draw a modified graph in which these parameter values are shared between the components of the mixture.
- 12.24** (***) We saw in Section 2.3.7 that Student's t-distribution can be viewed as an infinite mixture of Gaussians in which we marginalize with respect to a continuous latent variable. By exploiting this representation, formulate an EM algorithm for maximizing the log likelihood function for a multivariate Student's t-distribution given an observed set of data points, and derive the forms of the E and M step equations.
- 12.25** (**) **www** Consider a linear-Gaussian latent-variable model having a latent space distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ and a conditional distribution for the observed variable $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Phi})$ where $\boldsymbol{\Phi}$ is an arbitrary symmetric, positive-definite noise covariance matrix. Now suppose that we make a nonsingular linear transformation of the data variables $\mathbf{x} \rightarrow \mathbf{Ax}$, where \mathbf{A} is a $D \times D$ matrix. If $\boldsymbol{\mu}_{\text{ML}}$, \mathbf{W}_{ML} and $\boldsymbol{\Phi}_{\text{ML}}$ represent the maximum likelihood solution corresponding to the original untransformed data, show that $\mathbf{A}\boldsymbol{\mu}_{\text{ML}}$, \mathbf{AW}_{ML} , and $\mathbf{A}\boldsymbol{\Phi}_{\text{ML}}\mathbf{A}^T$ will represent the corresponding maximum likelihood solution for the transformed data set. Finally, show that the form of the model is preserved in two cases: (i) \mathbf{A} is a diagonal matrix and $\boldsymbol{\Phi}$ is a diagonal matrix. This corresponds to the case of factor analysis. The transformed $\boldsymbol{\Phi}$ remains diagonal, and hence factor analysis is *covariant* under component-wise re-scaling of the data variables; (ii) \mathbf{A} is orthogonal and $\boldsymbol{\Phi}$ is proportional to the unit matrix so that $\boldsymbol{\Phi} = \sigma^2 \mathbf{I}$. This corresponds to probabilistic PCA. The transformed $\boldsymbol{\Phi}$ matrix remains proportional to the unit matrix, and hence probabilistic PCA is covariant under a rotation of the axes of data space, as is the case for conventional PCA.
- 12.26** (**) Show that any vector \mathbf{a}_i that satisfies (12.80) will also satisfy (12.79). Also, show that for any solution of (12.80) having eigenvalue λ , we can add any multiple of an eigenvector of \mathbf{K} having zero eigenvalue, and obtain a solution to (12.79) that also has eigenvalue λ . Finally, show that such modifications do not affect the principal-component projection given by (12.82).
- 12.27** (**) Show that the conventional linear PCA algorithm is recovered as a special case of kernel PCA if we choose the linear kernel function given by $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$.
- 12.28** (**) **www** Use the transformation property (1.27) of a probability density under a change of variable to show that any density $p(y)$ can be obtained from a fixed density $q(x)$ that is everywhere nonzero by making a nonlinear change of variable $y = f(x)$ in which $f(x)$ is a monotonic function so that $0 \leq f'(x) < \infty$. Write down the differential equation satisfied by $f(x)$ and draw a diagram illustrating the transformation of the density.

- 12.29** (**) **www** Suppose that two variables z_1 and z_2 are independent so that $p(z_1, z_2) = p(z_1)p(z_2)$. Show that the covariance matrix between these variables is diagonal. This shows that independence is a sufficient condition for two variables to be uncorrelated. Now consider two variables y_1 and y_2 in which $-1 \leq y_1 \leq 1$ and $y_2 = y_1^2$. Write down the conditional distribution $p(y_2|y_1)$ and observe that this is dependent on y_1 , showing that the two variables are not independent. Now show that the covariance matrix between these two variables is again diagonal. To do this, use the relation $p(y_1, y_2) = p(y_1)p(y_2|y_1)$ to show that the off-diagonal terms are zero. This counter-example shows that zero correlation is not a sufficient condition for independence.