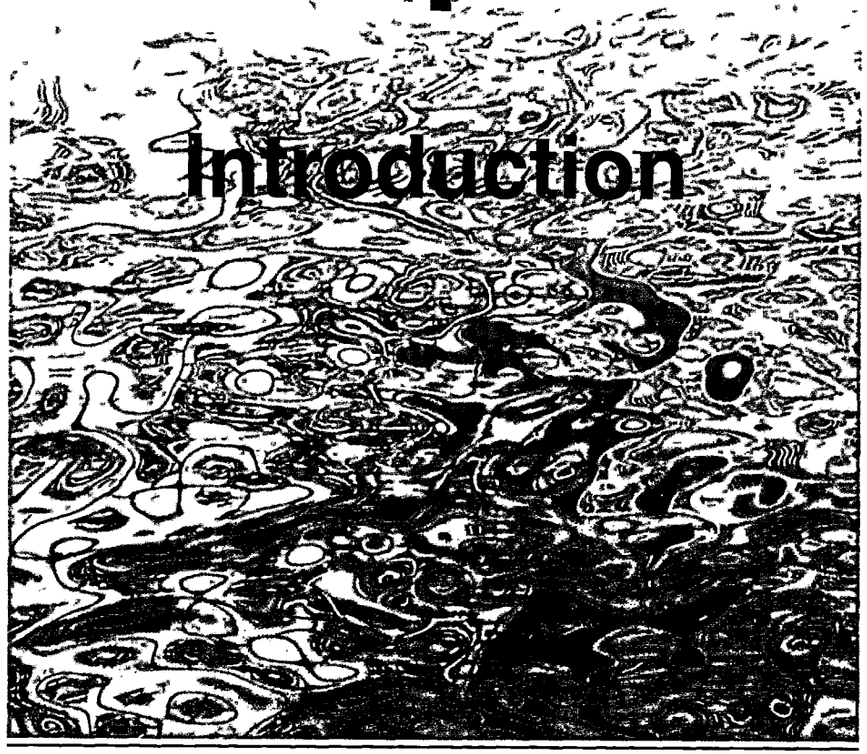
$\mathbb{E}[x]$. If the distribution of x is conditioned on another variable z, then the corresponding conditional expectation will be written $\mathbb{E}_x[f(x)|z]$. Similarly, the variance is denoted $\mathrm{var}[f(x)]$, and for vector variables the covariance is written $\mathrm{cov}[\mathbf{x}, \mathbf{y}]$. We shall also use $\mathrm{cov}[\mathbf{x}]$ as a shorthand notation for $\mathrm{cov}[\mathbf{x}, \mathbf{x}]$. The concepts of expectations and covariances are introduced in Section 1.2.2.

If we have N values $\mathbf{x}_1, \ldots, \mathbf{x}_N$ of a D-dimensional vector $\mathbf{x} = (x_1, \ldots, x_D)^{\mathrm{T}}$, we can combine the observations into a data matrix $\mathbf{X}$ in which the n^{th} row of $\mathbf{X}$ corresponds to the row vector $\mathbf{x}_n^{\mathrm{T}}$. Thus the n, i element of $\mathbf{X}$ corresponds to the i^{th} element of the n^{th} observation $\mathbf{x}_n$. For the case of one-dimensional variables we shall denote such a matrix by $\mathbf{x}$, which is a column vector whose n^{th} element is x_n. Note that $\mathbf{x}$ (which has dimensionality N) uses a different typeface to distinguish it from $\mathbf{x}$ (which has dimensionality D).

Contents

1 Introduction



The problem of searching for patterns in data is a fundamental one and has a long and successful history. For instance, the extensive astronomical observations of Tycho Brahe in the 16^{th} century allowed Johannes Kepler to discover the empirical laws of planetary motion, which in turn provided a springboard for the development of classical mechanics. Similarly, the discovery of regularities in atomic spectra played a key role in the development and verification of quantum physics in the early twentieth century. The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories.

Consider the example of recognizing handwritten digits, illustrated in Figure 1.1. Each digit corresponds to a 28×28 pixel image and so can be represented by a vector $\mathbf{x}$ comprising 784 real numbers. The goal is to build a machine that will take such a vector $\mathbf{x}$ as input and that will produce the identity of the digit $0, \ldots, 9$ as the output. This is a nontrivial problem due to the wide variability of handwriting. It could be

in a high-dimensional space whose dimensionality is determined by the number of pixels. Because the objects can occur at different positions within the image and in different orientations, there are three degrees of freedom of variability between images, and a set of images will live on a three dimensional *manifold* embedded within the high-dimensional space. Due to the complex relationships between the object position or orientation and the pixel intensities, this manifold will be highly nonlinear. If the goal is to learn a model that can take an input image and output the orientation of the object irrespective of its position, then there is only one degree of freedom of variability within the manifold that is significant.

1.5. Decision Theory

We have seen in Section 1.2 how probability theory provides us with a consistent mathematical framework for quantifying and manipulating uncertainty. Here we turn to a discussion of decision theory that, when combined with probability theory, allows us to make optimal decisions in situations involving uncertainty such as those encountered in pattern recognition.

Suppose we have an input vector $\mathbf{x}$ together with a corresponding vector $\mathbf{t}$ of target variables, and our goal is to predict $\mathbf{t}$ given a new value for $\mathbf{x}$. For regression problems, $\mathbf{t}$ will comprise continuous variables, whereas for classification problems $\mathbf{t}$ will represent class labels. The joint probability distribution $p(\mathbf{x}, \mathbf{t})$ provides a complete summary of the uncertainty associated with these variables. Determination of $p(\mathbf{x}, \mathbf{t})$ from a set of training data is an example of *inference* and is typically a very difficult problem whose solution forms the subject of much of this book. In a practical application, however, we must often make a specific prediction for the value of $\mathbf{t}$, or more generally take a specific action based on our understanding of the values $\mathbf{t}$ is likely to take, and this aspect is the subject of decision theory.

Consider, for example, a medical diagnosis problem in which we have taken an X-ray image of a patient, and we wish to determine whether the patient has cancer or not. In this case, the input vector $\mathbf{x}$ is the set of pixel intensities in the image, and output variable t will represent the presence of cancer, which we denote by the class $\mathcal{C}_1$, or the absence of cancer, which we denote by the class $\mathcal{C}_2$. We might, for instance, choose t to be a binary variable such that $t = 0$ corresponds to class $\mathcal{C}_1$ and $t = 1$ corresponds to class $\mathcal{C}_2$. We shall see later that this choice of label values is particularly convenient for probabilistic models. The general inference problem then involves determining the joint distribution $p(\mathbf{x}, \mathcal{C}_k)$, or equivalently $p(\mathbf{x}, t)$, which gives us the most complete probabilistic description of the situation. Although this can be a very useful and informative quantity, in the end we must decide either to give treatment to the patient or not, and we would like this choice to be optimal in some appropriate sense (Duda and Hart, 1973). This is the *decision* step, and it is the subject of decision theory to tell us how to make optimal decisions given the appropriate probabilities. We shall see that the decision stage is generally very simple, even trivial, once we have solved the inference problem.

Here we give an introduction to the key ideas of decision theory as required for

the rest of the book. Further background, as well as more detailed accounts, can be found in Berger (1985) and Bather (2000).

Before giving a more detailed analysis, let us first consider informally how we might expect probabilities to play a role in making decisions. When we obtain the X-ray image $\mathbf{x}$ for a new patient, our goal is to decide which of the two classes to assign to the image. We are interested in the probabilities of the two classes given the image, which are given by $p(\mathcal{C}_k|\mathbf{x})$. Using Bayes' theorem, these probabilities can be expressed in the form

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}. \tag{1.77}$$

Note that any of the quantities appearing in Bayes' theorem can be obtained from the joint distribution $p(\mathbf{x}, \mathcal{C}_k)$ by either marginalizing or conditioning with respect to the appropriate variables. We can now interpret $p(\mathcal{C}_k)$ as the prior probability for the class $\mathcal{C}_k$, and $p(\mathcal{C}_k|\mathbf{x})$ as the corresponding posterior probability. Thus $p(\mathcal{C}_1)$ represents the probability that a person has cancer, before we take the X-ray measurement. Similarly, $p(\mathcal{C}_1|\mathbf{x})$ is the corresponding probability, revised using Bayes' theorem in light of the information contained in the X-ray. If our aim is to minimize the chance of assigning $\mathbf{x}$ to the wrong class, then intuitively we would choose the class having the higher posterior probability. We now show that this intuition is correct, and we also discuss more general criteria for making decisions.

1.5.1 Minimizing the misclassification rate

Suppose that our goal is simply to make as few misclassifications as possible. We need a rule that assigns each value of $\mathbf{x}$ to one of the available classes. Such a rule will divide the input space into regions $\mathcal{R}_k$ called *decision regions*, one for each class, such that all points in $\mathcal{R}_k$ are assigned to class $\mathcal{C}_k$. The boundaries between decision regions are called *decision boundaries* or *decision surfaces*. Note that each decision region need not be contiguous but could comprise some number of disjoint regions. We shall encounter examples of decision boundaries and decision regions in later chapters. In order to find the optimal decision rule, consider first of all the case of two classes, as in the cancer problem for instance. A mistake occurs when an input vector belonging to class $\mathcal{C}_1$ is assigned to class $\mathcal{C}_2$ or vice versa. The probability of this occurring is given by

$$\begin{aligned} p(\text{mistake}) &= p(\mathbf{x} \in \mathcal{R}_1, \mathcal{C}_2) + p(\mathbf{x} \in \mathcal{R}_2, \mathcal{C}_1) \\ &= \int_{\mathcal{R}_1} p(\mathbf{x}, \mathcal{C}_2)\, \mathrm{d}\mathbf{x} + \int_{\mathcal{R}_2} p(\mathbf{x}, \mathcal{C}_1)\, \mathrm{d}\mathbf{x}. \end{aligned} \tag{1.78}$$

We are free to choose the decision rule that assigns each point $\mathbf{x}$ to one of the two classes. Clearly to minimize $p(\text{mistake})$ we should arrange that each $\mathbf{x}$ is assigned to whichever class has the smaller value of the integrand in (1.78). Thus, if $p(\mathbf{x}, \mathcal{C}_1) > p(\mathbf{x}, \mathcal{C}_2)$ for a given value of $\mathbf{x}$, then we should assign that $\mathbf{x}$ to class $\mathcal{C}_1$. From the product rule of probability we have $p(\mathbf{x}, \mathcal{C}_k) = p(\mathcal{C}_k|\mathbf{x})p(\mathbf{x})$. Because the factor $p(\mathbf{x})$ is common to both terms, we can restate this result as saying that the minimum

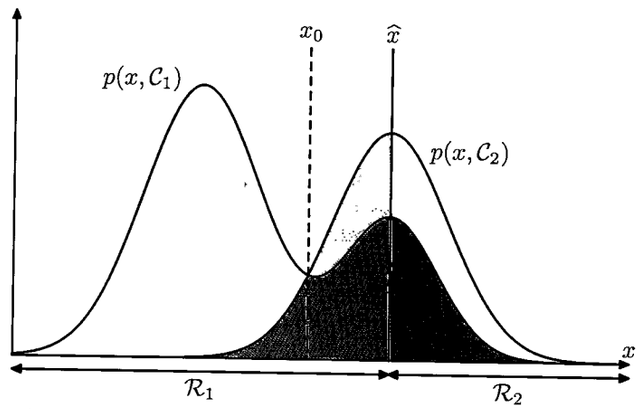


Figure 1.24 Schematic illustration of the joint probabilities $p(x, \mathcal{C}_k)$ for each of two classes plotted against x, together with the decision boundary $x = \widehat{x}$. Values of $x \geqslant \widehat{x}$ are classified as class $\mathcal{C}_2$ and hence belong to decision region $\mathcal{R}_2$, whereas points $x < \widehat{x}$ are classified as $\mathcal{C}_1$ and belong to $\mathcal{R}_1$. Errors arise from the blue, green, and red regions, so that for $x < \widehat{x}$ the errors are due to points from class $\mathcal{C}_2$ being misclassified as $\mathcal{C}_1$ (represented by the sum of the red and green regions), and conversely for points in the region $x \geqslant \widehat{x}$ the errors are due to points from class $\mathcal{C}_1$ being misclassified as $\mathcal{C}_2$ (represented by the blue region). As we vary the location $\widehat{x}$ of the decision boundary, the combined areas of the blue and green regions remains constant, whereas the size of the red region varies. The optimal choice for $\widehat{x}$ is where the curves for $p(x, \mathcal{C}_1)$ and $p(x, \mathcal{C}_2)$ cross, corresponding to $\widehat{x} = x_0$, because in this case the red region disappears. This is equivalent to the minimum misclassification rate decision rule, which assigns each value of x to the class having the higher posterior probability $p(\mathcal{C}_k|x)$.

probability of making a mistake is obtained if each value of $\mathbf{x}$ is assigned to the class for which the posterior probability $p(\mathcal{C}_k|\mathbf{x})$ is largest. This result is illustrated for two classes, and a single input variable x, in Figure 1.24.

For the more general case of K classes, it is slightly easier to maximize the probability of being correct, which is given by

$$\begin{aligned} p(\text{correct}) &= \sum_{k=1}^{K} p(\mathbf{x} \in \mathcal{R}_k, \mathcal{C}_k) \\ &= \sum_{k=1}^{K} \int_{\mathcal{R}_k} p(\mathbf{x}, \mathcal{C}_k)\, \mathrm{d}\mathbf{x} \end{aligned} \tag{1.79}$$

which is maximized when the regions $\mathcal{R}_k$ are chosen such that each $\mathbf{x}$ is assigned to the class for which $p(\mathbf{x}, \mathcal{C}_k)$ is largest. Again, using the product rule $p(\mathbf{x}, \mathcal{C}_k) = p(\mathcal{C}_k|\mathbf{x})p(\mathbf{x})$, and noting that the factor of $p(\mathbf{x})$ is common to all terms, we see that each $\mathbf{x}$ should be assigned to the class having the largest posterior probability $p(\mathcal{C}_k|\mathbf{x})$.

Figure 1.25 An example of a loss matrix with elements L_{kj} for the cancer treatment problem. The rows correspond to the true class, whereas the columns correspond to the assignment of class made by our decision criterion.

	cancer	normal
cancer	0	1000
normal	1	0

1.5.2 Minimizing the expected loss

For many applications, our objective will be more complex than simply minimizing the number of misclassifications. Let us consider again the medical diagnosis problem. We note that, if a patient who does not have cancer is incorrectly diagnosed as having cancer, the consequences may be some patient distress plus the need for further investigations. Conversely, if a patient with cancer is diagnosed as healthy, the result may be premature death due to lack of treatment. Thus the consequences of these two types of mistake can be dramatically different. It would clearly be better to make fewer mistakes of the second kind, even if this was at the expense of making more mistakes of the first kind.

We can formalize such issues through the introduction of a *loss function*, also called a *cost function*, which is a single, overall measure of loss incurred in taking any of the available decisions or actions. Our goal is then to minimize the total loss incurred. Note that some authors consider instead a *utility function*, whose value they aim to maximize. These are equivalent concepts if we take the utility to be simply the negative of the loss, and throughout this text we shall use the loss function convention. Suppose that, for a new value of $\mathbf{x}$, the true class is $\mathcal{C}_k$ and that we assign $\mathbf{x}$ to class $\mathcal{C}_j$ (where j may or may not be equal to k). In so doing, we incur some level of loss that we denote by L_{kj}, which we can view as the k, j element of a *loss matrix*. For instance, in our cancer example, we might have a loss matrix of the form shown in Figure 1.25. This particular loss matrix says that there is no loss incurred if the correct decision is made, there is a loss of 1 if a healthy patient is diagnosed as having cancer, whereas there is a loss of 1000 if a patient having cancer is diagnosed as healthy.

The optimal solution is the one which minimizes the loss function. However, the loss function depends on the true class, which is unknown. For a given input vector $\mathbf{x}$, our uncertainty in the true class is expressed through the joint probability distribution $p(\mathbf{x}, \mathcal{C}_k)$ and so we seek instead to minimize the average loss, where the average is computed with respect to this distribution, which is given by

$$\mathbb{E}[L] = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj} p(\mathbf{x}, \mathcal{C}_k)\, \mathrm{d}\mathbf{x}. \tag{1.80}$$

Each $\mathbf{x}$ can be assigned independently to one of the decision regions $\mathcal{R}_j$. Our goal is to choose the regions $\mathcal{R}_j$ in order to minimize the expected loss (1.80), which implies that for each $\mathbf{x}$ we should minimize $\sum_k L_{kj} p(\mathbf{x}, \mathcal{C}_k)$. As before, we can use the product rule $p(\mathbf{x}, \mathcal{C}_k) = p(\mathcal{C}_k|\mathbf{x})p(\mathbf{x})$ to eliminate the common factor of $p(\mathbf{x})$. Thus the decision rule that minimizes the expected loss is the one that assigns each

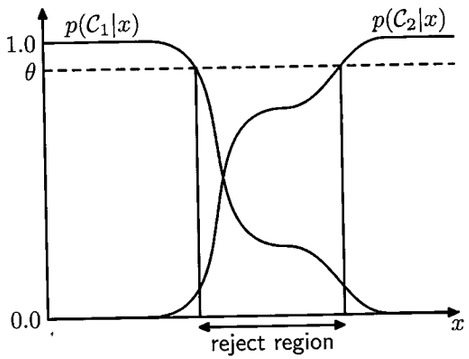


Figure 1.26 Illustration of the reject option. Inputs x such that the larger of the two posterior probabilities is less than or equal to some threshold θ will be rejected.

new $\mathbf{x}$ to the class j for which the quantity

$$\sum_k L_{kj} p(\mathcal{C}_k|\mathbf{x}) \tag{1.81}$$

is a minimum. This is clearly trivial to do, once we know the posterior class probabilities $p(\mathcal{C}_k|\mathbf{x})$.

1.5.3 The reject option

We have seen that classification errors arise from the regions of input space where the largest of the posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$ is significantly less than unity, or equivalently where the joint distributions $p(\mathbf{x}, \mathcal{C}_k)$ have comparable values. These are the regions where we are relatively uncertain about class membership. In some applications, it will be appropriate to avoid making decisions on the difficult cases in anticipation of a lower error rate on those examples for which a classification decision is made. This is known as the *reject option*. For example, in our hypothetical medical illustration, it may be appropriate to use an automatic system to classify those X-ray images for which there is little doubt as to the correct class, while leaving a human expert to classify the more ambiguous cases. We can achieve this by introducing a threshold θ and rejecting those inputs $\mathbf{x}$ for which the largest of the posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$ is less than or equal to θ. This is illustrated for the case of two classes, and a single continuous input variable x, in Figure 1.26. Note that setting $\theta = 1$ will ensure that all examples are rejected, whereas if there are K classes then setting $\theta < 1/K$ will ensure that no examples are rejected. Thus the fraction of examples that get rejected is controlled by the value of θ.

We can easily extend the reject criterion to minimize the expected loss, when a loss matrix is given, taking account of the loss incurred when a reject decision is made.

Exercise 1.24

1.5.4 Inference and decision

We have broken the classification problem down into two separate stages, the *inference stage* in which we use training data to learn a model for $p(\mathcal{C}_k|\mathbf{x})$, and the

subsequent *decision* stage in which we use these posterior probabilities to make optimal class assignments. An alternative possibility would be to solve both problems together and simply learn a function that maps inputs $\mathbf{x}$ directly into decisions. Such a function is called a *discriminant function*.

In fact, we can identify three distinct approaches to solving decision problems, all of which have been used in practical applications. These are given, in decreasing order of complexity, by:

(a) First solve the inference problem of determining the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ for each class $\mathcal{C}_k$ individually. Also separately infer the prior class probabilities $p(\mathcal{C}_k)$. Then use Bayes' theorem in the form

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})} \tag{1.82}$$

to find the posterior class probabilities $p(\mathcal{C}_k|\mathbf{x})$. As usual, the denominator in Bayes' theorem can be found in terms of the quantities appearing in the numerator, because

$$p(\mathbf{x}) = \sum_k p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k). \tag{1.83}$$

Equivalently, we can model the joint distribution $p(\mathbf{x}, \mathcal{C}_k)$ directly and then normalize to obtain the posterior probabilities. Having found the posterior probabilities, we use decision theory to determine class membership for each new input $\mathbf{x}$. Approaches that explicitly or implicitly model the distribution of inputs as well as outputs are known as *generative models*, because by sampling from them it is possible to generate synthetic data points in the input space.

(b) First solve the inference problem of determining the posterior class probabilities $p(\mathcal{C}_k|\mathbf{x})$, and then subsequently use decision theory to assign each new $\mathbf{x}$ to one of the classes. Approaches that model the posterior probabilities directly are called *discriminative models*.

(c) Find a function $f(\mathbf{x})$, called a discriminant function, which maps each input $\mathbf{x}$ directly onto a class label. For instance, in the case of two-class problems, $f(\cdot)$ might be binary valued and such that $f = 0$ represents class $\mathcal{C}_1$ and $f = 1$ represents class $\mathcal{C}_2$. In this case, probabilities play no role.

Let us consider the relative merits of these three alternatives. Approach (a) is the most demanding because it involves finding the joint distribution over both $\mathbf{x}$ and $\mathcal{C}_k$. For many applications, $\mathbf{x}$ will have high dimensionality, and consequently we may need a large training set in order to be able to determine the class-conditional densities to reasonable accuracy. Note that the class priors $p(\mathcal{C}_k)$ can often be estimated simply from the fractions of the training set data points in each of the classes. One advantage of approach (a), however, is that it also allows the marginal density of data $p(\mathbf{x})$ to be determined from (1.83). This can be useful for detecting new data points that have low probability under the model and for which the predictions may

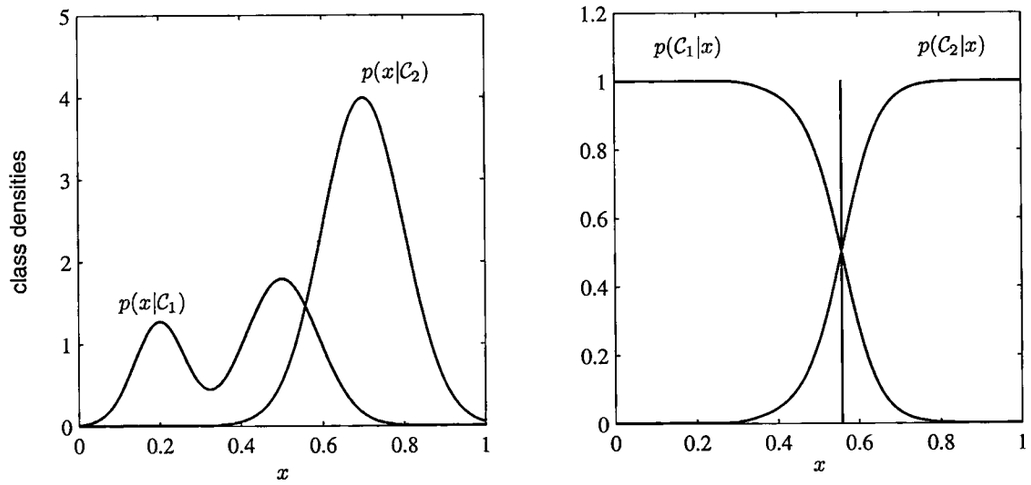


Figure 1.27 Example of the class-conditional densities for two classes having a single input variable x (left plot) together with the corresponding posterior probabilities (right plot). Note that the left-hand mode of the class-conditional density $p(\mathbf{x}|\mathcal{C}_1)$, shown in blue on the left plot, has no effect on the posterior probabilities. The vertical green line in the right plot shows the decision boundary in x that gives the minimum misclassification rate.

be of low accuracy, which is known as *outlier detection* or *novelty detection* (Bishop, 1994; Tarassenko, 1995).

However, if we only wish to make classification decisions, then it can be wasteful of computational resources, and excessively demanding of data, to find the joint distribution $p(\mathbf{x}, \mathcal{C}_k)$ when in fact we only really need the posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$, which can be obtained directly through approach (b). Indeed, the class-conditional densities may contain a lot of structure that has little effect on the posterior probabilities, as illustrated in Figure 1.27. There has been much interest in exploring the relative merits of generative and discriminative approaches to machine learning, and in finding ways to combine them (Jebara, 2004; Lasserre *et al.*, 2006).

An even simpler approach is (c) in which we use the training data to find a discriminant function $f(\mathbf{x})$ that maps each $\mathbf{x}$ directly onto a class label, thereby combining the inference and decision stages into a single learning problem. In the example of Figure 1.27, this would correspond to finding the value of x shown by the vertical green line, because this is the decision boundary giving the minimum probability of misclassification.

With option (c), however, we no longer have access to the posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$. There are many powerful reasons for wanting to compute the posterior probabilities, even if we subsequently use them to make decisions. These include:

Minimizing risk. Consider a problem in which the elements of the loss matrix are subjected to revision from time to time (such as might occur in a financial application). If we know the posterior probabilities, we can trivially revise the minimum risk decision criterion by modifying (1.81) appropriately. If we have only a discriminant function, then any change to the loss matrix would require that we return to the training data and solve the classification problem afresh.

Reject option. Posterior probabilities allow us to determine a rejection criterion that will minimize the misclassification rate, or more generally the expected loss, for a given fraction of rejected data points.

Compensating for class priors. Consider our medical X-ray problem again, and suppose that we have collected a large number of X-ray images from the general population for use as training data in order to build an automated screening system. Because cancer is rare amongst the general population, we might find that, say, only 1 in every 1,000 examples corresponds to the presence of cancer. If we used such a data set to train an adaptive model, we could run into severe difficulties due to the small proportion of the cancer class. For instance, a classifier that assigned every point to the normal class would already achieve 99.9% accuracy and it would be difficult to avoid this trivial solution. Also, even a large data set will contain very few examples of X-ray images corresponding to cancer, and so the learning algorithm will not be exposed to a broad range of examples of such images and hence is not likely to generalize well. A balanced data set in which we have selected equal numbers of examples from each of the classes would allow us to find a more accurate model. However, we then have to compensate for the effects of our modifications to the training data. Suppose we have used such a modified data set and found models for the posterior probabilities. From Bayes' theorem (1.82), we see that the posterior probabilities are proportional to the prior probabilities, which we can interpret as the fractions of points in each class. We can therefore simply take the posterior probabilities obtained from our artificially balanced data set and first divide by the class fractions in that data set and then multiply by the class fractions in the population to which we wish to apply the model. Finally, we need to normalize to ensure that the new posterior probabilities sum to one. Note that this procedure cannot be applied if we have learned a discriminant function directly instead of determining posterior probabilities.

Combining models. For complex applications, we may wish to break the problem into a number of smaller subproblems each of which can be tackled by a separate module. For example, in our hypothetical medical diagnosis problem, we may have information available from, say, blood tests as well as X-ray images. Rather than combine all of this heterogeneous information into one huge input space, it may be more effective to build one system to interpret the X-ray images and a different one to interpret the blood data. As long as each of the two models gives posterior probabilities for the classes, we can combine the outputs systematically using the rules of probability. One simple way to do this is to assume that, for each class separately, the distributions of inputs for the X-ray images, denoted by $\mathbf{x}_\mathrm{I}$, and the blood data, denoted by $\mathbf{x}_\mathrm{B}$, are

independent, so that

$$p(\mathbf{x}_\mathrm{I}, \mathbf{x}_\mathrm{B}|\mathcal{C}_k) = p(\mathbf{x}_\mathrm{I}|\mathcal{C}_k)p(\mathbf{x}_\mathrm{B}|\mathcal{C}_k). \tag{1.84}$$

Section 8.2 This is an example of *conditional independence* property, because the independence holds when the distribution is conditioned on the class $\mathcal{C}_k$. The posterior probability, given both the X-ray and blood data, is then given by

$$\begin{aligned} p(\mathcal{C}_k|\mathbf{x}_\mathrm{I}, \mathbf{x}_\mathrm{B}) &\propto p(\mathbf{x}_\mathrm{I}, \mathbf{x}_\mathrm{B}|\mathcal{C}_k)p(\mathcal{C}_k) \\ &\propto p(\mathbf{x}_\mathrm{I}|\mathcal{C}_k)p(\mathbf{x}_\mathrm{B}|\mathcal{C}_k)p(\mathcal{C}_k) \\ &\propto \frac{p(\mathcal{C}_k|\mathbf{x}_\mathrm{I})p(\mathcal{C}_k|\mathbf{x}_\mathrm{B})}{p(\mathcal{C}_k)} \end{aligned} \tag{1.85}$$

Thus we need the class prior probabilities $p(\mathcal{C}_k)$, which we can easily estimate from the fractions of data points in each class, and then we need to normalize the resulting posterior probabilities so they sum to one. The particular conditional independence assumption (1.84) is an example of the *naive Bayes model*. *Section 8.2.2* Note that the joint marginal distribution $p(\mathbf{x}_\mathrm{I}, \mathbf{x}_\mathrm{B})$ will typically not factorize under this model. We shall see in later chapters how to construct models for combining data that do not require the conditional independence assumption (1.84).

1.5.5 Loss functions for regression

So far, we have discussed decision theory in the context of classification problems. We now turn to the case of regression problems, such as the curve fitting *Section 1.1* example discussed earlier. The decision stage consists of choosing a specific estimate $y(\mathbf{x})$ of the value of t for each input $\mathbf{x}$. Suppose that in doing so, we incur a loss $L(t, y(\mathbf{x}))$. The average, or expected, loss is then given by

$$\mathbb{E}[L] = \iint L(t, y(\mathbf{x}))p(\mathbf{x}, t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t. \tag{1.86}$$

A common choice of loss function in regression problems is the squared loss given by $L(t, y(\mathbf{x})) = \{y(\mathbf{x}) - t\}^2$. In this case, the expected loss can be written

$$\mathbb{E}[L] = \iint \{y(\mathbf{x}) - t\}^2 p(\mathbf{x}, t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t. \tag{1.87}$$

Our goal is to choose $y(\mathbf{x})$ so as to minimize $\mathbb{E}[L]$. If we assume a completely *Appendix D* flexible function $y(\mathbf{x})$, we can do this formally using the calculus of variations to give

$$\frac{\delta \mathbb{E}[L]}{\delta y(\mathbf{x})} = 2\int \{y(\mathbf{x}) - t\}p(\mathbf{x}, t)\,\mathrm{d}t = 0. \tag{1.88}$$

Solving for $y(\mathbf{x})$, and using the sum and product rules of probability, we obtain

$$y(\mathbf{x}) = \frac{\int t p(\mathbf{x}, t)\,\mathrm{d}t}{p(\mathbf{x})} = \int t p(t|\mathbf{x})\,\mathrm{d}t = \mathbb{E}_t[t|\mathbf{x}] \tag{1.89}$$

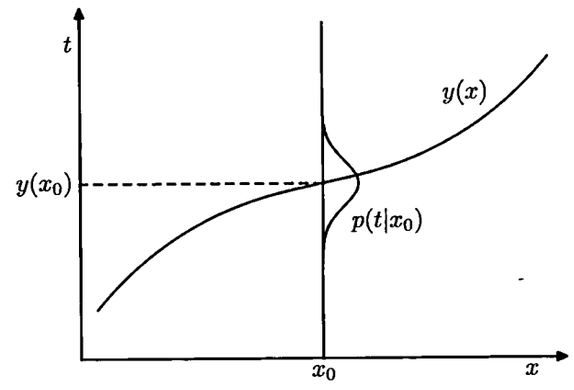


Figure 1.28 The regression function $y(x)$, which minimizes the expected squared loss, is given by the mean of the conditional distribution $p(t|x)$.

which is the conditional average of t conditioned on $\mathbf{x}$ and is known as the *regression function*. This result is illustrated in Figure 1.28. It can readily be extended to multiple target variables represented by the vector $\mathbf{t}$, in which case the optimal solution *Exercise 1.25* is the conditional average $\mathbf{y}(\mathbf{x}) = \mathbb{E}_t[\mathbf{t}|\mathbf{x}]$.

We can also derive this result in a slightly different way, which will also shed light on the nature of the regression problem. Armed with the knowledge that the optimal solution is the conditional expectation, we can expand the square term as follows

$$\begin{aligned} \{y(\mathbf{x}) - t\}^2 &= \{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}] + \mathbb{E}[t|\mathbf{x}] - t\}^2 \\ &= \{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}^2 + 2\{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}\{\mathbb{E}[t|\mathbf{x}] - t\} + \{\mathbb{E}[t|\mathbf{x}] - t\}^2 \end{aligned}$$

where, to keep the notation uncluttered, we use $\mathbb{E}[t|\mathbf{x}]$ to denote $\mathbb{E}_t[t|\mathbf{x}]$. Substituting into the loss function and performing the integral over t, we see that the cross-term vanishes and we obtain an expression for the loss function in the form

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int \{\mathbb{E}[t|\mathbf{x}] - t\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{1.90}$$

The function $y(\mathbf{x})$ we seek to determine enters only in the first term, which will be minimized when $y(\mathbf{x})$ is equal to $\mathbb{E}[t|\mathbf{x}]$, in which case this term will vanish. This is simply the result that we derived previously and that shows that the optimal least squares predictor is given by the conditional mean. The second term is the variance of the distribution of t, averaged over $\mathbf{x}$. It represents the intrinsic variability of the target data and can be regarded as noise. Because it is independent of $y(\mathbf{x})$, it represents the irreducible minimum value of the loss function.

As with the classification problem, we can either determine the appropriate probabilities and then use these to make optimal decisions, or we can build models that make decisions directly. Indeed, we can identify three distinct approaches to solving regression problems given, in order of decreasing complexity, by:

(a) First solve the inference problem of determining the joint density $p(\mathbf{x}, t)$. Then normalize to find the conditional density $p(t|\mathbf{x})$, and finally marginalize to find the conditional mean given by (1.89).

(b) First solve the inference problem of determining the conditional density $p(t|\mathbf{x})$, and then subsequently marginalize to find the conditional mean given by (1.89).

(c) Find a regression function $y(\mathbf{x})$ directly from the training data.

The relative merits of these three approaches follow the same lines as for classification problems above.

The squared loss is not the only possible choice of loss function for regression. Indeed, there are situations in which squared loss can lead to very poor results and where we need to develop more sophisticated approaches. An important example concerns situations in which the conditional distribution $p(t|\mathbf{x})$ is multimodal, as often arises in the solution of inverse problems. Here we consider briefly one simple generalization of the squared loss, called the *Minkowski* loss, whose expectation is given by

Section 5.6

$$\mathbb{E}[L_q] = \iint |y(\mathbf{x}) - t|^q p(\mathbf{x}, t)\, \mathrm{d}\mathbf{x}\, \mathrm{d}t \tag{1.91}$$

which reduces to the expected squared loss for $q = 2$. The function $|y - t|^q$ is plotted against $y - t$ for various values of q in Figure 1.29. The minimum of $\mathbb{E}[L_q]$ is given by the conditional mean for $q = 2$, the conditional median for $q = 1$, and the conditional mode for $q \to 0$.

Exercise 1.27

1.6. Information Theory

In this chapter, we have discussed a variety of concepts from probability theory and decision theory that will form the foundations for much of the subsequent discussion in this book. We close this chapter by introducing some additional concepts from the field of information theory, which will also prove useful in our development of pattern recognition and machine learning techniques. Again, we shall focus only on the key concepts, and we refer the reader elsewhere for more detailed discussions (Viterbi and Omura, 1979; Cover and Thomas, 1991; MacKay, 2003) .

We begin by considering a discrete random variable x and we ask how much information is received when we observe a specific value for this variable. The amount of information can be viewed as the 'degree of surprise' on learning the value of x. If we are told that a highly improbable event has just occurred, we will have received more information than if we were told that some very likely event has just occurred, and if we knew that the event was certain to happen we would receive no information. Our measure of information content will therefore depend on the probability distribution $p(x)$, and we therefore look for a quantity $h(x)$ that is a monotonic function of the probability $p(x)$ and that expresses the information content. The form of $h(\cdot)$ can be found by noting that if we have two events x and y that are unrelated, then the information gain from observing both of them should be the sum of the information gained from each of them separately, so that $h(x, y) = h(x) + h(y)$. Two unrelated events will be statistically independent and so $p(x, y) = p(x)p(y)$. From these two relationships, it is easily shown that $h(x)$ must be given by the logarithm of $p(x)$ and so we have

Exercise 1.28

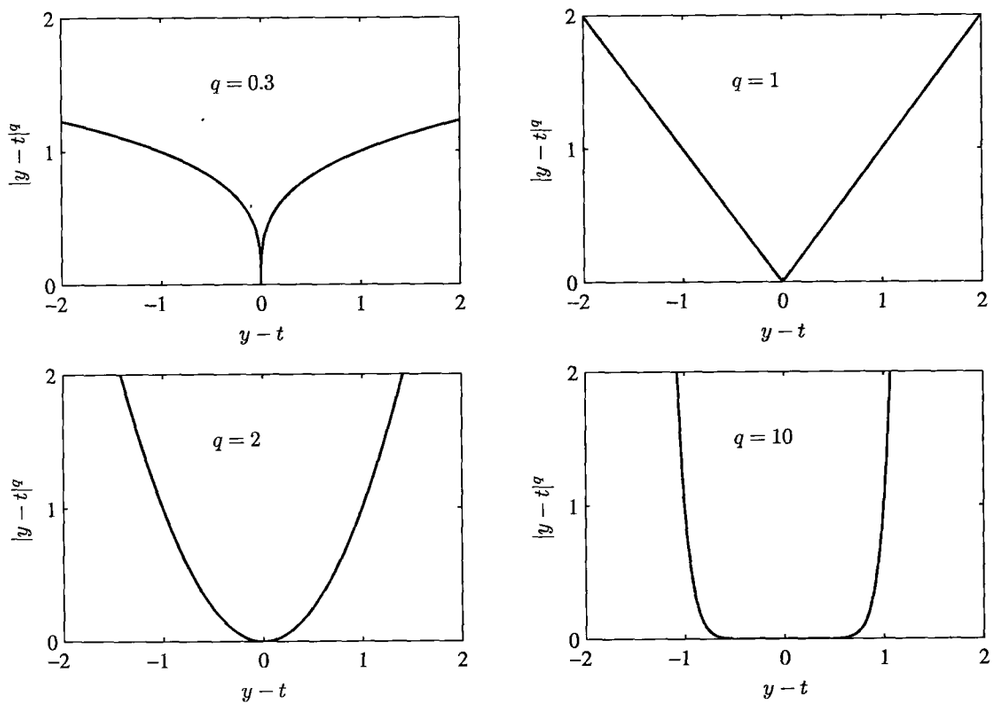


Figure 1.29 Plots of the quantity $L_q = |y - t|^q$ for various values of q.

$$h(x) = -\log_2 p(x) \tag{1.92}$$

where the negative sign ensures that information is positive or zero. Note that low probability events x correspond to high information content. The choice of basis for the logarithm is arbitrary, and for the moment we shall adopt the convention prevalent in information theory of using logarithms to the base of 2. In this case, as we shall see shortly, the units of $h(x)$ are bits ('binary digits').

Now suppose that a sender wishes to transmit the value of a random variable to a receiver. The average amount of information that they transmit in the process is obtained by taking the expectation of (1.92) with respect to the distribution $p(x)$ and is given by

$$\mathrm{H}[x] = -\sum_x p(x) \log_2 p(x). \tag{1.93}$$

This important quantity is called the *entropy* of the random variable x. Note that $\lim_{p \to 0} p \ln p = 0$ and so we shall take $p(x) \ln p(x) = 0$ whenever we encounter a value for x such that $p(x) = 0$.

So far we have given a rather heuristic motivation for the definition of informa-

tion (1.92) and the corresponding entropy (1.93). We now show that these definitions indeed possess useful properties. Consider a random variable x having 8 possible states, each of which is equally likely. In order to communicate the value of x to a receiver, we would need to transmit a message of length 3 bits. Notice that the entropy of this variable is given by

$$\mathrm{H}[x] = -8 \times \frac{1}{8} \log_2 \frac{1}{8} = 3 \text{ bits}.$$

Now consider an example (Cover and Thomas, 1991) of a variable having 8 possible states $\{a, b, c, d, e, f, g, h\}$ for which the respective probabilities are given by $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64})$. The entropy in this case is given by

$$\mathrm{H}[x] = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{8}\log_2\frac{1}{8} - \frac{1}{16}\log_2\frac{1}{16} - \frac{4}{64}\log_2\frac{1}{64} = 2 \text{ bits}.$$

We see that the nonuniform distribution has a smaller entropy than the uniform one, and we shall gain some insight into this shortly when we discuss the interpretation of entropy in terms of disorder. For the moment, let us consider how we would transmit the identity of the variable's state to a receiver. We could do this, as before, using a 3-bit number. However, we can take advantage of the nonuniform distribution by using shorter codes for the more probable events, at the expense of longer codes for the less probable events, in the hope of getting a shorter average code length. This can be done by representing the states $\{a, b, c, d, e, f, g, h\}$ using, for instance, the following set of code strings: 0, 10, 110, 1110, 111100, 111101, 111110, 111111. The average length of the code that has to be transmitted is then

$$\text{average code length} = \frac{1}{2} \times 1 + \frac{1}{4} \times 2 + \frac{1}{8} \times 3 + \frac{1}{16} \times 4 + 4 \times \frac{1}{64} \times 6 = 2 \text{ bits}$$

which again is the same as the entropy of the random variable. Note that shorter code strings cannot be used because it must be possible to disambiguate a concatenation of such strings into its component parts. For instance, 11001110 decodes uniquely into the state sequence c, a, d.

This relation between entropy and shortest coding length is a general one. The *noiseless coding theorem* (Shannon, 1948) states that the entropy is a lower bound on the number of bits needed to transmit the state of a random variable.

From now on, we shall switch to the use of natural logarithms in defining entropy, as this will provide a more convenient link with ideas elsewhere in this book. In this case, the entropy is measured in units of 'nats' instead of bits, which differ simply by a factor of $\ln 2$.

We have introduced the concept of entropy in terms of the average amount of information needed to specify the state of a random variable. In fact, the concept of entropy has much earlier origins in physics where it was introduced in the context of equilibrium thermodynamics and later given a deeper interpretation as a measure of disorder through developments in statistical mechanics. We can understand this alternative view of entropy by considering a set of N identical objects that are to be divided amongst a set of bins, such that there are n_i objects in the i^{th} bin. Consider

the number of different ways of allocating the objects to the bins. There are N ways to choose the first object, $(N-1)$ ways to choose the second object, and so on, leading to a total of $N!$ ways to allocate all N objects to the bins, where $N!$ (pronounced 'factorial N') denotes the product $N \times (N-1) \times \cdots \times 2 \times 1$. However, we don't wish to distinguish between rearrangements of objects within each bin. In the i^{th} bin there are $n_i!$ ways of reordering the objects, and so the total number of ways of allocating the N objects to the bins is given by

$$W = \frac{N!}{\prod_i n_i!} \tag{1.94}$$

which is called the *multiplicity*. The entropy is then defined as the logarithm of the multiplicity scaled by an appropriate constant

$$\mathrm{H} = \frac{1}{N} \ln W = \frac{1}{N} \ln N! - \frac{1}{N} \sum_i \ln n_i!. \tag{1.95}$$

We now consider the limit $N \to \infty$, in which the fractions n_i/N are held fixed, and apply Stirling's approximation

$$\ln N! \simeq N \ln N - N \tag{1.96}$$

which gives

$$\mathrm{H} = -\lim_{N\to\infty} \sum_i \left(\frac{n_i}{N}\right) \ln\left(\frac{n_i}{N}\right) = -\sum_i p_i \ln p_i \tag{1.97}$$

where we have used $\sum_i n_i = N$. Here $p_i = \lim_{N\to\infty}(n_i/N)$ is the probability of an object being assigned to the i^{th} bin. In physics terminology, the specific arrangements of objects in the bins is called a *microstate*, and the overall distribution of occupation numbers, expressed through the ratios n_i/N, is called a *macrostate*. The multiplicity W is also known as the *weight* of the macrostate.

We can interpret the bins as the states x_i of a discrete random variable X, where $p(X = x_i) = p_i$. The entropy of the random variable X is then

$$\mathrm{H}[p] = -\sum_i p(x_i) \ln p(x_i). \tag{1.98}$$

Distributions $p(x_i)$ that are sharply peaked around a few values will have a relatively low entropy, whereas those that are spread more evenly across many values will have higher entropy, as illustrated in Figure 1.30. Because $0 \leqslant p_i \leqslant 1$, the entropy is nonnegative, and it will equal its minimum value of 0 when one of the $p_i = 1$ and all other $p_{j\neq i} = 0$. The maximum entropy configuration can be found by maximizing H using a Lagrange multiplier to enforce the normalization constraint on the probabilities. Thus we maximize

Appendix E

$$\widetilde{\mathrm{H}} = -\sum_i p(x_i) \ln p(x_i) + \lambda \left(\sum_i p(x_i) - 1 \right) \tag{1.99}$$

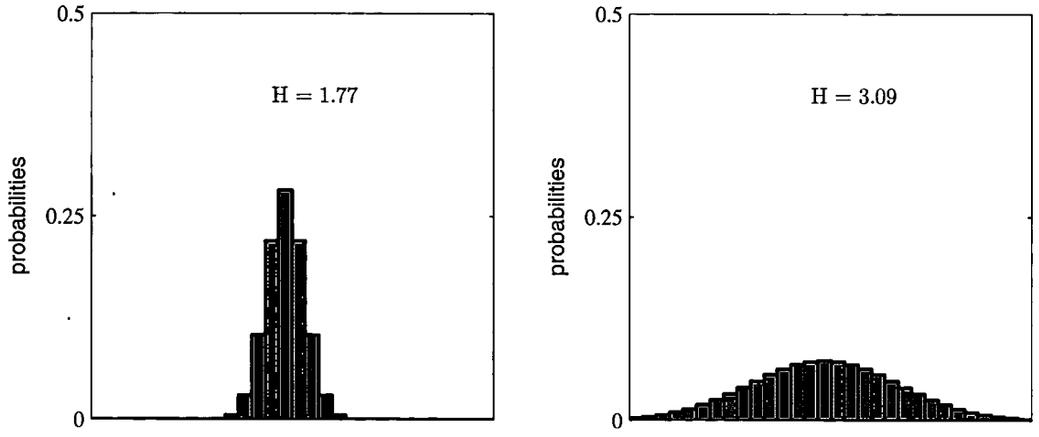


Figure 1.30 Histograms of two probability distributions over 30 bins illustrating the higher value of the entropy H for the broader distribution. The largest entropy would arise from a uniform distribution that would give $\mathrm{H} = -\ln(1/30) = 3.40$.

from which we find that all of the $p(x_i)$ are equal and are given by $p(x_i) = 1/M$ where M is the total number of states x_i. The corresponding value of the entropy is then $\mathrm{H} = \ln M$. This result can also be derived from Jensen's inequality (to be discussed shortly). To verify that the stationary point is indeed a maximum, we can evaluate the second derivative of the entropy, which gives

Exercise 1.29

$$\frac{\partial \widetilde{\mathrm{H}}}{\partial p(x_i)\partial p(x_j)} = -I_{ij}\frac{1}{p_i} \tag{1.100}$$

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

We can extend the definition of entropy to include distributions $p(x)$ over continuous variables x as follows. First divide x into bins of width Δ. Then, assuming $p(x)$ is continuous, the *mean value theorem* (Weisstein, 1999) tells us that, for each such bin, there must exist a value x_i such that

$$\int_{i\Delta}^{(i+1)\Delta} p(x)\,\mathrm{d}x = p(x_i)\Delta. \tag{1.101}$$

We can now quantize the continuous variable x by assigning any value x to the value x_i whenever x falls in the i^{th} bin. The probability of observing the value x_i is then $p(x_i)\Delta$. This gives a discrete distribution for which the entropy takes the form

$$\mathrm{H}_\Delta = -\sum_i p(x_i)\Delta \ln\left(p(x_i)\Delta\right) = -\sum_i p(x_i)\Delta \ln p(x_i) - \ln\Delta \tag{1.102}$$

where we have used $\sum_i p(x_i)\Delta = 1$, which follows from (1.101). We now omit the second term $-\ln\Delta$ on the right-hand side of (1.102) and then consider the limit

$\Delta \to 0$. The first term on the right-hand side of (1.102) will approach the integral of $p(x)\ln p(x)$ in this limit so that

$$\lim_{\Delta\to 0}\left\{\sum_i p(x_i)\Delta \ln p(x_i)\right\} = -\int p(x)\ln p(x)\,\mathrm{d}x \tag{1.103}$$

where the quantity on the right-hand side is called the *differential entropy*. We see that the discrete and continuous forms of the entropy differ by a quantity $\ln\Delta$, which diverges in the limit $\Delta \to 0$. This reflects the fact that to specify a continuous variable very precisely requires a large number of bits. For a density defined over multiple continuous variables, denoted collectively by the vector $\mathbf{x}$, the differential entropy is given by

$$\mathrm{H}[\mathbf{x}] = -\int p(\mathbf{x})\ln p(\mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{1.104}$$

In the case of discrete distributions, we saw that the maximum entropy configuration corresponded to an equal distribution of probabilities across the possible states of the variable. Let us now consider the maximum entropy configuration for a continuous variable. In order for this maximum to be well defined, it will be necessary to constrain the first and second moments of $p(x)$ as well as preserving the normalization constraint. We therefore maximize the differential entropy with the

Ludwig Boltzmann
1844–1906

Ludwig Eduard Boltzmann was an Austrian physicist who created the field of statistical mechanics. Prior to Boltzmann, the concept of entropy was already known from classical thermodynamics where it was stated the fact that when we take energy from a system, not all of that energy is typically available to do useful work. Boltzmann showed that the thermodynamic entropy S, a macroscopic quantity, could be related to the statistical properties at the microscopic level. This is expressed through the famous equation $S = k\ln W$ in which W represents the number of possible microstates in a macrostate, and $k \simeq 1.38 \times 10^{-23}$ (in units of Joules per Kelvin) is known as Boltzmann's constant. Boltzmann's ideas were disputed by many scientists of they day. One difficulty they had arose from the second law of thermodynamics, which states that the entropy of a closed system tends to increase with time. By contrast, at the microscopic level the classical Newtonian equations of physics are reversible, and so they found it difficult to see how the latter could explain the former. They didn't fully appreciate Boltzmann's arguments, which were statistical in nature and which concluded not that entropy could never decrease over time but simply that with overwhelming probability it would generally increase. Boltzmann even had a long-running dispute with the editor of the leading German physics journal who refused to let him refer to atoms and molecules as anything other than convenient theoretical constructs. The continued attacks on his work lead to bouts of depression, and eventually he committed suicide. Shortly after Boltzmann's death, new experiments by Perrin on colloidal suspensions verified his theories and confirmed the value of the Boltzmann constant. The equation $S = k\ln W$ is carved on Boltzmann's tombstone.

three constraints

$$\int_{-\infty}^{\infty} p(x)\,\mathrm{d}x = 1 \tag{1.105}$$

$$\int_{-\infty}^{\infty} xp(x)\,\mathrm{d}x = \mu \tag{1.106}$$

$$\int_{-\infty}^{\infty} (x-\mu)^2 p(x)\,\mathrm{d}x = \sigma^2. \tag{1.107}$$

Appendix E The constrained maximization can be performed using Lagrange multipliers so that we maximize the following functional with respect to $p(x)$

$$-\int_{-\infty}^{\infty} p(x)\ln p(x)\,\mathrm{d}x + \lambda_1\left(\int_{-\infty}^{\infty} p(x)\,\mathrm{d}x - 1\right) + \lambda_2\left(\int_{-\infty}^{\infty} xp(x)\,\mathrm{d}x - \mu\right) + \lambda_3\left(\int_{-\infty}^{\infty} (x-\mu)^2 p(x)\,\mathrm{d}x - \sigma^2\right).$$

Appendix D Using the calculus of variations, we set the derivative of this functional to zero giving

$$p(x) = \exp\left\{-1 + \lambda_1 + \lambda_2 x + \lambda_3 (x-\mu)^2\right\}. \tag{1.108}$$

Exercise 1.34 The Lagrange multipliers can be found by back substitution of this result into the three constraint equations, leading finally to the result

$$p(x) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} \tag{1.109}$$

and so the distribution that maximizes the differential entropy is the Gaussian. Note that we did not constrain the distribution to be nonnegative when we maximized the entropy. However, because the resulting distribution is indeed nonnegative, we see with hindsight that such a constraint is not necessary.

Exercise 1.35 If we evaluate the differential entropy of the Gaussian, we obtain

$$\mathrm{H}[x] = \frac{1}{2}\left\{1 + \ln(2\pi\sigma^2)\right\}. \tag{1.110}$$

Thus we see again that the entropy increases as the distribution becomes broader, i.e., as σ^2 increases. This result also shows that the differential entropy, unlike the discrete entropy, can be negative, because $\mathrm{H}(x) < 0$ in (1.110) for $\sigma^2 < 1/(2\pi e)$.

Suppose we have a joint distribution $p(\mathbf{x}, \mathbf{y})$ from which we draw pairs of values of $\mathbf{x}$ and $\mathbf{y}$. If a value of $\mathbf{x}$ is already known, then the additional information needed to specify the corresponding value of $\mathbf{y}$ is given by $-\ln p(\mathbf{y}|\mathbf{x})$. Thus the average additional information needed to specify $\mathbf{y}$ can be written as

$$\mathrm{H}[\mathbf{y}|\mathbf{x}] = -\iint p(\mathbf{y}, \mathbf{x}) \ln p(\mathbf{y}|\mathbf{x})\,\mathrm{d}\mathbf{y}\,\mathrm{d}\mathbf{x} \tag{1.111}$$

Exercise 1.37 which is called the *conditional entropy* of $\mathbf{y}$ given $\mathbf{x}$. It is easily seen, using the product rule, that the conditional entropy satisfies the relation

$$\mathrm{H}[\mathbf{x}, \mathbf{y}] = \mathrm{H}[\mathbf{y}|\mathbf{x}] + \mathrm{H}[\mathbf{x}] \tag{1.112}$$

where $\mathrm{H}[\mathbf{x}, \mathbf{y}]$ is the differential entropy of $p(\mathbf{x}, \mathbf{y})$ and $\mathrm{H}[\mathbf{x}]$ is the differential entropy of the marginal distribution $p(\mathbf{x})$. Thus the information needed to describe $\mathbf{x}$ and $\mathbf{y}$ is given by the sum of the information needed to describe $\mathbf{x}$ alone plus the additional information required to specify $\mathbf{y}$ given $\mathbf{x}$.

1.6.1 Relative entropy and mutual information

So far in this section, we have introduced a number of concepts from information theory, including the key notion of entropy. We now start to relate these ideas to pattern recognition. Consider some unknown distribution $p(\mathbf{x})$, and suppose that we have modelled this using an approximating distribution $q(\mathbf{x})$. If we use $q(\mathbf{x})$ to construct a coding scheme for the purpose of transmitting values of $\mathbf{x}$ to a receiver, then the average *additional* amount of information (in nats) required to specify the value of $\mathbf{x}$ (assuming we choose an efficient coding scheme) as a result of using $q(\mathbf{x})$ instead of the true distribution $p(\mathbf{x})$ is given by

$$\begin{aligned}\mathrm{KL}(p\|q) &= -\int p(\mathbf{x}) \ln q(\mathbf{x})\,\mathrm{d}\mathbf{x} - \left(-\int p(\mathbf{x}) \ln p(\mathbf{x})\,\mathrm{d}\mathbf{x}\right) \\ &= -\int p(\mathbf{x}) \ln \left\{\frac{q(\mathbf{x})}{p(\mathbf{x})}\right\} \mathrm{d}\mathbf{x}.\end{aligned} \tag{1.113}$$

This is known as the *relative entropy* or *Kullback-Leibler divergence*, or *KL divergence* (Kullback and Leibler, 1951), between the distributions $p(\mathbf{x})$ and $q(\mathbf{x})$. Note that it is not a symmetrical quantity, that is to say $\mathrm{KL}(p\|q) \not\equiv \mathrm{KL}(q\|p)$.

We now show that the Kullback-Leibler divergence satisfies $\mathrm{KL}(p\|q) \geqslant 0$ with equality if, and only if, $p(\mathbf{x}) = q(\mathbf{x})$. To do this we first introduce the concept of *convex* functions. A function $f(x)$ is said to be convex if it has the property that every chord lies on or above the function, as shown in Figure 1.31. Any value of x in the interval from $x = a$ to $x = b$ can be written in the form $\lambda a + (1-\lambda)b$ where $0 \leqslant \lambda \leqslant 1$. The corresponding point on the chord is given by $\lambda f(a) + (1-\lambda)f(b)$,

Claude Shannon
1916–2001

After graduating from Michigan and MIT, Shannon joined the AT&T Bell Telephone laboratories in 1941. His paper 'A Mathematical Theory of Communication' published in the *Bell System Technical Journal* in 1948 is the foundations for modern information theory. This paper introduced the word 'bit', and his concept that information could be sent as a stream of 1s and 0s paved the way for the communications revolution. It is said that von Neumann recommended to Shannon that he use the term entropy, not only because of its similarity to the quantity used in physics, but also because "nobody knows what entropy really is, so in any discussion you will always have an advantage".

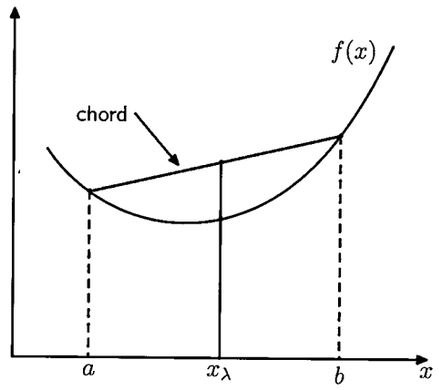


Figure 1.31 A convex function $f(x)$ is one for which every chord (shown in blue) lies on or above the function (shown in red).

and the corresponding value of the function is $f(\lambda a + (1-\lambda)b)$. Convexity then implies

$$f(\lambda a + (1-\lambda)b) \leqslant \lambda f(a) + (1-\lambda)f(b). \tag{1.114}$$

Exercise 1.36 This is equivalent to the requirement that the second derivative of the function be everywhere positive. Examples of convex functions are $x \ln x$ (for $x > 0$) and x^2. A function is called *strictly convex* if the equality is satisfied only for $\lambda = 0$ and $\lambda = 1$. If a function has the opposite property, namely that every chord lies on or below the function, it is called *concave*, with a corresponding definition for *strictly concave*. If a function $f(x)$ is convex, then $-f(x)$ will be concave.

Exercise 1.38 Using the technique of proof by induction, we can show from (1.114) that a convex function $f(x)$ satisfies

$$f\left(\sum_{i=1}^{M} \lambda_i x_i\right) \leqslant \sum_{i=1}^{M} \lambda_i f(x_i) \tag{1.115}$$

where $\lambda_i \geqslant 0$ and $\sum_i \lambda_i = 1$, for any set of points $\{x_i\}$. The result (1.115) is known as *Jensen's inequality*. If we interpret the λ_i as the probability distribution over a discrete variable x taking the values $\{x_i\}$, then (1.115) can be written

$$f(\mathbb{E}[x]) \leqslant \mathbb{E}[f(x)] \tag{1.116}$$

where $\mathbb{E}[\cdot]$ denotes the expectation. For continuous variables, Jensen's inequality takes the form

$$f\left(\int \mathbf{x} p(\mathbf{x})\,\mathrm{d}\mathbf{x}\right) \leqslant \int f(\mathbf{x}) p(\mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{1.117}$$

We can apply Jensen's inequality in the form (1.117) to the Kullback-Leibler divergence (1.113) to give

$$\mathrm{KL}(p\|q) = -\int p(\mathbf{x}) \ln\left\{\frac{q(\mathbf{x})}{p(\mathbf{x})}\right\} \mathrm{d}\mathbf{x} \geqslant -\ln \int q(\mathbf{x})\,\mathrm{d}\mathbf{x} = 0 \tag{1.118}$$

where we have used the fact that $-\ln x$ is a convex function, together with the normalization condition $\int q(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1$. In fact, $-\ln x$ is a strictly convex function, so the equality will hold if, and only if, $q(\mathbf{x}) = p(\mathbf{x})$ for all $\mathbf{x}$. Thus we can interpret the Kullback-Leibler divergence as a measure of the dissimilarity of the two distributions $p(\mathbf{x})$ and $q(\mathbf{x})$.

We see that there is an intimate relationship between data compression and density estimation (i.e., the problem of modelling an unknown probability distribution) because the most efficient compression is achieved when we know the true distribution. If we use a distribution that is different from the true one, then we must necessarily have a less efficient coding, and on average the additional information that must be transmitted is (at least) equal to the Kullback-Leibler divergence between the two distributions.

Suppose that data is being generated from an unknown distribution $p(\mathbf{x})$ that we wish to model. We can try to approximate this distribution using some parametric distribution $q(\mathbf{x}|\boldsymbol{\theta})$, governed by a set of adjustable parameters $\boldsymbol{\theta}$, for example a multivariate Gaussian. One way to determine $\boldsymbol{\theta}$ is to minimize the Kullback-Leibler divergence between $p(\mathbf{x})$ and $q(\mathbf{x}|\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. We cannot do this directly because we don't know $p(\mathbf{x})$. Suppose, however, that we have observed a finite set of training points $\mathbf{x}_n$, for $n = 1, \ldots, N$, drawn from $p(\mathbf{x})$. Then the expectation with respect to $p(\mathbf{x})$ can be approximated by a finite sum over these points, using (1.35), so that

$$\mathrm{KL}(p\|q) \simeq \sum_{n=1}^{N} \{-\ln q(\mathbf{x}_n|\boldsymbol{\theta}) + \ln p(\mathbf{x}_n)\}. \tag{1.119}$$

The second term on the right-hand side of (1.119) is independent of $\boldsymbol{\theta}$, and the first term is the negative log likelihood function for $\boldsymbol{\theta}$ under the distribution $q(\mathbf{x}|\boldsymbol{\theta})$ evaluated using the training set. Thus we see that minimizing this Kullback-Leibler divergence is equivalent to maximizing the likelihood function.

Now consider the joint distribution between two sets of variables $\mathbf{x}$ and $\mathbf{y}$ given by $p(\mathbf{x}, \mathbf{y})$. If the sets of variables are independent, then their joint distribution will factorize into the product of their marginals $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})p(\mathbf{y})$. If the variables are not independent, we can gain some idea of whether they are 'close' to being independent by considering the Kullback-Leibler divergence between the joint distribution and the product of the marginals, given by

$$\begin{aligned} \mathrm{I}[\mathbf{x}, \mathbf{y}] &\equiv \mathrm{KL}(p(\mathbf{x}, \mathbf{y})\|p(\mathbf{x})p(\mathbf{y})) \\ &= -\iint p(\mathbf{x}, \mathbf{y}) \ln\left(\frac{p(\mathbf{x})p(\mathbf{y})}{p(\mathbf{x}, \mathbf{y})}\right) \mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{y} \end{aligned} \tag{1.120}$$

which is called the *mutual information* between the variables $\mathbf{x}$ and $\mathbf{y}$. From the properties of the Kullback-Leibler divergence, we see that $I(\mathbf{x}, \mathbf{y}) \geqslant 0$ with equality if, and only if, $\mathbf{x}$ and $\mathbf{y}$ are independent. Using the sum and product rules of probability, we see that the mutual information is related to the conditional entropy through

Exercise 1.41

$$\mathrm{I}[\mathbf{x}, \mathbf{y}] = \mathrm{H}[\mathbf{x}] - \mathrm{H}[\mathbf{x}|\mathbf{y}] = \mathrm{H}[\mathbf{y}] - \mathrm{H}[\mathbf{y}|\mathbf{x}]. \tag{1.121}$$

Thus we can view the mutual information as the reduction in the uncertainty about $\mathbf{x}$ by virtue of being told the value of $\mathbf{y}$ (or vice versa). From a Bayesian perspective, we can view $p(\mathbf{x})$ as the prior distribution for $\mathbf{x}$ and $p(\mathbf{x}|\mathbf{y})$ as the posterior distribution after we have observed new data $\mathbf{y}$. The mutual information therefore represents the reduction in uncertainty about $\mathbf{x}$ as a consequence of the new observation $\mathbf{y}$.

Exercises

1.1 (⋆) WWW Consider the sum-of-squares error function given by (1.2) in which the function $y(x, \mathbf{w})$ is given by the polynomial (1.1). Show that the coefficients $\mathbf{w} = \{w_i\}$ that minimize this error function are given by the solution to the following set of linear equations

$$\sum_{j=0}^{M} A_{ij} w_j = T_i \tag{1.122}$$

where

$$A_{ij} = \sum_{n=1}^{N} (x_n)^{i+j}, \qquad T_i = \sum_{n=1}^{N} (x_n)^i t_n. \tag{1.123}$$

Here a suffix i or j denotes the index of a component, whereas $(x)^i$ denotes x raised to the power of i.

1.2 (⋆) Write down the set of coupled linear equations, analogous to (1.122), satisfied by the coefficients w_i which minimize the regularized sum-of-squares error function given by (1.4).

1.3 (⋆⋆) Suppose that we have three coloured boxes r (red), b (blue), and g (green). Box r contains 3 apples, 4 oranges, and 3 limes, box b contains 1 apple, 1 orange, and 0 limes, and box g contains 3 apples, 3 oranges, and 4 limes. If a box is chosen at random with probabilities $p(r) = 0.2$, $p(b) = 0.2$, $p(g) = 0.6$, and a piece of fruit is removed from the box (with equal probability of selecting any of the items in the box), then what is the probability of selecting an apple? If we observe that the selected fruit is in fact an orange, what is the probability that it came from the green box?

1.4 (⋆⋆) WWW Consider a probability density $p_x(x)$ defined over a continuous variable x, and suppose that we make a nonlinear change of variable using $x = g(y)$, so that the density transforms according to (1.27). By differentiating (1.27), show that the location $\widehat{y}$ of the maximum of the density in y is not in general related to the location $\widehat{x}$ of the maximum of the density over x by the simple functional relation $\widehat{x} = g(\widehat{y})$ as a consequence of the Jacobian factor. This shows that the maximum of a probability density (in contrast to a simple function) is dependent on the choice of variable. Verify that, in the case of a linear transformation, the location of the maximum transforms in the same way as the variable itself.

1.5 (⋆) Using the definition (1.38) show that $\mathrm{var}[f(x)]$ satisfies (1.39).

1.6 (⋆) Show that if two variables x and y are independent, then their covariance is zero.

1.7 (⋆⋆) WWW In this exercise, we prove the normalization condition (1.48) for the univariate Gaussian. To do this consider, the integral

$$I = \int_{-\infty}^{\infty} \exp\left(-\frac{1}{2\sigma^2}x^2\right) \mathrm{d}x \tag{1.124}$$

which we can evaluate by first writing its square in the form

$$I^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \exp\left(-\frac{1}{2\sigma^2}x^2 - \frac{1}{2\sigma^2}y^2\right) \mathrm{d}x\,\mathrm{d}y. \tag{1.125}$$

Now make the transformation from Cartesian coordinates (x, y) to polar coordinates (r, θ) and then substitute $u = r^2$. Show that, by performing the integrals over θ and u, and then taking the square root of both sides, we obtain

$$I = \left(2\pi\sigma^2\right)^{1/2}. \tag{1.126}$$

Finally, use this result to show that the Gaussian distribution $\mathcal{N}(x|\mu, \sigma^2)$ is normalized.

1.8 (⋆⋆) WWW By using a change of variables, verify that the univariate Gaussian distribution given by (1.46) satisfies (1.49). Next, by differentiating both sides of the normalization condition

$$\int_{-\infty}^{\infty} \mathcal{N}\left(x|\mu, \sigma^2\right) \mathrm{d}x = 1 \tag{1.127}$$

with respect to σ^2, verify that the Gaussian satisfies (1.50). Finally, show that (1.51) holds.

1.9 (⋆) WWW Show that the mode (i.e. the maximum) of the Gaussian distribution (1.46) is given by μ. Similarly, show that the mode of the multivariate Gaussian (1.52) is given by $\boldsymbol{\mu}$.

1.10 (⋆) WWW Suppose that the two variables x and z are statistically independent. Show that the mean and variance of their sum satisfies

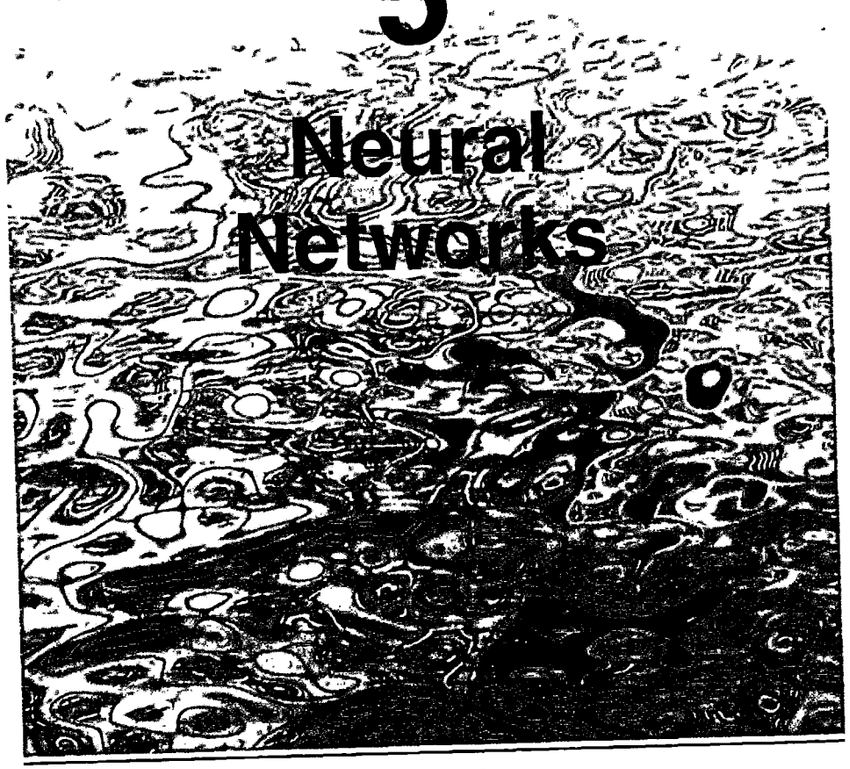
$$\mathbb{E}[x + z] = \mathbb{E}[x] + \mathbb{E}[z] \tag{1.128}$$

$$\mathrm{var}[x + z] = \mathrm{var}[x] + \mathrm{var}[z]. \tag{1.129}$$

1.11 (⋆) By setting the derivatives of the log likelihood function (1.54) with respect to μ and σ^2 equal to zero, verify the results (1.55) and (1.56).

4.26 ($\star\star$) 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

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

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 $x_1, \ldots, x_D$ in the form

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

where $j = 1, \ldots, M$, 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

$$z_j = h(a_j). \tag{5.3}$$

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*

Exercise 5.1

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{5.4}$$

where $k = 1, \ldots, K$, 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

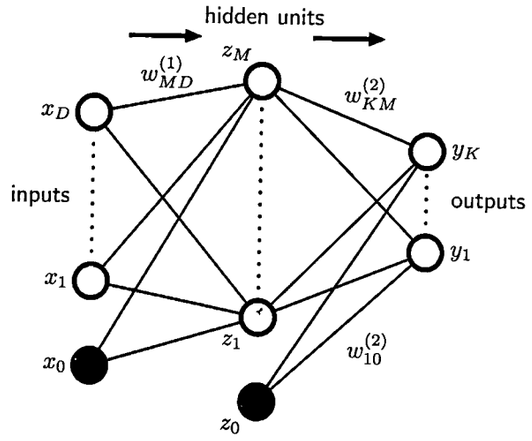


Figure 5.1 Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and 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

$$y_k = \sigma(a_k) \tag{5.5}$$

where

$$\sigma(a) = \frac{1}{1 + \exp(-a)}. \tag{5.6}$$

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

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{5.7}$$

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

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i. \tag{5.8}$$

We can similarly absorb the second-layer biases into the second-layer weights, so that the overall network function becomes

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right). \tag{5.9}$$

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

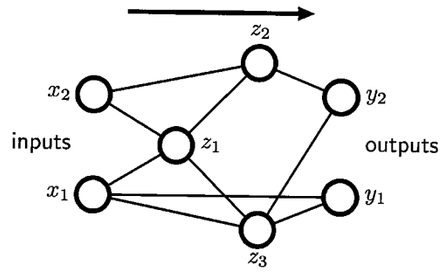


Figure 5.2 Example of a neural network having a general feed-forward topology. Note that each hidden and output unit has an associated bias parameter (omitted for clarity).

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

$$z_k = h\left(\sum_j w_{kj} z_j\right) \tag{5.10}$$

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; Stinchecombe 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

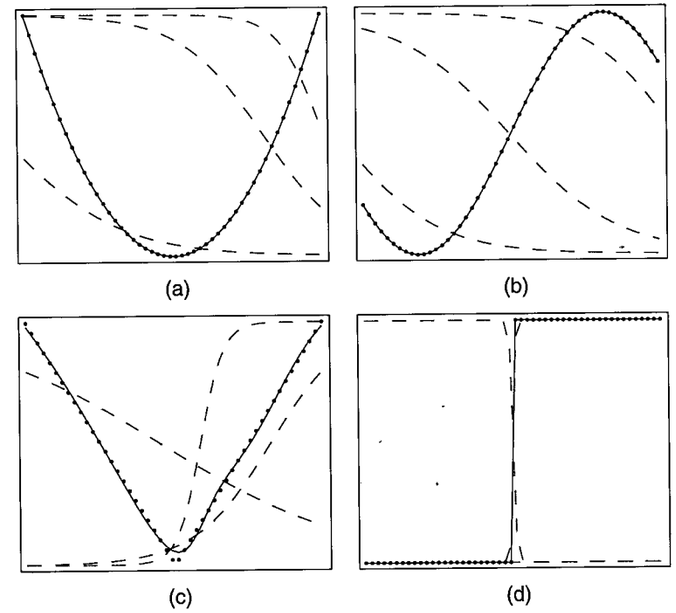


Figure 5.3 Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c), $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown as blue dots, 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 $\mathbf{w}$ 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'

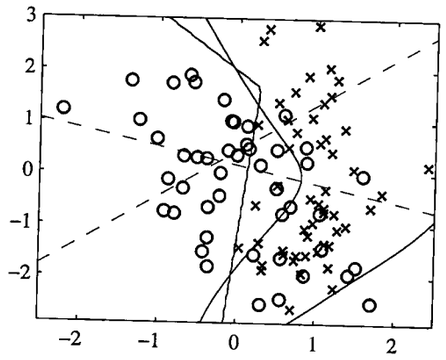


Figure 5.4 Example of the solution of a simple two-class classification problem involving synthetic data using a neural network having two inputs, two hidden units with 'tanh' activation functions, and a single output having a logistic sigmoid activation function. The dashed blue lines show the $z = 0.5$ 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 (Kůrková 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 $n = 1, \ldots, N$, together with a corresponding set of

target vectors $\{\mathbf{t}_n\}$, we minimize the error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2. \tag{5.11}$$

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

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right) \tag{5.12}$$

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 $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, along with corresponding target values $\mathbf{t} = \{t_1, \ldots, t_N\}$, 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

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi) \tag{5.13}$$

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

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 \tag{5.14}$$

where we have discarded additive and multiplicative constants. The value of $\mathbf{w}$ found by minimizing $E(\mathbf{w})$ will be denoted $\mathbf{w}_{\mathrm{ML}}$ 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 $\mathbf{w}_{\mathrm{ML}}$, the value of β can be found by minimizing the negative log likelihood to give

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}_{\mathrm{ML}}) - t_n\}^2. \tag{5.15}$$

Note that this can be evaluated once the iterative optimization required to find $\mathbf{w}_{\mathrm{ML}}$ 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

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(\mathbf{t}|\mathbf{y}(\mathbf{x}, \mathbf{w}), \beta^{-1}\mathbf{I}\right). \tag{5.16}$$

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

Exercise 5.2

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{NK}\sum_{n=1}^{N}\|\mathbf{y}(\mathbf{x}_n, \mathbf{w}_{\mathrm{ML}}) - \mathbf{t}_n\|^2 \tag{5.17}$$

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.

Exercise 5.3

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

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{5.18}$$

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

$$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)} \tag{5.19}$$

so that $0 \leqslant y(\mathbf{x}, \mathbf{w}) \leqslant 1$. 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

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}. \tag{5.20}$$

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

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\} \tag{5.21}$$

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.

Exercise 5.4

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 $k = 1, \ldots, K$. If we assume that the class labels are independent, given the input vector, then the conditional distribution of the targets is

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k}\left[1 - y_k(\mathbf{x}, \mathbf{w})\right]^{1-t_k}. \tag{5.22}$$

Taking the negative logarithm of the corresponding likelihood function then gives the following error function

Exercise 5.5

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk}\ln y_{nk} + (1 - t_{nk})\ln(1 - y_{nk})\} \tag{5.23}$$

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.

Exercise 5.6

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

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn}\ln y_k(\mathbf{x}_n, \mathbf{w}). \tag{5.24}$$

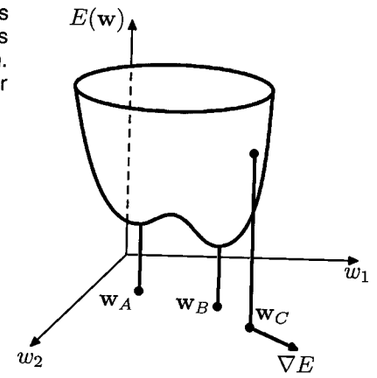


Figure 5.5 Geometrical view of the error function $E(\mathbf{w})$ as a surface sitting over weight space. Point $\mathbf{w}_A$ is a local minimum and $\mathbf{w}_B$ is the global minimum. At any point $\mathbf{w}_C$, the local gradient of the error surface is given by the vector ∇E.

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

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))} \tag{5.25}$$

which satisfies $0 \leqslant y_k \leqslant 1$ 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.

Exercise 5.7 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 $\delta E \simeq \delta\mathbf{w}^{\mathrm{T}} \nabla E(\mathbf{w})$, 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

point in weight space such that the gradient of the error function vanishes, so that

$$\nabla E(\mathbf{w}) = 0 \tag{5.26}$$

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.

Section 5.1.1

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

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{5.27}$$

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 $\widehat{\mathbf{w}}$ in weight space

$$E(\mathbf{w}) \simeq E(\widehat{\mathbf{w}}) + (\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{b} + \frac{1}{2}(\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}}) \tag{5.28}$$

where cubic and higher terms have been omitted. Here $\mathbf{b}$ is defined to be the gradient of E evaluated at $\widehat{\mathbf{w}}$

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w}=\widehat{\mathbf{w}}} \tag{5.29}$$

and the Hessian matrix $\mathbf{H} = \nabla\nabla E$ has elements

$$(\mathbf{H})_{ij} \equiv \left.\frac{\partial E}{\partial w_i \partial w_j}\right|_{\mathbf{w}=\widehat{\mathbf{w}}}. \tag{5.30}$$

From (5.28), the corresponding local approximation to the gradient is given by

$$\nabla E \simeq \mathbf{b} + \mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}}). \tag{5.31}$$

For points $\mathbf{w}$ that are sufficiently close to $\widehat{\mathbf{w}}$, these expressions will give reasonable approximations for the error and its gradient.

Consider the particular case of a local quadratic approximation around a point $\mathbf{w}^\star$ that is a minimum of the error function. In this case there is no linear term, because $\nabla E = 0$ at $\mathbf{w}^\star$, and (5.28) becomes

$$E(\mathbf{w}) = E(\mathbf{w}^\star) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^\star)^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \mathbf{w}^\star) \tag{5.32}$$

where the Hessian $\mathbf{H}$ is evaluated at $\mathbf{w}^\star$. In order to interpret this geometrically, consider the eigenvalue equation for the Hessian matrix

$$\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{5.33}$$

where the eigenvectors $\mathbf{u}_i$ form a complete orthonormal set (Appendix C) so that

$$\mathbf{u}_i^{\mathrm{T}}\mathbf{u}_j = \delta_{ij}. \tag{5.34}$$

We now expand $(\mathbf{w} - \mathbf{w}^\star)$ as a linear combination of the eigenvectors in the form

$$\mathbf{w} - \mathbf{w}^\star = \sum_i \alpha_i \mathbf{u}_i. \tag{5.35}$$

This can be regarded as a transformation of the coordinate system in which the origin is translated to the point $\mathbf{w}^\star$, 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

$$E(\mathbf{w}) = E(\mathbf{w}^\star) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2. \tag{5.36}$$

A matrix $\mathbf{H}$ is said to be *positive definite* if, and only if,

$$\mathbf{v}^{\mathrm{T}}\mathbf{H}\mathbf{v} > 0 \qquad \text{for all } \mathbf{v}. \tag{5.37}$$

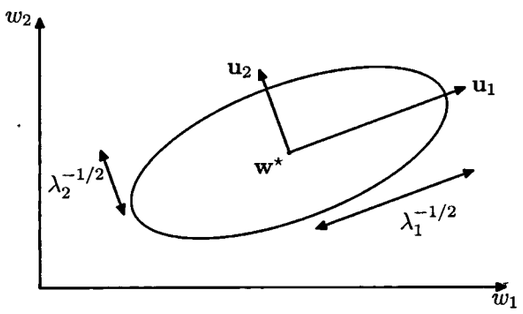


Figure 5.6 In the neighbourhood of a minimum $\mathbf{w}^\star$, the error function can be approximated by a quadratic. Contours of constant error are then ellipses whose axes are aligned with the eigenvectors $\mathbf{u}_i$ of the Hessian matrix, with lengths that are inversely proportional to the square roots of the corresponding eigenvectors λ_i.

Because the eigenvectors $\{\mathbf{u}_i\}$ form a complete set, an arbitrary vector $\mathbf{v}$ can be written in the form

$$\mathbf{v} = \sum_i c_i \mathbf{u}_i. \tag{5.38}$$

From (5.33) and (5.34), we then have

$$\mathbf{v}^{\mathrm{T}}\mathbf{H}\mathbf{v} = \sum_i c_i^2 \lambda_i \tag{5.39}$$

Exercise 5.10 and so $\mathbf{H}$ 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 *Exercise 5.11* $\{\mathbf{u}_i\}$, 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 $w^\star$ will be a minimum if

$$\left.\frac{\partial^2 E}{\partial w^2}\right|_{w^\star} > 0. \tag{5.40}$$

The corresponding result in D-dimensions is that the Hessian matrix, evaluated at *Exercise 5.12* $\mathbf{w}^\star$, 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 $\mathbf{b}$ and $\mathbf{H}$, which contain a total of $W(W+1)/2$ independent elements (because the matrix $\mathbf{H}$ is symmetric), where W is the *Exercise 5.13* dimensionality of $\mathbf{w}$ (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

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

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{5.41}$$

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

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}). \tag{5.42}$$

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

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}). \tag{5.43}$$

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 that 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

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}). \tag{5.44}$$

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

$$y_k = \sum_i w_{ki} x_i \tag{5.45}$$

together with an error function that, for a particular input pattern n, takes the form

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \tag{5.46}$$

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

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{5.47}$$

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

$$a_j = \sum_i w_{ji} z_i \tag{5.48}$$

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

$$z_j = h(a_j). \tag{5.49}$$

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

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}. \tag{5.50}$$

We now introduce a useful notation

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \tag{5.51}$$

where the δ's are often referred to as *errors* for reasons we shall see shortly. Using (5.48), we can write

$$\frac{\partial a_j}{\partial w_{ji}} = z_i. \tag{5.52}$$

Substituting (5.51) and (5.52) into (5.50), we then obtain

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i. \tag{5.53}$$

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

$$\delta_k = y_k - t_k \tag{5.54}$$

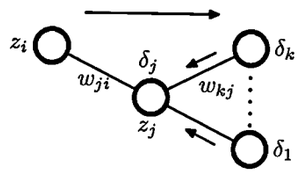


Figure 5.7 Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.

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,

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{5.55}$$

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

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \tag{5.56}$$

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:

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}}. \tag{5.57}$$

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

$$h(a) \equiv \tanh(a) \tag{5.58}$$

where

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}. \tag{5.59}$$

A useful feature of this function is that its derivative can be expressed in a particularly simple form:

$$h'(a) = 1 - h(a)^2. \tag{5.60}$$

We also consider a standard sum-of-squares error function, so that for pattern n the error is given by

$$E_n = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2 \tag{5.61}$$

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

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{5.62}$$

$$z_j = \tanh(a_j) \tag{5.63}$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j. \tag{5.64}$$

Next we compute the δ's for each output unit using

$$\delta_k = y_k - t_k. \tag{5.65}$$

Then we backpropagate these to obtain δs for the hidden units using

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k. \tag{5.66}$$

Finally, the derivatives with respect to the first-layer and second-layer weights are given by

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j. \tag{5.67}$$

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

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji})}{\epsilon} + O(\epsilon) \tag{5.68}$$

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

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji} - \epsilon)}{2\epsilon} + O(\epsilon^2). \tag{5.69}$$

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

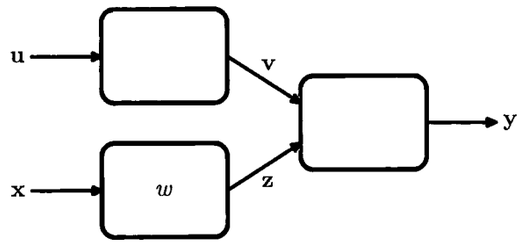


Figure 5.8 Illustration of a modular pattern recognition system in which the Jacobian matrix can be used to backpropagate error signals from the outputs through to earlier modules in the system.

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

$$J_{ki} \equiv \frac{\partial y_k}{\partial x_i} \tag{5.70}$$

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

$$\frac{\partial E}{\partial w} = \sum_{k,j} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_j} \frac{\partial z_j}{\partial w} \tag{5.71}$$

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

$$\Delta y_k \simeq \sum_i \frac{\partial y_k}{\partial x_i} \Delta x_i \tag{5.72}$$

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

$$\begin{aligned} J_{ki} = \frac{\partial y_k}{\partial x_i} &= \sum_j \frac{\partial y_k}{\partial a_j} \frac{\partial a_j}{\partial x_i} \\ &= \sum_j w_{ji} \frac{\partial y_k}{\partial a_j} \end{aligned} \tag{5.73}$$

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 $\partial y_k / \partial a_j$

$$\begin{aligned} \frac{\partial y_k}{\partial a_j} &= \sum_l \frac{\partial y_k}{\partial a_l} \frac{\partial a_l}{\partial a_j} \\ &= h'(a_j) \sum_l w_{lj} \frac{\partial y_k}{\partial a_l} \end{aligned} \tag{5.74}$$

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

$$\frac{\partial y_k}{\partial a_j} = \delta_{kj} \sigma'(a_j) \tag{5.75}$$

whereas for softmax outputs we have

$$\frac{\partial y_k}{\partial a_j} = \delta_{kj} y_k - y_k y_j. \tag{5.76}$$

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.

Exercise 5.15

Again, the implementation of such algorithms can be checked by using numerical differentiation in the form

$$\frac{\partial y_k}{\partial x_i} = \frac{y_k(x_i + \epsilon) - y_k(x_i - \epsilon)}{2\epsilon} + O(\epsilon^2) \tag{5.77}$$

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

$$\frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}}. \tag{5.78}$$

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 $i, j \in \{1, \ldots, W\}$ 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.

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

$$\frac{\partial^2 E_n}{\partial w_{ji}^2} = \frac{\partial^2 E_n}{\partial a_j^2} z_i^2. \tag{5.79}$$

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

$$\frac{\partial^2 E_n}{\partial a_j^2} = h'(a_j)^2 \sum_k \sum_{k'} w_{kj} w_{k'j} \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} + h''(a_j) \sum_k w_{kj} \frac{\partial E^n}{\partial a_k}. \tag{5.80}$$

If we now neglect off-diagonal elements in the second-derivative terms, we obtain (Becker and Le Cun, 1989; Le Cun *et al.*, 1990)

$$\frac{\partial^2 E_n}{\partial a_j^2} = h'(a_j)^2 \sum_k w_{kj}^2 \frac{\partial^2 E_n}{\partial a_k^2} + h''(a_j) \sum_k w_{kj} \frac{\partial E_n}{\partial a_k}. \tag{5.81}$$

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 be 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

$$E = \frac{1}{2} \sum_{n=1}^{N} (y_n - t_n)^2 \tag{5.82}$$

Exercise 5.16 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

$$\mathbf{H} = \nabla\nabla E = \sum_{n=1}^{N} \nabla y_n \nabla y_n + \sum_{n=1}^{N} (y_n - t_n) \nabla\nabla y_n. \tag{5.83}$$

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.

Exercise 5.17

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

$$\mathbf{H} \simeq \sum_{n=1}^{N} \mathbf{b}_n \mathbf{b}_n^{\mathrm{T}} \tag{5.84}$$

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

Exercise 5.19 output-unit activation functions, the corresponding approximation is given by

$$\mathbf{H} \simeq \sum_{n=1}^{N} y_n (1 - y_n) \mathbf{b}_n \mathbf{b}_n^{\mathrm{T}}. \tag{5.85}$$

An analogous result can be obtained for multiclass networks having softmax output-

Exercise 5.20 unit activation functions.

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

$$\mathbf{H}_N = \sum_{n=1}^{N} \mathbf{b}_n \mathbf{b}_n^{\mathrm{T}} \tag{5.86}$$

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

$$\mathbf{H}_{L+1} = \mathbf{H}_L + \mathbf{b}_{L+1}\mathbf{b}_{L+1}^{\mathrm{T}}. \tag{5.87}$$

In order to evaluate the inverse of the Hessian, we now consider the matrix identity

$$\left(\mathbf{M} + \mathbf{v}\mathbf{v}^{\mathrm{T}}\right)^{-1} = \mathbf{M}^{-1} - \frac{\left(\mathbf{M}^{-1}\mathbf{v}\right)\left(\mathbf{v}^{\mathrm{T}}\mathbf{M}^{-1}\right)}{1 + \mathbf{v}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{v}} \tag{5.88}$$

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

$$\mathbf{H}_{L+1}^{-1} = \mathbf{H}_L^{-1} - \frac{\mathbf{H}_L^{-1}\mathbf{b}_{L+1}\mathbf{b}_{L+1}^{\mathrm{T}}\mathbf{H}_L^{-1}}{1 + \mathbf{b}_{L+1}^{\mathrm{T}}\mathbf{H}_L^{-1}\mathbf{b}_{L+1}}. \tag{5.89}$$

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.

Exercise 5.21

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

$$\frac{\partial^2 E}{\partial w_{ji}\partial w_{lk}} = \frac{1}{4\epsilon^2}\{E(w_{ji}+\epsilon, w_{lk}+\epsilon) - E(w_{ji}+\epsilon, w_{lk}-\epsilon) - E(w_{ji}-\epsilon, w_{lk}+\epsilon) + E(w_{ji}-\epsilon, w_{lk}-\epsilon)\} + O(\epsilon^2). \tag{5.90}$$

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

$$\frac{\partial^2 E}{\partial w_{ji}\partial w_{lk}} = \frac{1}{2\epsilon}\left\{\frac{\partial E}{\partial w_{ji}}(w_{lk}+\epsilon) - \frac{\partial E}{\partial w_{ji}}(w_{lk}-\epsilon)\right\} + O(\epsilon^2). \tag{5.91}$$

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 backpropagation 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).

Exercise 5.22

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 denoted hidden units, and indices k and k' to denote outputs. We first define

$$\delta_k = \frac{\partial E_n}{\partial a_k}, \qquad M_{kk'} \equiv \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} \tag{5.92}$$

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:

$$\frac{\partial^2 E_n}{\partial w_{kj}^{(2)}\partial w_{k'j'}^{(2)}} = z_j z_{j'} M_{kk'}. \tag{5.93}$$

2. Both weights in the first layer:

$$\frac{\partial^2 E_n}{\partial w_{ji}^{(1)} \partial w_{j'i'}^{(1)}} = x_i x_{i'} h''(a_{j'}) I_{jj'} \sum_k w_{kj'}^{(2)} \delta_k + x_i x_{i'} h'(a_{j'}) h'(a_j) \sum_k \sum_{k'} w_{k'j'}^{(2)} w_{kj}^{(2)} M_{kk'}. \tag{5.94}$$

3. One weight in each layer:

$$\frac{\partial^2 E_n}{\partial w_{ji}^{(1)} \partial w_{kj'}^{(2)}} = x_i h'(a_{j'}) \left\{ \delta_k I_{jj'} + z_j \sum_{k'} w_{k'j'}^{(2)} H_{kk'} \right\}. \tag{5.95}$$

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 $\mathbf{v}^{\mathrm{T}}\mathbf{H}$ 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 $\mathbf{v}^{\mathrm{T}}\mathbf{H}$ directly in a way that requires only $O(W)$ operations.

To do this, we first note that

$$\mathbf{v}^{\mathrm{T}}\mathbf{H} = \mathbf{v}^{\mathrm{T}} \nabla (\nabla E) \tag{5.96}$$

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 $\mathbf{v}^{\mathrm{T}}\mathbf{H}$ (Møller, 1993; Pearlmutter, 1994). This corresponds to acting on the original forward-propagation and backpropagation equations with a differential operator $\mathbf{v}^{\mathrm{T}}\nabla$. Pearlmutter (1994) used the notation $\mathcal{R}\{\cdot\}$ to denote the operator $\mathbf{v}^{\mathrm{T}}\nabla$, and we shall follow this convention. The analysis is straightforward and makes use of the usual rules of differential calculus, together with the result

$$\mathcal{R}\{\mathbf{w}\} = \mathbf{v}. \tag{5.97}$$

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

$$a_j = \sum_i w_{ji} x_i \tag{5.98}$$

$$z_j = h(a_j) \tag{5.99}$$

$$y_k = \sum_j w_{kj} z_j. \tag{5.100}$$

We now act on these equations using the $\mathcal{R}\{\cdot\}$ operator to obtain a set of forward propagation equations in the form

$$\mathcal{R}\{a_j\} = \sum_i v_{ji} x_i \tag{5.101}$$

$$\mathcal{R}\{z_j\} = h'(a_j) \mathcal{R}\{a_j\} \tag{5.102}$$

$$\mathcal{R}\{y_k\} = \sum_j w_{kj} \mathcal{R}\{z_j\} + \sum_j v_{kj} z_j \tag{5.103}$$

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:

$$\delta_k = y_k - t_k \tag{5.104}$$

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k. \tag{5.105}$$

Again, we act on these equations with the $\mathcal{R}\{\cdot\}$ operator to obtain a set of backpropagation equations in the form

$$\mathcal{R}\{\delta_k\} = \mathcal{R}\{y_k\} \tag{5.106}$$

$$\mathcal{R}\{\delta_j\} = h''(a_j) \mathcal{R}\{a_j\} \sum_k w_{kj} \delta_k + h'(a_j) \sum_k v_{kj} \delta_k + h'(a_j) \sum_k w_{kj} \mathcal{R}\{\delta_k\}. \tag{5.107}$$

Finally, we have the usual equations for the first derivatives of the error

$$\frac{\partial E}{\partial w_{kj}} = \delta_k z_j \tag{5.108}$$

$$\frac{\partial E}{\partial w_{ji}} = \delta_j x_i \tag{5.109}$$

and acting on these with the $\mathcal{R}\{\cdot\}$ operator, we obtain expressions for the elements of the vector $\mathbf{v}^{\mathrm{T}}\mathbf{H}$

$$\mathcal{R}\left\{\frac{\partial E}{\partial w_{kj}}\right\} = \mathcal{R}\{\delta_k\}z_j + \delta_k\mathcal{R}\{z_j\} \tag{5.110}$$

$$\mathcal{R}\left\{\frac{\partial E}{\partial w_{ji}}\right\} = x_i\mathcal{R}\{\delta_j\}. \tag{5.111}$$

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 $\mathbf{v}^{\mathrm{T}}\mathbf{H}$ are then given by (5.110) and (5.111). An elegant aspect of this technique is that the equations for evaluating $\mathbf{v}^{\mathrm{T}}\mathbf{H}$ 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 $(0, 0, \ldots, 1, \ldots, 0)$ 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

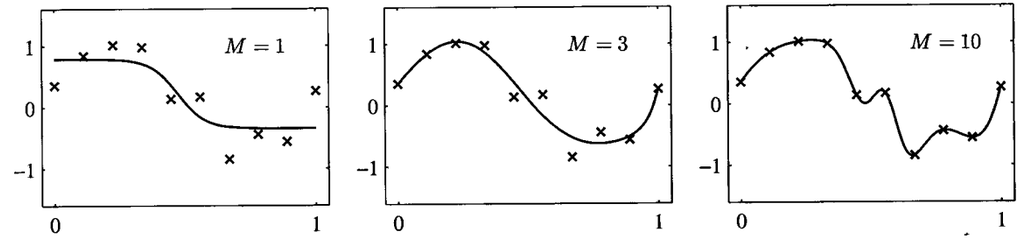


Figure 5.9 Examples of two-layer networks trained on 10 data points drawn from the sinusoidal data set. The graphs show the result of fitting networks having $M = 1$, 3 and 10 hidden units, respectively, by minimizing a sum-of-squares error function using a scaled conjugate-gradient algorithm.

of the form

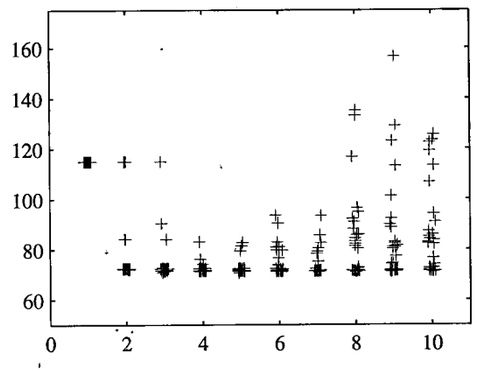
$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}. \tag{5.112}$$

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

$$z_j = h\left(\sum_i w_{ji}x_i + w_{j0}\right) \tag{5.113}$$

while the activations of the output units are given by

$$y_k = \sum_j w_{kj}z_j + w_{k0}. \tag{5.114}$$

Suppose we perform a linear transformation of the input data of the form

$$x_i \to \widetilde{x}_i = ax_i + b. \tag{5.115}$$

Exercise 5.24 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

$$w_{ji} \to \widetilde{w}_{ji} = \frac{1}{a}w_{ji} \tag{5.116}$$

$$w_{j0} \to \widetilde{w}_{j0} = w_{j0} - \frac{b}{a}\sum_i w_{ji}. \tag{5.117}$$

Similarly, a linear transformation of the output variables of the network of the form

$$y_k \to \widetilde{y}_k = cy_k + d \tag{5.118}$$

can be achieved by making a transformation of the second-layer weights and biases using

$$w_{kj} \to \widetilde{w}_{kj} = cw_{kj} \tag{5.119}$$

$$w_{k0} \to \widetilde{w}_{k0} = cw_{k0} + d. \tag{5.120}$$

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

$$\frac{\lambda_1}{2}\sum_{w\in\mathcal{W}_1} w^2 + \frac{\lambda_2}{2}\sum_{w\in\mathcal{W}_2} w^2 \tag{5.121}$$

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

will remain unchanged under the weight transformations provided the regularization parameters are re-scaled using $\lambda_1 \to a^{1/2}\lambda_1$ and $\lambda_2 \to c^{-1/2}\lambda_2$.

The regularizer (5.121) corresponds to a prior of the form

$$p(\mathbf{w}|\alpha_1,\alpha_2) \propto \exp\left(-\frac{\alpha_1}{2}\sum_{w\in\mathcal{W}_1} w^2 - \frac{\alpha_2}{2}\sum_{w\in\mathcal{W}_2} w^2\right). \tag{5.122}$$

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

$$p(\mathbf{w}) \propto \exp\left(-\frac{1}{2}\sum_k \alpha_k\|\mathbf{w}\|_k^2\right) \tag{5.123}$$

where

$$\|\mathbf{w}\|_k^2 = \sum_{j\in\mathcal{W}_k} w_j^2. \tag{5.124}$$

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 to grows during the training process, corresponding to a steady increase in the effective complexity of the model. Halting training before

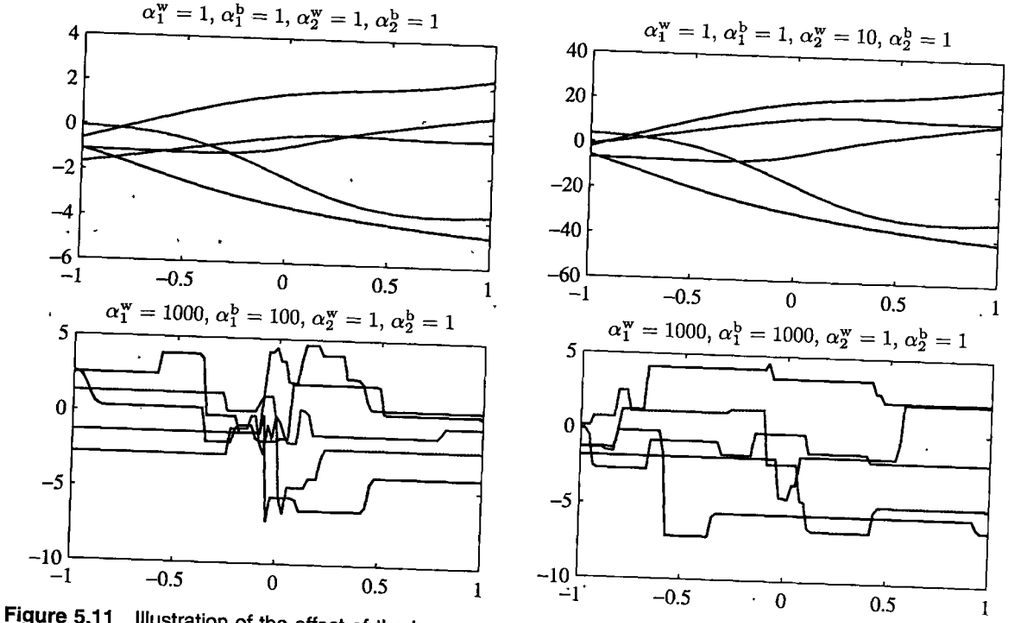


Figure 5.11 Illustration of the effect of the hyperparameters governing the prior distribution over weights and biases in a two-layer network having a single input, a single linear output, and 12 hidden units having 'tanh' activation functions. The priors are governed by four hyperparameters α_1^b, α_1^w, 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 $\widetilde{\mathbf{w}}$ and then move towards the minimum of the error function $\mathbf{w}_{\mathrm{ML}}$. This follows from the shape of the error surface and the widely differing eigenvalues of the Hessian. Stopping at a point near $\widetilde{\mathbf{w}}$ 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

Exercise 5.25

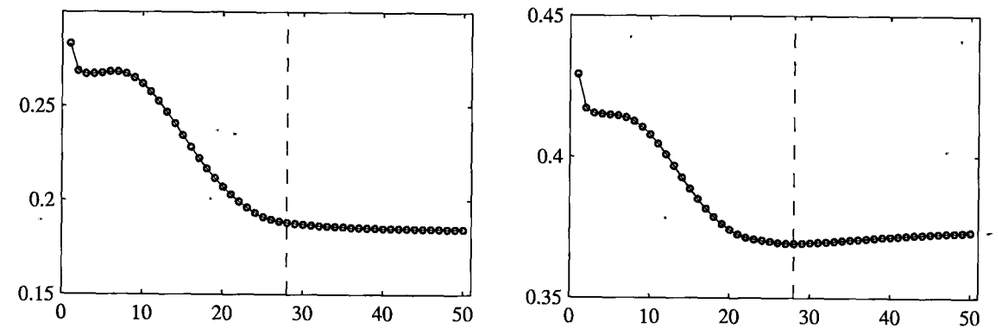


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped 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

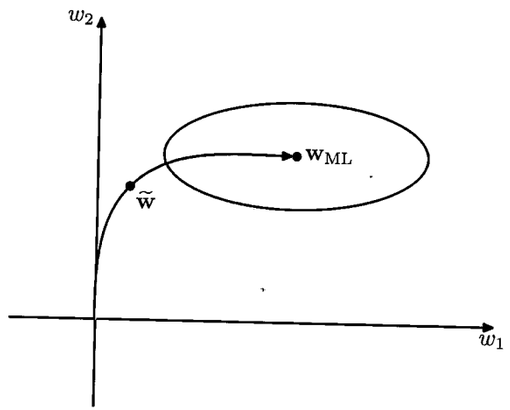


Figure 5.13 A schematic illustration of why early stopping can give similar results to weight decay in the case of a quadratic error function. The ellipse shows a contour of constant error, and $\mathbf{w}_{\mathrm{ML}}$ denotes the minimum of the error function. If the weight vector starts at the origin 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 $\widetilde{\mathbf{w}}$ 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.

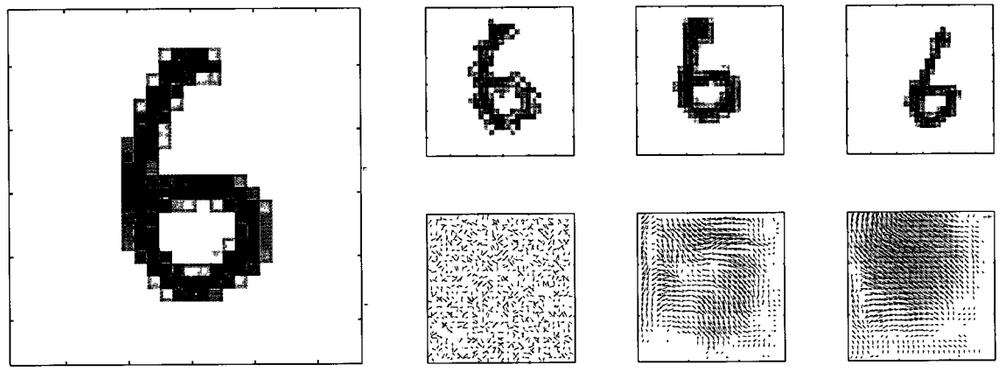


Figure 5.14 Illustration of the synthetic warping of a handwritten digit. The original image is shown on the left. On the right, the top row shows three examples of warped digits, with the corresponding displacement fields shown on the bottom row. These displacement fields are generated by sampling random 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 $\mathbf{x}_n$. 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 $\mathbf{x}_n$

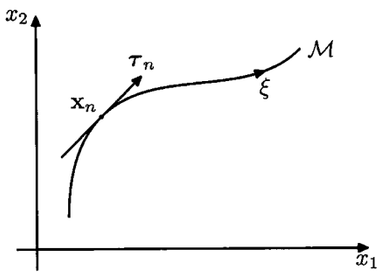


Figure 5.15 Illustration of a two-dimensional input space showing the effect of a continuous transformation on a particular input vector $\mathbf{x}_n$. A one-dimensional transformation, parameterized by the continuous variable ξ, applied to $\mathbf{x}_n$ causes it to sweep out a one-dimensional manifold $\mathcal{M}$. Locally, the effect of the transformation can be approximated by the tangent vector $\boldsymbol{\tau}_n$.

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

$$\boldsymbol{\tau}_n = \left. \frac{\partial \mathbf{s}(\mathbf{x}_n, \xi)}{\partial \xi} \right|_{\xi=0}. \tag{5.125}$$

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

$$\left. \frac{\partial y_k}{\partial \xi} \right|_{\xi=0} = \sum_{i=1}^{D} \frac{\partial y_k}{\partial x_i} \left. \frac{\partial x_i}{\partial \xi} \right|_{\xi=0} = \sum_{i=1}^{D} J_{ki} \tau_i \tag{5.126}$$

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

$$\widetilde{E} = E + \lambda \Omega \tag{5.127}$$

where λ is a regularization coefficient and

$$\Omega = \frac{1}{2} \sum_n \sum_k \left(\left. \frac{\partial y_{nk}}{\partial \xi} \right|_{\xi=0} \right)^2 = \frac{1}{2} \sum_n \sum_k \left(\sum_{i=1}^{D} J_{nki} \tau_{ni} \right)^2. \tag{5.128}$$

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.

Exercise 5.26

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).

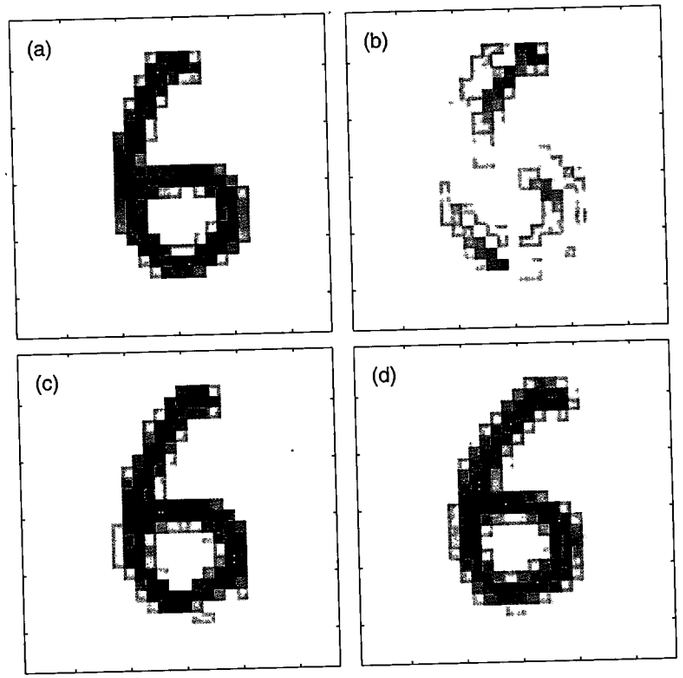


Figure 5.16 Illustration showing (a) the original image $\mathbf{x}$ of a handwritten digit, (b) the tangent vector $\boldsymbol{\tau}$ corresponding to an infinitesimal clockwise rotation, (c) the result of adding a small contribution from the tangent vector to the original image giving $\mathbf{x} + \epsilon\boldsymbol{\tau}$ 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 $\mathbf{s}(\mathbf{x}, \xi)$, with $\mathbf{s}(\mathbf{x}, 0) = \mathbf{x}$. 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

$$E = \frac{1}{2} \iint \{y(\mathbf{x}) - t\}^2 p(t|\mathbf{x}) p(\mathbf{x}) \, \mathrm{d}\mathbf{x} \, \mathrm{d}t \tag{5.129}$$

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

$$\widetilde{E} = \frac{1}{2}\iiint \{y(\mathbf{s}(\mathbf{x},\xi)) - t\}^2 p(t|\mathbf{x})p(\mathbf{x})p(\xi)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t\,\mathrm{d}\xi. \tag{5.130}$$

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

$$\begin{aligned}\mathbf{s}(\mathbf{x},\xi) &= \mathbf{s}(\mathbf{x},0) + \xi\,\frac{\partial}{\partial\xi}\mathbf{s}(\mathbf{x},\xi)\bigg|_{\xi=0} + \frac{\xi^2}{2}\,\frac{\partial^2}{\partial\xi^2}\mathbf{s}(\mathbf{x},\xi)\bigg|_{\xi=0} + O(\xi^3)\\ &= \mathbf{x} + \xi\boldsymbol{\tau} + \frac{1}{2}\xi^2\boldsymbol{\tau}' + O(\xi^3)\end{aligned}$$

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

$$y(\mathbf{s}(\mathbf{x},\xi)) = y(\mathbf{x}) + \xi\boldsymbol{\tau}^{\mathrm{T}}\nabla y(\mathbf{x}) + \frac{\xi^2}{2}\left[(\boldsymbol{\tau}')^{\mathrm{T}}\nabla y(\mathbf{x}) + \boldsymbol{\tau}^{\mathrm{T}}\nabla\nabla y(\mathbf{x})\boldsymbol{\tau}\right] + O(\xi^3).$$

Substituting into the mean error function (5.130) and expanding, we then have

$$\begin{aligned}\widetilde{E} &= \frac{1}{2}\iint \{y(\mathbf{x}) - t\}^2 p(t|\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t\\ &+ \mathbb{E}[\xi]\iint \{y(\mathbf{x}) - t\}\boldsymbol{\tau}^{\mathrm{T}}\nabla y(\mathbf{x})p(t|\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t\\ &+ \mathbb{E}[\xi^2]\iint \left[\{y(\mathbf{x}) - t\}\frac{1}{2}\left\{(\boldsymbol{\tau}')^{\mathrm{T}}\nabla y(\mathbf{x}) + \boldsymbol{\tau}^{\mathrm{T}}\nabla\nabla y(\mathbf{x})\boldsymbol{\tau}\right\}\right.\\ &\left. + \left(\boldsymbol{\tau}^{\mathrm{T}}\nabla y(\mathbf{x})\right)^2\right] p(t|\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}\,\mathrm{d}t + O(\xi^3).\end{aligned}$$

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

$$\widetilde{E} = E + \lambda\Omega \tag{5.131}$$

where E is the original sum-of-squares error, and the regularization term Ω takes the form

$$\begin{aligned}\Omega &= \int \left[\{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}\frac{1}{2}\left\{(\boldsymbol{\tau}')^{\mathrm{T}}\nabla y(\mathbf{x}) + \boldsymbol{\tau}^{\mathrm{T}}\nabla\nabla y(\mathbf{x})\boldsymbol{\tau}\right\}\right.\\ &\left. + \left(\boldsymbol{\tau}^{T}\nabla y(\mathbf{x})\right)^2\right] p(\mathbf{x})\,\mathrm{d}\mathbf{x}\end{aligned} \tag{5.132}$$

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

$$y(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] + O(\xi). \tag{5.133}$$

Thus, to leading order in ξ, the first term in the regularizer vanishes and we are left with

$$\Omega = \frac{1}{2}\int \left(\boldsymbol{\tau}^T\nabla y(\mathbf{x})\right)^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x} \tag{5.134}$$

which is equivalent to the tangent propagation regularizer (5.128).

Exercise 5.27

If we consider the special case in which the transformation of the inputs simply consists of the addition of random noise, so that $\mathbf{x} \to \mathbf{x} + \boldsymbol{\xi}$, then the regularizer takes the form

$$\Omega = \frac{1}{2}\int \|\nabla y(\mathbf{x})\|^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x} \tag{5.135}$$

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 transformation 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

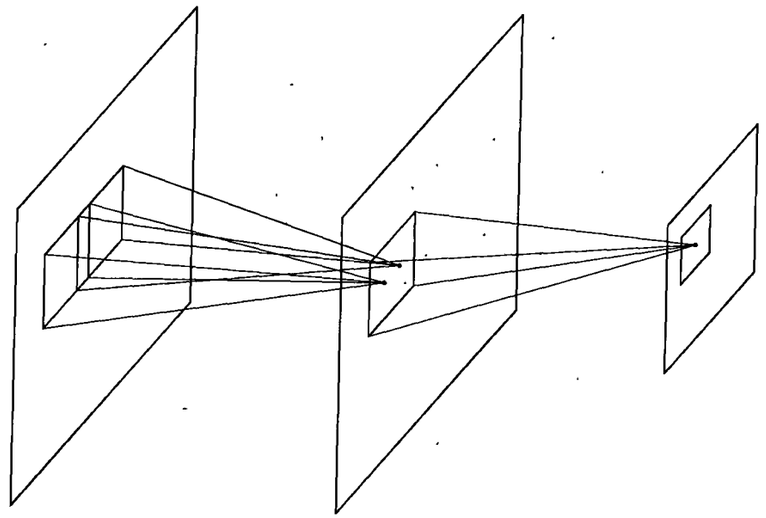


Figure 5.17 Diagram illustrating part of a convolutional neural network, showing a layer of convolutional units followed by a layer of subsampling units. Several successive pairs of such layers may be used.

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.

Exercise 5.28

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.

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

Section 2.3.9

$$p(\mathbf{w}) = \prod_i p(w_i) \tag{5.136}$$

where

$$p(w_i) = \sum_{j=1}^{M} \pi_j \mathcal{N}(w_i|\mu_j, \sigma_j^2) \tag{5.137}$$

and π_j are the mixing coefficients. Taking the negative logarithm then leads to a regularization function of the form

$$\Omega(\mathbf{w}) = -\sum_i \ln \left(\sum_{j=1}^{M} \pi_j \mathcal{N}(w_i|\mu_j, \sigma_j^2) \right). \tag{5.138}$$

The total error function is then given by

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda \Omega(\mathbf{w}) \tag{5.139}$$

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

$$\gamma_j(w) = \frac{\pi_j \mathcal{N}(w|\mu_j, \sigma_j^2)}{\sum_k \pi_k \mathcal{N}(w|\mu_k, \sigma_k^2)}. \tag{5.140}$$

The derivatives of the total error function with respect to the weights are then given by

Exercise 5.29

$$\frac{\partial \widetilde{E}}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda \sum_j \gamma_j(w_i) \frac{(w_i - \mu_j)}{\sigma_j^2}. \tag{5.141}$$

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

Exercise 5.30

$$\frac{\partial \widetilde{E}}{\partial \mu_j} = \lambda \sum_i \gamma_j(w_i) \frac{(\mu_i - w_j)}{\sigma_j^2} \tag{5.142}$$

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

Exercise 5.31

$$\frac{\partial \widetilde{E}}{\partial \sigma_j} = \lambda \sum_i \gamma_j(w_i) \left(\frac{1}{\sigma_j} - \frac{(w_i - \mu_j)^2}{\sigma_j^3} \right) \tag{5.143}$$

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

$$\sigma_j^2 = \exp(\eta_j) \tag{5.144}$$

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

$$\sum_j \pi_j = 1, \qquad 0 \leqslant \pi_i \leqslant 1 \tag{5.145}$$

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

$$\pi_j = \frac{\exp(\eta_j)}{\sum_{k=1}^{M} \exp(\eta_k)}. \tag{5.146}$$

The derivatives of the regularized error function with respect to the $\{\eta_j\}$ then take the form

Exercise 5.32

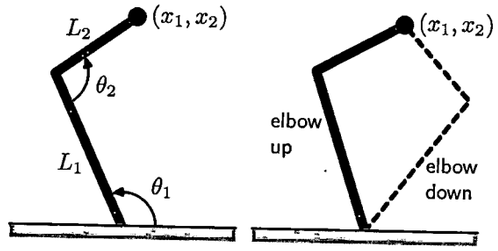


Figure 5.18 The left figure shows a two-link robot arm, in which the Cartesian coordinates (x_1, x_2) of the end effector are determined uniquely by the two joint angles θ_1 and θ_2 and the (fixed) lengths L_1 and L_2 of the arms. This is know as the *forward kinematics* of the arm. In practice, we have 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'.

$$\frac{\partial \widetilde{E}}{\partial \eta_j} = \sum_i \{\pi_j - \gamma_j(w_i)\}\,. \tag{5.147}$$

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.

Exercise 5.33

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 corresponds 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

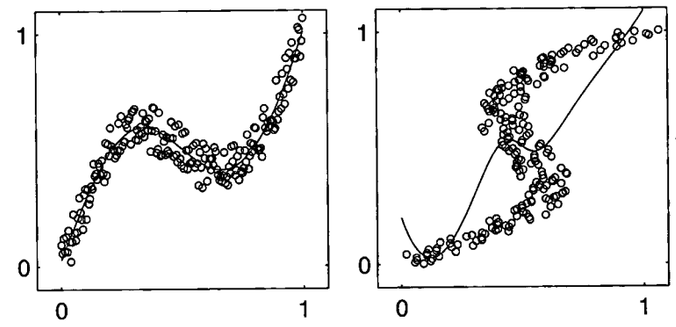


Figure 5.19 On the left is the data set for a simple 'forward problem' in which the red curve shows the result of fitting a two-layer neural network by minimizing the sum-of-squares error function. The corresponding inverse problem, shown on the right, is obtained by exchanging the roles of x and t. 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 $p(\mathbf{t}|\mathbf{x})$ in which both the mixing coefficients as well as the component densities are flexible functions of the input vector $\mathbf{x}$, giving rise to the *mixture density network*. For any given value of $\mathbf{x}$, the mixture model provides a general formalism for modelling an arbitrary conditional density function $p(\mathbf{t}|\mathbf{x})$. 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

$$p(\mathbf{t}|\mathbf{x}) = \sum_{k=1}^{K} \pi_k(\mathbf{x}) \mathcal{N}\left(\mathbf{t}|\boldsymbol{\mu}_k(\mathbf{x}), \sigma_k^2(\mathbf{x})\right). \tag{5.148}$$

This is an example of a *heteroscedastic* model since the noise variance on the data is a function of the input vector $\mathbf{x}$. 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 $p(\mathbf{t}|\mathbf{x})$ does not assume factorization with respect to the components of $\mathbf{t}$ (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 $\pi_k(\mathbf{x})$, the means $\boldsymbol{\mu}_k(\mathbf{x})$, and the variances $\sigma_k^2(\mathbf{x})$, to be governed by

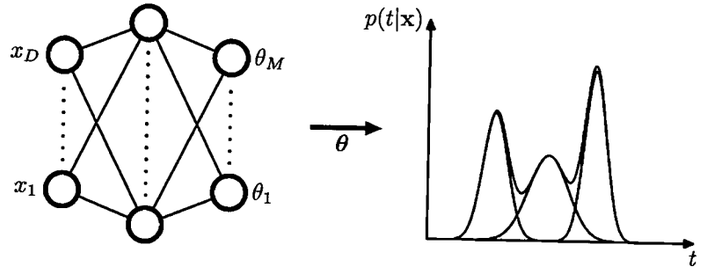


Figure 5.20 The *mixture density network* can represent general conditional probability densities $p(\mathbf{t}|\mathbf{x})$ by considering a parametric mixture model for the distribution of $\mathbf{t}$ whose parameters are determined by the outputs of a neural network that takes $\mathbf{x}$ as its 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 $\boldsymbol{\mu}_k(\mathbf{x})$. 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

$$\sum_{k=1}^{K} \pi_k(\mathbf{x}) = 1, \qquad 0 \leqslant \pi_k(\mathbf{x}) \leqslant 1 \tag{5.149}$$

which can be achieved using a set of softmax outputs

$$\pi_k(\mathbf{x}) = \frac{\exp(a_k^\pi)}{\sum_{l=1}^{K} \exp(a_l^\pi)}. \tag{5.150}$$

Similarly, the variances must satisfy $\sigma_k^2(\mathbf{x}) \geqslant 0$ and so can be represented in terms of the exponentials of the corresponding network activations using

$$\sigma_k(\mathbf{x}) = \exp(a_k^\sigma). \tag{5.151}$$

Finally, because the means $\boldsymbol{\mu}_k(\mathbf{x})$ have real components, they can be represented

directly by the network output activations

$$\mu_{kj}(\mathbf{x}) = a_{kj}^\mu. \tag{5.152}$$

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

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{k} \pi_k(\mathbf{x}_n, \mathbf{w}) \mathcal{N}\left(\mathbf{t}_n | \boldsymbol{\mu}_k(\mathbf{x}_n, \mathbf{w}), \sigma_k^2(\mathbf{x}_n, \mathbf{w})\right) \right\} \tag{5.153}$$

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 backpropagated 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

$$\gamma_k(\mathbf{t}|\mathbf{x}) = \frac{\pi_k \mathcal{N}_{nk}}{\sum_{l=1}^{K} \pi_l \mathcal{N}_{nl}} \tag{5.154}$$

where $\mathcal{N}_{nk}$ denotes $\mathcal{N}\left(\mathbf{t}_n | \boldsymbol{\mu}_k(\mathbf{x}_n), \sigma_k^2(\mathbf{x}_n)\right)$.

Exercise 5.34 The derivatives with respect to the network output activations governing the mixing coefficients are given by

$$\frac{\partial E_n}{\partial a_k^\pi} = \pi_k - \gamma_k. \tag{5.155}$$

Exercise 5.35 Similarly, the derivatives with respect to the output activations controlling the component means are given by

$$\frac{\partial E_n}{\partial a_{kl}^\mu} = \gamma_k \left\{ \frac{\mu_{kl} - t_l}{\sigma_k^2} \right\}. \tag{5.156}$$

Exercise 5.36 Finally, the derivatives with respect to the output activations controlling the component variances are given by

$$\frac{\partial E_n}{\partial a_k^\sigma} = -\gamma_k \left\{ \frac{\|\mathbf{t} - \boldsymbol{\mu}_k\|^2}{\sigma_k^3} - \frac{1}{\sigma_k} \right\}. \tag{5.157}$$

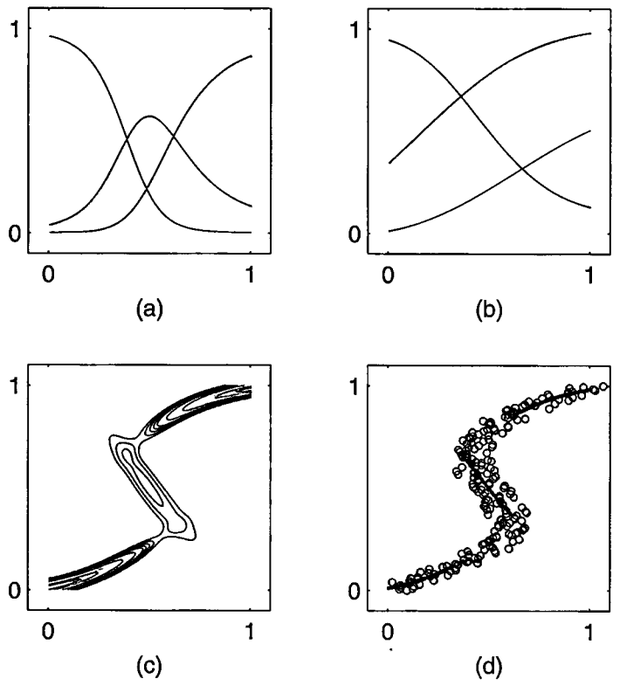


Figure 5.21 (a) Plot of the mixing coefficients $\pi_k(x)$ as a function of x for the three kernel functions in a mixture density network trained on the data shown in Figure 5.19. The model has three Gaussian components, and uses a two-layer multilayer perceptron with five 'tanh' sigmoidal units in the 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 $\pi_k(\mathbf{x})$.

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

$$\mathbb{E}[\mathbf{t}|\mathbf{x}] = \int \mathbf{t} p(\mathbf{t}|\mathbf{x})\, \mathrm{d}\mathbf{t} = \sum_{k=1}^{K} \pi_k(\mathbf{x}) \boldsymbol{\mu}_k(\mathbf{x}) \tag{5.158}$$

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.

Exercise 5.37

We can similarly evaluate the variance of the density function about the conditional average, to give

$$s^2(\mathbf{x}) = \mathbb{E}\left[\|\mathbf{t} - \mathbb{E}[\mathbf{t}|\mathbf{x}]\|^2 |\mathbf{x}\right] \tag{5.159}$$

$$= \sum_{k=1}^{K} \pi_k(\mathbf{x}) \left\{ \sigma_k^2(\mathbf{x}) + \left\| \boldsymbol{\mu}_k(\mathbf{x}) - \sum_{l=1}^{K} \pi_l(\mathbf{x}) \boldsymbol{\mu}_l(\mathbf{x}) \right\|^2 \right\} \tag{5.160}$$

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

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

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}). \tag{5.161}$$

Similarly, we shall choose a prior distribution over the weights $\mathbf{w}$ that is Gaussian of the form

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}). \tag{5.162}$$

For an i.i.d. data set of N observations $\mathbf{x}_1, \ldots, \mathbf{x}_N$, with a corresponding set of target values $\mathcal{D} = \{t_1, \ldots, t_N\}$, the likelihood function is given by

$$p(\mathcal{D}|\mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}) \tag{5.163}$$

and so the resulting posterior distribution is then

$$p(\mathbf{w}|\mathcal{D}, \alpha, \beta) \propto p(\mathbf{w}|\alpha)p(\mathcal{D}|\mathbf{w}, \beta). \tag{5.164}$$

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

$$\ln p(\mathbf{w}|\mathcal{D}) = -\frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} - \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \text{const} \tag{5.165}$$

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 $\mathbf{w}_{\mathrm{MAP}}$, by standard nonlinear optimization algorithms such as conjugate gradients, using error backpropagation to evaluate the required derivatives.

Having found a mode $\mathbf{w}_{\mathrm{MAP}}$, 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

$$\mathbf{A} = -\nabla\nabla \ln p(\mathbf{w}|\mathcal{D}, \alpha, \beta) = \alpha\mathbf{I} + \beta\mathbf{H} \tag{5.166}$$

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

$$q(\mathbf{w}|\mathcal{D}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}}, \mathbf{A}^{-1}). \tag{5.167}$$

Similarly, the predictive distribution is obtained by marginalizing with respect to this posterior distribution

$$p(t|\mathbf{x}, \mathcal{D}) = \int p(t|\mathbf{x}, \mathbf{w})q(\mathbf{w}|\mathcal{D})\,\mathrm{d}\mathbf{w}. \tag{5.168}$$

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 $\mathbf{w}_{\mathrm{MAP}}$ and retain only the linear terms

$$y(\mathbf{x}, \mathbf{w}) \simeq y(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}}) + \mathbf{g}^{\mathrm{T}}(\mathbf{w} - \mathbf{w}_{\mathrm{MAP}}) \tag{5.169}$$

where we have defined

$$\mathbf{g} = \nabla_{\mathbf{w}} y(\mathbf{x}, \mathbf{w})|_{\mathbf{w}=\mathbf{w}_{\mathrm{MAP}}}. \tag{5.170}$$

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

$$p(t|\mathbf{x}, \mathbf{w}, \beta) \simeq \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}}) + \mathbf{g}^{\mathrm{T}}(\mathbf{w} - \mathbf{w}_{\mathrm{MAP}}), \beta^{-1}\right). \tag{5.171}$$

Exercise 5.38

We can therefore make use of the general result (2.115) for the marginal $p(t)$ to give

$$p(t|\mathbf{x}, \mathcal{D}, \alpha, \beta) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}}), \sigma^2(\mathbf{x})\right) \tag{5.172}$$

where the input-dependent variance is given by

$$\sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{g}. \tag{5.173}$$

We see that the predictive distribution $p(t|\mathbf{x}, \mathcal{D})$ is a Gaussian whose mean is given by the network function $y(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}})$ 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

$$p(\mathcal{D}|\alpha, \beta) = \int p(\mathcal{D}|\mathbf{w}, \beta)p(\mathbf{w}|\alpha)\,\mathrm{d}\mathbf{w}. \tag{5.174}$$

Exercise 5.39

This is easily evaluated by making use of the Laplace approximation result (4.135). Taking logarithms then gives

$$\ln p(\mathcal{D}|\alpha, \beta) \simeq -E(\mathbf{w}_{\mathrm{MAP}}) - \frac{1}{2}\ln|\mathbf{A}| + \frac{W}{2}\ln\alpha + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) \tag{5.175}$$

where W is the total number of parameters in $\mathbf{w}$, and the regularized error function is defined by

$$E(\mathbf{w}_{\mathrm{MAP}}) = \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}_{\mathrm{MAP}}) - t_n\}^2 + \frac{\alpha}{2}\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}}. \tag{5.176}$$

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

$$\beta\mathbf{H}\mathbf{u}_i = \lambda_i\mathbf{u}_i \tag{5.177}$$

where $\mathbf{H}$ is the Hessian matrix comprising the second derivatives of the sum-of-squares error function, evaluated at $\mathbf{w} = \mathbf{w}_{\mathrm{MAP}}$. By analogy with (3.92), we obtain

$$\alpha = \frac{\gamma}{\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}}} \tag{5.178}$$

Section 3.5.3

where γ represents the effective number of parameters and is defined by

$$\gamma = \sum_{i=1}^{W}\frac{\lambda_i}{\alpha + \lambda_i}. \tag{5.179}$$

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

$$\frac{1}{\beta} = \frac{1}{N - \gamma}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}_{\mathrm{MAP}}) - t_n\}^2. \tag{5.180}$$

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 $\mathbf{w}_{\mathrm{MAP}}$ 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.

Section 5.1.1

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

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.

Exercise 5.40

The log likelihood function for this model is given by

$$\ln p(\mathcal{D}|\mathbf{w}) = \sum_{n} = 1^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \tag{5.181}$$

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

$$E(\mathbf{w}) = -\ln p(\mathcal{D}|\mathbf{w}) + \frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \tag{5.182}$$

and can be achieved using error backpropagation combined with standard optimization algorithms, as discussed in Section 5.3.

Having found a solution $\mathbf{w}_{\mathrm{MAP}}$ 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).

Exercise 5.41

To optimize the hyperparameter α, we again maximize the marginal likelihood, which is easily shown to take the form

$$\ln p(\mathcal{D}|\alpha) \simeq -E(\mathbf{w}_{\mathrm{MAP}}) - \frac{1}{2}\ln|\mathbf{A}| + \frac{W}{2}\ln\alpha + \text{const} \tag{5.183}$$

where the regularized error function is defined by

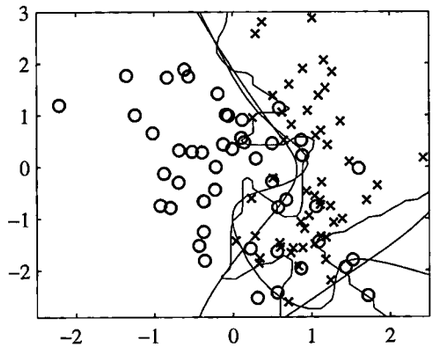
$$E(\mathbf{w}_{\mathrm{MAP}}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\} + \frac{\alpha}{2}\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}} \tag{5.184}$$

in which $y_n \equiv y(\mathbf{x}_n, \mathbf{w}_{\mathrm{MAP}})$. 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

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

$$p(t|\mathbf{x}, \mathcal{D}) \simeq p(t|\mathbf{x}, \mathbf{w}_{\mathrm{MAP}}). \tag{5.185}$$

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

$$a(\mathbf{x}, \mathbf{w}) \simeq a_{\mathrm{MAP}}(\mathbf{x}) + \mathbf{b}^{\mathrm{T}}(\mathbf{w} - \mathbf{w}_{\mathrm{MAP}}) \tag{5.186}$$

where $a_{\mathrm{MAP}}(\mathbf{x}) = a(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}})$, and the vector $\mathbf{b} \equiv \nabla a(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}})$ 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

$$p(a|\mathbf{x}, \mathcal{D}) = \int \delta\left(a - a_{\mathrm{MAP}}(\mathbf{x}) - \mathbf{b}^{\mathrm{T}}(\mathbf{x})(\mathbf{w} - \mathbf{w}_{\mathrm{MAP}})\right) q(\mathbf{w}|\mathcal{D})\,\mathrm{d}\mathbf{w} \tag{5.187}$$

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 $a_{\mathrm{MAP}} \equiv a(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}})$, and variance

$$\sigma_a^2(\mathbf{x}) = \mathbf{b}^{\mathrm{T}}(\mathbf{x})\mathbf{A}^{-1}\mathbf{b}(\mathbf{x}). \tag{5.188}$$

Finally, to obtain the predictive distribution, we must marginalize over a using

$$p(t = 1|\mathbf{x}, \mathcal{D}) = \int \sigma(a) p(a|\mathbf{x}, \mathcal{D})\,\mathrm{d}a. \tag{5.189}$$

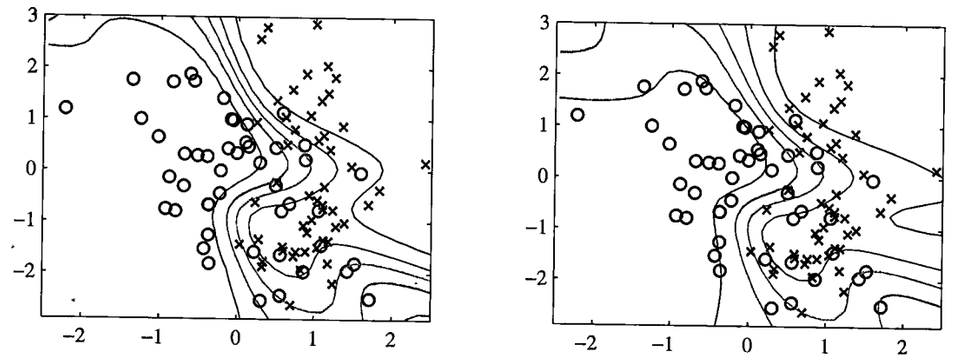


Figure 5.23 An illustration of the Laplace approximation for a Bayesian neural network having 8 hidden units with 'tanh' activation functions and a single logistic-sigmoid output unit. The weight parameters were found using scaled conjugate gradients, and the hyperparameter α was optimized using the evidence framework. On the left is the result of using the simple approximation (5.185) based on a point estimate $\mathbf{w}_{\mathrm{MAP}}$ 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

$$p(t = 1|\mathbf{x}, \mathcal{D}) = \sigma\left(\kappa(\sigma_a^2)\mathbf{b}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}}\right) \tag{5.190}$$

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 ($\star\star$) 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

$$\sigma(a) = \{1 + \exp(-a)\}^{-1}. \tag{5.191}$$

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 ($\star$) 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 ($\star\star$) 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

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x}, \mathbf{w}), \boldsymbol{\Sigma}) \tag{5.192}$$

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 ($\star\star$) 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 ($\star$) 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 ($\star$) 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 ($\star$) 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 ($\star$) 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 ($\star$) WWW The error function (5.21) for binary classification problems was derived for a network having a logistic-sigmoid output activation function, so that $0 \leqslant y(\mathbf{x}, \mathbf{w}) \leqslant 1$, and data having target values $t \in \{0, 1\}$. Derive the corresponding error function if we consider a network having an output $-1 \leqslant y(\mathbf{x}, \mathbf{w}) \leqslant 1$ and target values $t = 1$ for class $\mathcal{C}_1$ and $t = -1$ for class $\mathcal{C}_2$. What would be the appropriate choice of output unit activation function?

5.10 ($\star$) 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.

5.39 (⋆) 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).

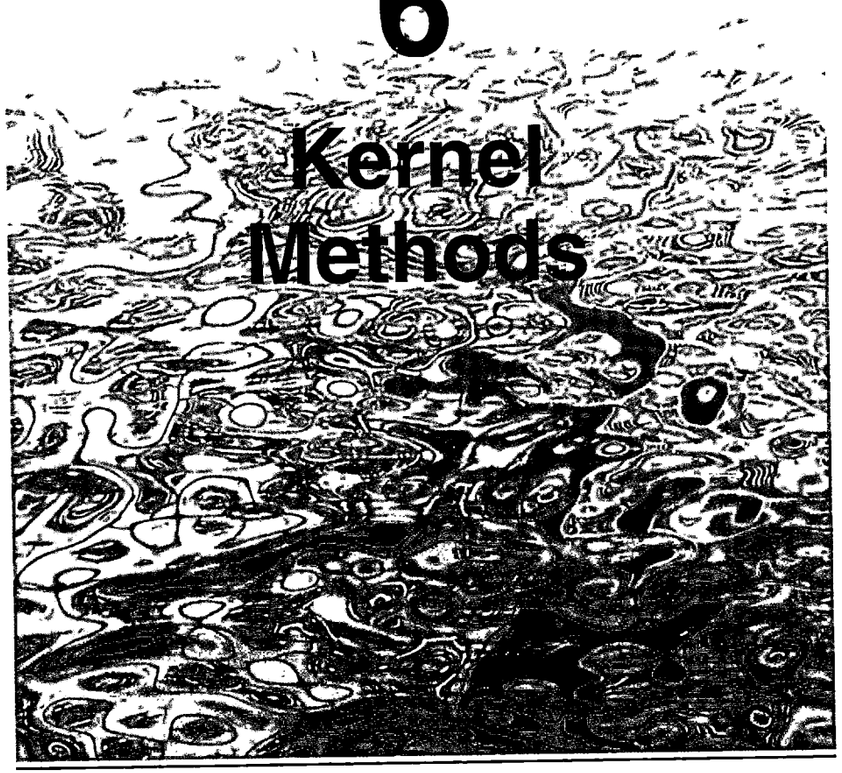
5.40 (⋆) WWW Outline the modifications needed to the framework for Bayesian neural networks, discussed in Section 5.7.3, to handle multiclass problems using networks having softmax output-unit activation functions.

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.

apter 5

tion 2.5.1

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

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}} \phi(\mathbf{x}'). \tag{6.1}$$

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.

Chapter 7

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 $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{x}'$. 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).

Section 12.3

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 $k(\mathbf{x}, \mathbf{x}') = k(\|\mathbf{x} - \mathbf{x}'\|)$.

Section 6.3

For recent textbooks on kernel methods, see Schölkopf and Smola (2002), Herbrich (2002), and Shawe-Taylor and Cristianini (2004).

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

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \tag{6.2}$$

where $\lambda \geqslant 0$. 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

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) - t_n\right\}\phi(\mathbf{x}_n) = \sum_{n=1}^{N} a_n\phi(\mathbf{x}_n) = \mathbf{\Phi}^{\mathrm{T}}\mathbf{a} \tag{6.3}$$

where $\mathbf{\Phi}$ is the design matrix, whose n^{th} row is given by $\phi(\mathbf{x}_n)^{\mathrm{T}}$. Here the vector $\mathbf{a} = (a_1, \ldots, a_N)^{\mathrm{T}}$, and we have defined

$$a_n = -\frac{1}{\lambda}\left\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) - t_n\right\}. \tag{6.4}$$

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 $\mathbf{w} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{a}$ into $J(\mathbf{w})$, we obtain

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{a} \tag{6.5}$$

where $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$. We now define the *Gram* matrix $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}$, which is an $N \times N$ symmetric matrix with elements

$$K_{nm} = \phi(\mathbf{x}_n)^{\mathrm{T}}\phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) \tag{6.6}$$

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

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}. \tag{6.7}$$

Setting the gradient of $J(\mathbf{a})$ with respect to $\mathbf{a}$ to zero, we obtain the following solution

$$\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}. \tag{6.8}$$

If we substitute this back into the linear regression model, we obtain the following prediction for a new input $\mathbf{x}$

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) = \mathbf{a}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\phi}(\mathbf{x}) = \mathbf{k}(\mathbf{x})^{\mathrm{T}}\left(\mathbf{K} + \lambda \mathbf{I}_N\right)^{-1}\mathbf{t} \tag{6.9}$$

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 $\boldsymbol{\phi}(\mathbf{x})$, 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.

Exercise 6.1

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 $\boldsymbol{\phi}(\mathbf{x})$, which allows us implicitly to use feature spaces of high, even infinite, dimensionality.

Exercise 6.2

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 $\boldsymbol{\phi}(\mathbf{x})$ 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

$$k(x, x') = \boldsymbol{\phi}(x)^{\mathrm{T}}\boldsymbol{\phi}(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x') \tag{6.10}$$

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

$$k(\mathbf{x}, \mathbf{z}) = \left(\mathbf{x}^{\mathrm{T}}\mathbf{z}\right)^2. \tag{6.11}$$

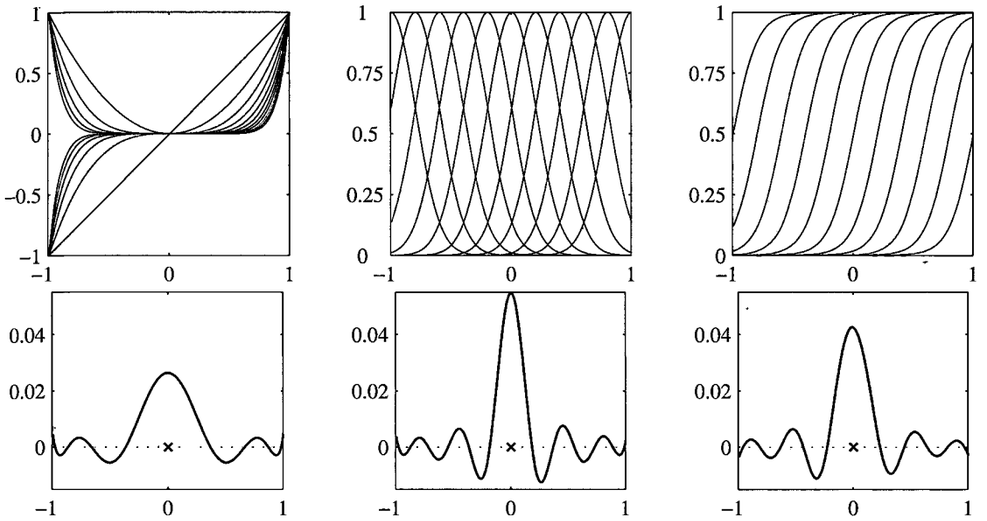


Figure 6.1 Illustration of the construction of kernel functions starting from a corresponding set of basis functions. In each column the lower plot shows the kernel function $k(x, x')$ defined by (6.10) plotted as a function of x for $x' = 0$, while the upper plot shows the corresponding basis functions 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

$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= \left(\mathbf{x}^{\mathrm{T}}\mathbf{z}\right)^2 = (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2}x_1 x_2, x_2^2)(z_1^2, \sqrt{2}z_1 z_2, z_2^2)^{\mathrm{T}} \\ &= \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{z}). \end{aligned} \tag{6.12}$$

We see that the feature mapping takes the form $\boldsymbol{\phi}(\mathbf{x}) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)^{\mathrm{T}}$ 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 $\boldsymbol{\phi}(\mathbf{x})$ 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.

Appendix C

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:

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') \tag{6.13}$$
$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \tag{6.14}$$
$$k(\mathbf{x}, \mathbf{x}') = q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \tag{6.15}$$
$$k(\mathbf{x}, \mathbf{x}') = \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \tag{6.16}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{6.17}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \tag{6.18}$$
$$k(\mathbf{x}, \mathbf{x}') = k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \tag{6.19}$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' \tag{6.20}$$
$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \tag{6.21}$$
$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b) \tag{6.22}$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ 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 $k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^{\mathrm{T}}\mathbf{x}'\right)^2$ contains only terms of degree two. If we consider the slightly generalized kernel $k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^{\mathrm{T}}\mathbf{x}' + c\right)^2$ with $c > 0$, then the corresponding feature mapping $\boldsymbol{\phi}(\mathbf{x})$ contains constant and linear terms as well as terms of order two. Similarly, $k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^{\mathrm{T}}\mathbf{x}'\right)^M$ 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 $k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^{\mathrm{T}}\mathbf{x}' + c\right)^M$ 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

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x} - \mathbf{x}'\|^2/2\sigma^2\right) \tag{6.23}$$

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

$$\|\mathbf{x} - \mathbf{x}'\|^2 = \mathbf{x}^{\mathrm{T}}\mathbf{x} + (\mathbf{x}')^{\mathrm{T}}\mathbf{x}' - 2\mathbf{x}^{\mathrm{T}}\mathbf{x}' \tag{6.24}$$

to give

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\mathbf{x}^{\mathrm{T}}\mathbf{x}/2\sigma^2\right)\exp\left(\mathbf{x}^{\mathrm{T}}\mathbf{x}'/\sigma^2\right)\exp\left(-(\mathbf{x}')^{\mathrm{T}}\mathbf{x}'/2\sigma^2\right) \tag{6.25}$$

and then making use of (6.14) and (6.16), together with the validity of the linear kernel $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{x}'$. Note that the feature vector that corresponds to the Gaussian kernel has infinite dimensionality.

Exercise 6.11

The Gaussian kernel is not restricted to the use of Euclidean distance. If we use kernel substitution in (6.24) to replace $\mathbf{x}^{\mathrm{T}}\mathbf{x}'$ with a nonlinear kernel $\kappa(\mathbf{x}, \mathbf{x}')$, we obtain

$$k(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{1}{2\sigma^2}\left(\kappa(\mathbf{x}, \mathbf{x}) + \kappa(\mathbf{x}', \mathbf{x}') - 2\kappa(\mathbf{x}, \mathbf{x}')\right)\right\}. \tag{6.26}$$

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

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|} \tag{6.27}$$

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.

Exercise 6.12

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

$$k(\mathbf{x}, \mathbf{x}') = p(\mathbf{x})p(\mathbf{x}'). \tag{6.28}$$

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

$$k(\mathbf{x}, \mathbf{x}') = \sum_i p(\mathbf{x}|i)p(\mathbf{x}'|i)p(i). \tag{6.29}$$

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

Section 9.2

$$k(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{x}'|\mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z} \tag{6.30}$$

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 $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_L\}$. 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 $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_L\}$. 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

Section 13.2

$$k(\mathbf{X}, \mathbf{X}') = \sum_{\mathbf{Z}} p(\mathbf{X}|\mathbf{Z})p(\mathbf{X}'|\mathbf{Z})p(\mathbf{Z}) \tag{6.31}$$

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*

$$\mathbf{g}(\boldsymbol{\theta}, \mathbf{x}) = \nabla_{\boldsymbol{\theta}} \ln p(\mathbf{x}|\boldsymbol{\theta}) \tag{6.32}$$

from which the Fisher kernel is defined by

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^{\mathrm{T}} \mathbf{F}^{-1} \mathbf{g}(\boldsymbol{\theta}, \mathbf{x}'). \tag{6.33}$$

Here $\mathbf{F}$ is the *Fisher information matrix*, given by

$$\mathbf{F} = \mathbb{E}_{\mathbf{x}}\left[\mathbf{g}(\boldsymbol{\theta}, \mathbf{x})\mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^{\mathrm{T}}\right] \tag{6.34}$$

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 $\boldsymbol{\theta} \rightarrow \psi(\boldsymbol{\theta})$.

Exercise 6.13

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

$$\mathbf{F} \simeq \frac{1}{N} \sum_{n=1}^{N} \mathbf{g}(\boldsymbol{\theta}, \mathbf{x}_n)\mathbf{g}(\boldsymbol{\theta}, \mathbf{x}_n)^{\mathrm{T}}. \tag{6.35}$$

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

Section 12.1.3

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{g}(\boldsymbol{\theta}, \mathbf{x})^{\mathrm{T}} \mathbf{g}(\boldsymbol{\theta}, \mathbf{x}'). \tag{6.36}$$

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

$$k(\mathbf{x}, \mathbf{x}') = \tanh\left(a\mathbf{x}^{\mathrm{T}}\mathbf{x}' + b\right) \tag{6.37}$$

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 $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ along with corresponding target values $\{t_1, \ldots, t_N\}$, 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 $n = 1, \ldots, N$. This is achieved by expressing $f(\mathbf{x})$ as a linear combination of radial basis functions, one centred on every data point

$$f(\mathbf{x}) = \sum_{n=1}^{N} w_n h(\|\mathbf{x} - \mathbf{x}_n\|). \tag{6.38}$$

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 $\boldsymbol{\xi}$ having a distribution $\nu(\boldsymbol{\xi})$, then the sum-of-squares error function becomes

$$E = \frac{1}{2}\sum_{n=1}^{N}\int \{y(\mathbf{x}_n + \boldsymbol{\xi}) - t_n\}^2 \nu(\boldsymbol{\xi})\,\mathrm{d}\boldsymbol{\xi}. \tag{6.39}$$

Appendix D

Using the calculus of variations, we can optimize with respect to the function $f(\mathbf{x})$ to give

Exercise 6.17

$$y(\mathbf{x}_n) = \sum_{n=1}^{N} t_n h(\mathbf{x} - \mathbf{x}_n) \tag{6.40}$$

where the basis functions are given by

$$h(\mathbf{x} - \mathbf{x}_n) = \frac{\nu(\mathbf{x} - \mathbf{x}_n)}{\sum_{n=1}^{N}\nu(\mathbf{x} - \mathbf{x}_n)}. \tag{6.41}$$

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 $\nu(\boldsymbol{\xi})$ is isotropic, so that it is a function only of $\|\boldsymbol{\xi}\|$, 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 $\boldsymbol{\mu}_i$ 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.

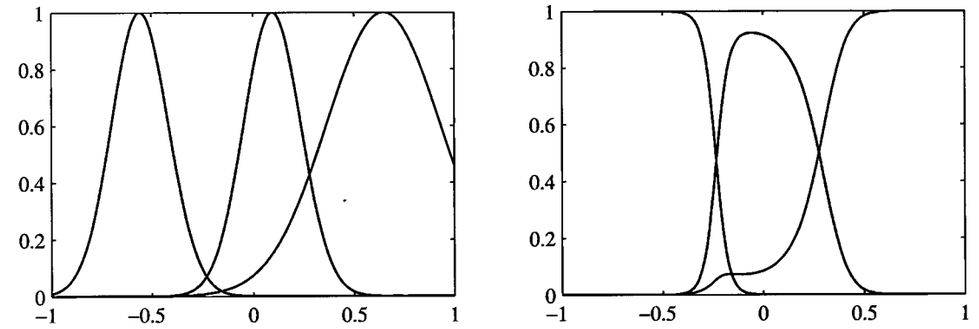


Figure 6.2 Plot of a set of Gaussian basis functions on the left, together with the corresponding normalized basis functions on the right.

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

Section 2.5.1

$$p(\mathbf{x}, t) = \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x} - \mathbf{x}_n, t - t_n) \tag{6.42}$$

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

$$\begin{aligned} y(\mathbf{x}) &= \mathbb{E}[t|\mathbf{x}] = \int_{-\infty}^{\infty} t p(t|\mathbf{x})\,\mathrm{d}t \\ &= \frac{\int t p(\mathbf{x},t)\,\mathrm{d}t}{\int p(\mathbf{x},t)\,\mathrm{d}t} \\ &= \frac{\sum_n \int t f(\mathbf{x}-\mathbf{x}_n, t-t_n)\,\mathrm{d}t}{\sum_m \int f(\mathbf{x}-\mathbf{x}_m, t-t_m)\,\mathrm{d}t}. \end{aligned} \tag{6.43}$$

We now assume for simplicity that the component density functions have zero mean so that

$$\int_{-\infty}^{\infty} f(\mathbf{x},t)t\,\mathrm{d}t = 0 \tag{6.44}$$

for all values of $\mathbf{x}$. Using a simple change of variable, we then obtain

$$\begin{aligned} y(\mathbf{x}) &= \frac{\sum_n g(\mathbf{x}-\mathbf{x}_n)t_n}{\sum_m g(\mathbf{x}-\mathbf{x}_m)} \\ &= \sum_n k(\mathbf{x},\mathbf{x}_n)t_n \end{aligned} \tag{6.45}$$

where $n, m = 1, \ldots, N$ and the kernel function $k(\mathbf{x}, \mathbf{x}_n)$ is given by

$$k(\mathbf{x},\mathbf{x}_n) = \frac{g(\mathbf{x}-\mathbf{x}_n)}{\sum_m g(\mathbf{x}-\mathbf{x}_m)} \tag{6.46}$$

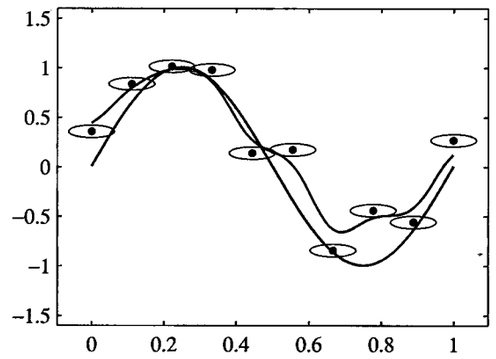
and we have defined

$$g(\mathbf{x}) = \int_{-\infty}^{\infty} f(\mathbf{x},t)\,\mathrm{d}t. \tag{6.47}$$

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 ellipse around each data point shows 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

$$p(t|\mathbf{x}) = \frac{p(t,\mathbf{x})}{\int p(t,\mathbf{x})\,\mathrm{d}t} = \frac{\sum_n f(\mathbf{x}-\mathbf{x}_n, t-t_n)}{\sum_m \int f(\mathbf{x}-\mathbf{x}_m, t-t_m)\,\mathrm{d}t} \tag{6.48}$$

from which other expectations can be evaluated.

Exercise 6.18

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 $y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$ in which $\mathbf{w}$ is a vector of parameters and $\boldsymbol{\phi}(\mathbf{x})$ 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 $\boldsymbol{\phi}(\mathbf{x})$ so that

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) \tag{6.49}$$

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

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) \tag{6.50}$$

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

$\mathbf{x}_1, \ldots, \mathbf{x}_N$. We are therefore interested in the joint distribution of the function values $y(\mathbf{x}_1), \ldots, y(\mathbf{x}_N)$, which we denote by the vector $\mathbf{y}$ with elements $y_n = y(\mathbf{x}_n)$ for $n = 1, \ldots, N$. From (6.49), this vector is given by

$$\mathbf{y} = \boldsymbol{\Phi}\mathbf{w} \tag{6.51}$$

where $\boldsymbol{\Phi}$ 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

Exercise 2.31

$$\mathbb{E}[\mathbf{y}] = \boldsymbol{\Phi}\mathbb{E}[\mathbf{w}] = \mathbf{0} \tag{6.52}$$

$$\mathrm{cov}[\mathbf{y}] = \mathbb{E}\left[\mathbf{y}\mathbf{y}^{\mathrm{T}}\right] = \boldsymbol{\Phi}\mathbb{E}\left[\mathbf{w}\mathbf{w}^{\mathrm{T}}\right]\boldsymbol{\Phi}^{\mathrm{T}} = \frac{1}{\alpha}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} = \mathbf{K} \tag{6.53}$$

where $\mathbf{K}$ is the Gram matrix with elements

$$K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \frac{1}{\alpha}\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_m) \tag{6.54}$$

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 $\mathbf{x}_1, \ldots, \mathbf{x}_N$ 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 $y(\mathbf{x}_1), \ldots, y(\mathbf{x}_N)$ in a consistent manner.

A key point about Gaussian stochastic processes is that the joint distribution over N variables $y_1, \ldots, y_N$ 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

$$\mathbb{E}\left[y(\mathbf{x}_n)y(\mathbf{x}_m)\right] = k(\mathbf{x}_n, \mathbf{x}_m). \tag{6.55}$$

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

$$k(x, x') = \exp\left(-\theta|x - x'|\right) \tag{6.56}$$

which corresponds to the *Ornstein-Uhlenbeck process* originally introduced by Uhlenbeck and Ornstein (1930) to describe Brownian motion.

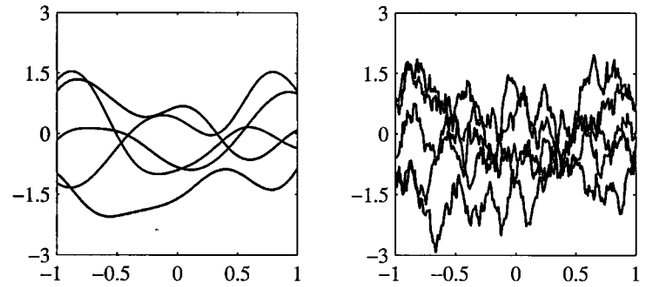


Figure 6.4 Samples from Gaussian processes for a 'Gaussian' kernel (left) and an exponential kernel (right).

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

$$t_n = y_n + \epsilon_n \tag{6.57}$$

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

$$p(t_n|y_n) = \mathcal{N}(t_n|y_n, \beta^{-1}) \tag{6.58}$$

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 $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$ conditioned on the values of $\mathbf{y} = (y_1, \ldots, y_N)^{\mathrm{T}}$ is given by an isotropic Gaussian of the form

$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}(\mathbf{t}|\mathbf{y}, \beta^{-1}\mathbf{I}_N) \tag{6.59}$$

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

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}). \tag{6.60}$$

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 $\mathbf{x}_1, \ldots, \mathbf{x}_N$, 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

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})\,\mathrm{d}\mathbf{y} = \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C}) \tag{6.61}$$

where the covariance matrix $\mathbf{C}$ has elements

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm}. \tag{6.62}$$

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

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left\{-\frac{\theta_1}{2}\|\mathbf{x}_n - \mathbf{x}_m\|^2\right\} + \theta_2 + \theta_3 \mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m. \tag{6.63}$$

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 $\theta_0, \ldots, \theta_3$ 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 $\mathbf{t}_N = (t_1, \ldots, t_N)^{\mathrm{T}}$, corresponding to input values $\mathbf{x}_1, \ldots, \mathbf{x}_N$, 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 $\mathbf{x}_1, \ldots, \mathbf{x}_N$ 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 $(t_1, \ldots, t_N, t_{N+1})^{\mathrm{T}}$. 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 $t_1, \ldots, t_{N+1}$ will be given by

$$p(\mathbf{t}_{N+1}) = \mathcal{N}(\mathbf{t}_{N+1}|\mathbf{0}, \mathbf{C}_{N+1}) \tag{6.64}$$

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

$$\mathbf{C}_{N+1} = \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^{\mathrm{T}} & c \end{pmatrix} \tag{6.65}$$

where $\mathbf{C}_N$ is the $N \times N$ covariance matrix with elements given by (6.62) for $n, m = 1, \ldots, N$, the vector $\mathbf{k}$ has elements $k(\mathbf{x}_n, \mathbf{x}_{N+1})$ for $n = 1, \ldots, N$, and the scalar

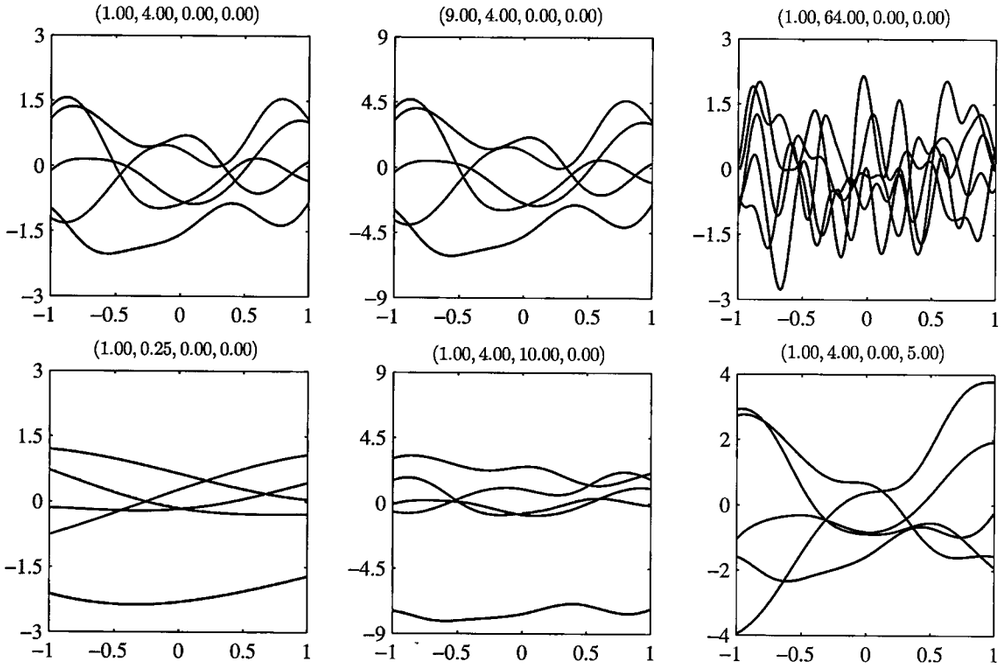


Figure 6.5 Samples from a Gaussian process prior defined by the covariance function (6.63). The title above each plot denotes $(\theta_0, \theta_1, \theta_2, \theta_3)$.

$c = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \beta^{-1}$. 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

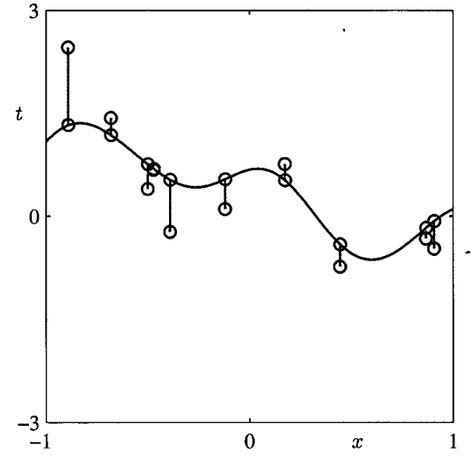
$$m(\mathbf{x}_{N+1}) = \mathbf{k}^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{t} \tag{6.66}$$

$$\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{k}. \tag{6.67}$$

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 $\lambda_i \geqslant 0$, 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

$$m(\mathbf{x}_{N+1}) = \sum_{n=1}^{N} a_n k(\mathbf{x}_n, \mathbf{x}_{N+1}) \tag{6.68}$$

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.

Exercise 6.21

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

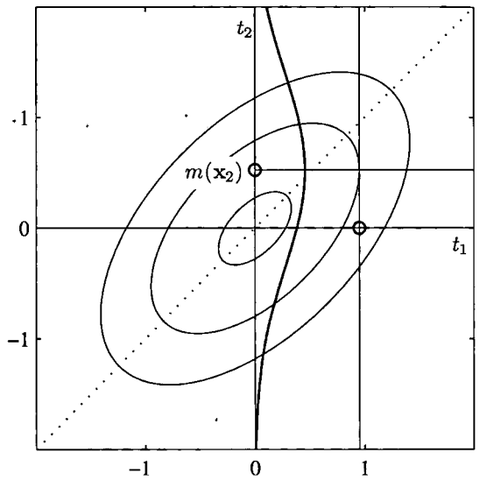


Figure 6.7 Illustration of the mechanism of Gaussian process regression for the case of one training point and one test point, in which the red ellipses show contours of the joint distribution $p(t_1, t_2)$. Here t_1 is the training data point, and conditioning on the value of t_1, corresponding to the 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

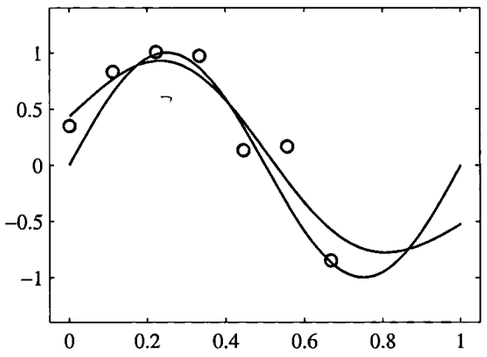


Figure 6.8 Illustration of Gaussian process regression applied to the sinusoidal data set in Figure A.6 in which the three right-most data points have been omitted. The green curve shows the sinusoidal function from which the data points, shown in blue, are obtained by sampling and addition of Gaussian 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).

Section 3.5

The log likelihood function for a Gaussian process regression model is easily evaluated using the standard form for a multivariate Gaussian distribution, giving

$$\ln p(\mathbf{t}|\boldsymbol{\theta}) = -\frac{1}{2}\ln|\mathbf{C}_N| - \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{t} - \frac{N}{2}\ln(2\pi). \tag{6.69}$$

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

$$\frac{\partial}{\partial\theta_i}\ln p(\mathbf{t}|\boldsymbol{\theta}) = -\frac{1}{2}\mathrm{Tr}\left(\mathbf{C}_N^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta_i}\right) + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{C}_N^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta_i}\mathbf{C}_N^{-1}\mathbf{t}. \tag{6.70}$$

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

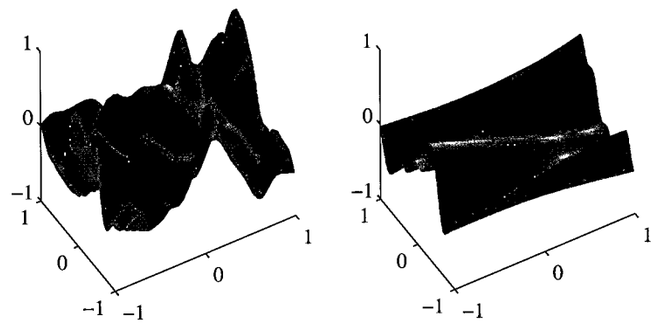


Figure 6.9 Samples from the ARD prior for Gaussian processes, in which the kernel function is given by (6.71). The left plot corresponds to $\eta_1 = \eta_2 = 1$, and the right plot corresponds to $\eta_1 = 1$, $\eta_2 = 0.01$.

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

$$k(\mathbf{x}, \mathbf{x}') = \theta_0 \exp\left\{-\frac{1}{2}\sum_{i=1}^{2} \eta_i (x_i - x_i')^2\right\}. \tag{6.71}$$

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

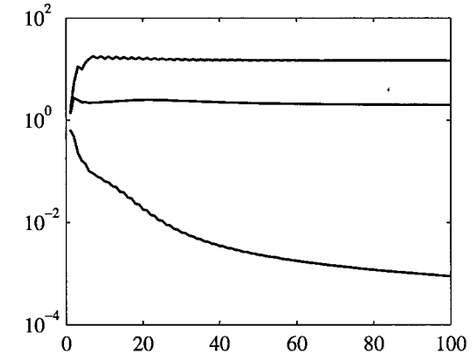


Figure 6.10 Illustration of automatic relevance determination in a Gaussian process for a synthetic problem having three inputs x_1, x_2, and x_3, for which the curves show the corresponding values of the hyperparameters η_1 (red), η_2 (green), and η_3 (blue) as a function of the number of iterations 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

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left\{-\frac{1}{2}\sum_{i=1}^{D} \eta_i (x_{ni} - x_{mi})^2\right\} + \theta_2 + \theta_3 \sum_{i=1}^{D} x_{ni} x_{mi} \tag{6.72}$$

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-

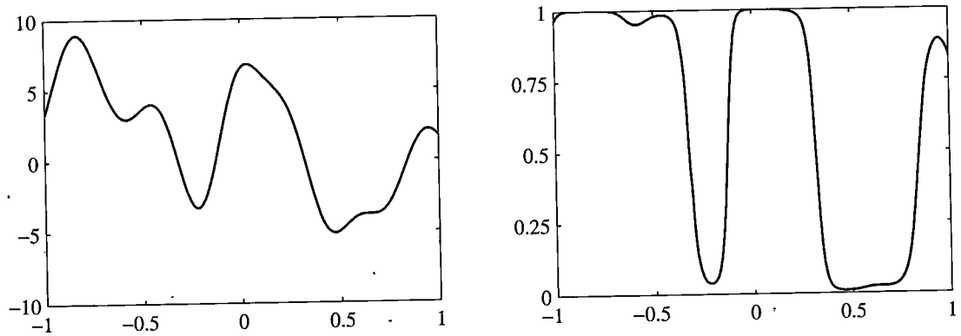


Figure 6.11 The left plot shows a sample from a Gaussian process prior over functions $a(\mathbf{x})$, and the right plot shows the result of transforming this sample using a logistic sigmoid function.

bution over the target variable t is then given by the Bernoulli distribution

$$p(t|a) = \sigma(a)^t(1-\sigma(a))^{1-t}. \tag{6.73}$$

As usual, we denote the training set inputs by $\mathbf{x}_1, \ldots, \mathbf{x}_N$ with corresponding observed target variables $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$. 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 $a(\mathbf{x}_1), \ldots, a(\mathbf{x}_{N+1})$. 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

$$p(\mathbf{a}_{N+1}) = \mathcal{N}(\mathbf{a}_{N+1}|\mathbf{0}, \mathbf{C}_{N+1}). \tag{6.74}$$

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

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \nu\delta_{nm} \tag{6.75}$$

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 $\boldsymbol{\theta}$ of parameters, and we shall later discuss how $\boldsymbol{\theta}$ 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

$$p(t_{N+1} = 1|\mathbf{t}_N) = \int p(t_{N+1} = 1|a_{N+1})p(a_{N+1}|\mathbf{t}_N)\,\mathrm{d}a_{N+1} \tag{6.76}$$

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

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 $p(\mathbf{t}_N|\boldsymbol{\theta})$. 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).

Section 10.1

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.

Section 10.7

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

Section 4.4

$$\begin{aligned} p(a_{N+1}|\mathbf{t}_N) &= \int p(a_{N+1}, \mathbf{a}_N|\mathbf{t}_N)\,\mathrm{d}\mathbf{a}_N \\ &= \frac{1}{p(\mathbf{t}_N)} \int p(a_{N+1}, \mathbf{a}_N)p(\mathbf{t}_N|a_{N+1}, \mathbf{a}_N)\,\mathrm{d}\mathbf{a}_N \\ &= \frac{1}{p(\mathbf{t}_N)} \int p(a_{N+1}|\mathbf{a}_N)p(\mathbf{a}_N)p(\mathbf{t}_N|\mathbf{a}_N)\,\mathrm{d}\mathbf{a}_N \\ &= \int p(a_{N+1}|\mathbf{a}_N)p(\mathbf{a}_N|\mathbf{t}_N)\,\mathrm{d}\mathbf{a}_N \end{aligned} \tag{6.77}$$

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

$$p(a_{N+1}|\mathbf{a}_N) = \mathcal{N}(a_{N+1}|\mathbf{k}^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{a}_N, c - \mathbf{k}^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{k}). \tag{6.78}$$

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

$$p(\mathbf{t}_N|\mathbf{a}_N) = \prod_{n=1}^{N} \sigma(a_n)^{t_n}(1 - \sigma(a_n))^{1-t_n} = \prod_{n=1}^{N} e^{a_n t_n}\sigma(-a_n). \tag{6.79}$$

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

$$\begin{aligned}\Psi(\mathbf{a}_N) &= \ln p(\mathbf{a}_N) + \ln p(\mathbf{t}_N|\mathbf{a}_N) \\ &= -\frac{1}{2}\mathbf{a}_N^{\mathrm{T}}\mathbf{C}_N^{-1}\mathbf{a}_N - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\mathbf{C}_N| + \mathbf{t}_N^{\mathrm{T}}\mathbf{a}_N \\ &\quad - \sum_{n=1}^{N}\ln(1 + e^{a_n}) + \text{const.}\end{aligned} \tag{6.80}$$

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

$$\nabla\Psi(\mathbf{a}_N) = \mathbf{t}_N - \boldsymbol{\sigma}_N - \mathbf{C}_N^{-1}\mathbf{a}_N \tag{6.81}$$

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

Section 4.3.3

$$\nabla\nabla\Psi(\mathbf{a}_N) = -\mathbf{W}_N - \mathbf{C}_N^{-1} \tag{6.82}$$

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

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

Exercise 6.25

$$\mathbf{a}_N^{\text{new}} = \mathbf{C}_N(\mathbf{I} + \mathbf{W}_N\mathbf{C}_N)^{-1}\{\mathbf{t}_N - \boldsymbol{\sigma}_N + \mathbf{W}_N\mathbf{a}_N\}. \tag{6.83}$$

These equations are iterated until they converge to the mode which we denote by $\mathbf{a}_N^\star$. At the mode, the gradient $\nabla\Psi(\mathbf{a}_N)$ will vanish, and hence $\mathbf{a}_N^\star$ will satisfy

$$\mathbf{a}_N^\star = \mathbf{C}_N(\mathbf{t}_N - \boldsymbol{\sigma}_N). \tag{6.84}$$

Once we have found the mode $\mathbf{a}_N^\star$ of the posterior, we can evaluate the Hessian matrix given by

$$\mathbf{H} = -\nabla\nabla\Psi(\mathbf{a}_N) = \mathbf{W}_N + \mathbf{C}_N^{-1} \tag{6.85}$$

where the elements of $\mathbf{W}_N$ are evaluated using $\mathbf{a}_N^\star$. This defines our Gaussian approximation to the posterior distribution $p(\mathbf{a}_N|\mathbf{t}_N)$ given by

$$q(\mathbf{a}_N) = \mathcal{N}(\mathbf{a}_N|\mathbf{a}_N^\star, \mathbf{H}^{-1}). \tag{6.86}$$

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

Exercise 6.26

$$\begin{aligned}\mathbb{E}[a_{N+1}|\mathbf{t}_N] &= \mathbf{k}^{\mathrm{T}}(\mathbf{t}_N - \boldsymbol{\sigma}_N) &(6.87)\\ \text{var}[a_{N+1}|\mathbf{t}_N] &= c - \mathbf{k}^{\mathrm{T}}(\mathbf{W}_N^{-1} + \mathbf{C}_N)^{-1}\mathbf{k}. &(6.88)\end{aligned}$$

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

$$p(\mathbf{t}_N|\boldsymbol{\theta}) = \int p(\mathbf{t}_N|\mathbf{a}_N)p(\mathbf{a}_N|\boldsymbol{\theta})\,\mathrm{d}\mathbf{a}_N. \tag{6.89}$$

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

$$\ln p(\mathbf{t}_N|\boldsymbol{\theta}) = \Psi(\mathbf{a}_N^\star) - \frac{1}{2}\ln|\mathbf{W}_N + \mathbf{C}_N^{-1}| + \frac{N}{2}\ln(2\pi) \tag{6.90}$$

where $\Psi(\mathbf{a}_N^\star) = \ln p(\mathbf{a}_N^\star|\boldsymbol{\theta}) + \ln p(\mathbf{t}_N|\mathbf{a}_N^\star)$. 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 $\mathbf{a}_N^\star$, 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 $\mathbf{a}_N^\star$ 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

$$\frac{\partial \ln p(\mathbf{t}_N|\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{2}\mathbf{a}_N^{\star\mathrm{T}}\mathbf{C}_N^{-1}\frac{\partial \mathbf{C}_N}{\partial \theta_j}\mathbf{C}_N^{-1}\mathbf{a}_N^\star - \frac{1}{2}\mathrm{Tr}\left[(\mathbf{I}+\mathbf{C}_N\mathbf{W}_N)^{-1}\mathbf{W}_N\frac{\partial \mathbf{C}_N}{\partial \theta_j}\right]. \tag{6.91}$$

To compute the terms arising from the dependence of $\mathbf{a}_N^\star$ on $\boldsymbol{\theta}$, we note that the Laplace approximation has been constructed such that $\Psi(\mathbf{a}_N)$ has zero gradient at $\mathbf{a}_N = \mathbf{a}_N^\star$, and so $\Psi(\mathbf{a}_N^\star)$ gives no contribution to the gradient as a result of its dependence on $\mathbf{a}_N^\star$. This leaves the following contribution to the derivative with respect to a component θ_j of $\boldsymbol{\theta}$

$$\begin{aligned} &-\frac{1}{2}\sum_{n=1}^{N}\frac{\partial \ln|\mathbf{W}_N+\mathbf{C}_N^{-1}|}{\partial a_n^\star}\frac{\partial a_n^\star}{\partial \theta_j} \\ &= -\frac{1}{2}\sum_{n=1}^{N}\left[(\mathbf{I}+\mathbf{C}_N\mathbf{W}_N)^{-1}\mathbf{C}_N\right]_{nn}\sigma_n^\star(1-\sigma_n^\star)(1-2\sigma_n^\star)\frac{\partial a_n^\star}{\partial \theta_j} \end{aligned} \tag{6.92}$$

where $\sigma_n^\star = \sigma(a_n^\star)$, and again we have used the result (C.22) together with the definition of $\mathbf{W}_N$. We can evaluate the derivative of $a_N^\star$ with respect to θ_j by differentiating the relation (6.84) with respect to θ_j to give

$$\frac{\partial a_n^\star}{\partial \theta_j} = \frac{\partial \mathbf{C}_N}{\partial \theta_j}(\mathbf{t}_N - \boldsymbol{\sigma}_N) - \mathbf{C}_N\mathbf{W}_N\frac{\partial a_n^\star}{\partial \theta_j}. \tag{6.93}$$

Rearranging then gives

$$\frac{\partial a_n^\star}{\partial \theta_j} = (\mathbf{I}+\mathbf{W}_N\mathbf{C}_N)^{-1}\frac{\partial \mathbf{C}_N}{\partial \theta_j}(\mathbf{t}_N - \boldsymbol{\sigma}_N). \tag{6.94}$$

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).

Appendix A

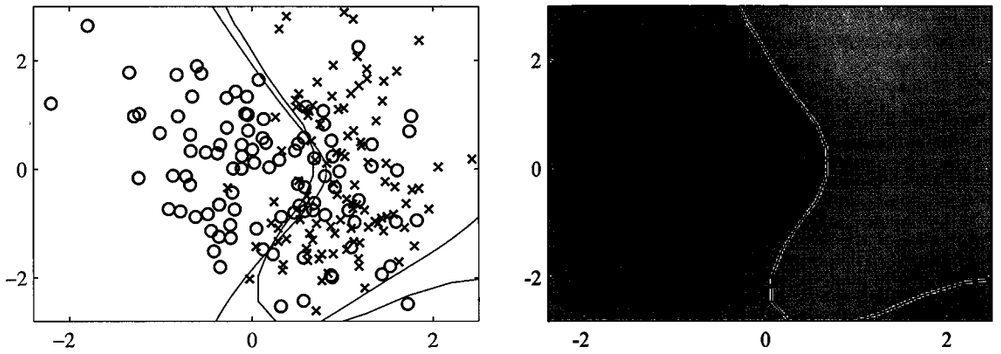


Figure 6.12 Illustration of the use of a Gaussian process for classification, showing the data on the left together with the optimal decision boundary from the true distribution in green, and the decision boundary from the Gaussian process classifier in black. On the right is the predicted posterior 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 $\mathbf{w}$, in conjunction with the network function $f(\mathbf{x}, \mathbf{w})$, produces a prior distribution over functions from $y(\mathbf{x})$ where $\mathbf{y}$ is the vector of network outputs. Neal (1996) has shown that, for a broad class of prior distributions over $\mathbf{w}$, the distribution of functions generated by a neural network will tend to a Gaussian process in the limit $M \to \infty$. 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 $k(\mathbf{x}, \mathbf{x}')$ are nonstationary, i.e. they cannot be expressed as a function of the difference $\mathbf{x} - \mathbf{x}'$, 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 $\boldsymbol{\phi}(\mathbf{x}_n)$. 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 $t_n\boldsymbol{\phi}(\mathbf{x}_n)$ 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 $\boldsymbol{\phi}(\mathbf{x})$ enters only in the form of the kernel function $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}')$.

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 $\boldsymbol{\phi}(A)$ where A is a subset of D and the element $\phi_U(A)$, indexed by the subset U, is given by

$$\phi_U(A) = \begin{cases} 1, & \text{if } U \subseteq A; \\ 0, & \text{otherwise.} \end{cases} \tag{6.95}$$

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 $\boldsymbol{\theta} \to \boldsymbol{\psi}(\boldsymbol{\theta})$, where the function $\boldsymbol{\psi}(\cdot)$ 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

$$k(x_1, x_2)^2 \leqslant k(x_1, x_1)k(x_2, x_2). \tag{6.96}$$

6.16 (⋆⋆) Consider a parametric model governed by the parameter vector $\mathbf{w}$ together with a data set of input values $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and a nonlinear feature mapping $\boldsymbol{\phi}(\mathbf{x})$. Suppose that the dependence of the error function on $\mathbf{w}$ takes the form

$$J(\mathbf{w}) = f(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_N)) + g(\mathbf{w}^{\mathrm{T}}\mathbf{w}) \tag{6.97}$$

where $g(\cdot)$ is a monotonically increasing function. By writing $\mathbf{w}$ in the form

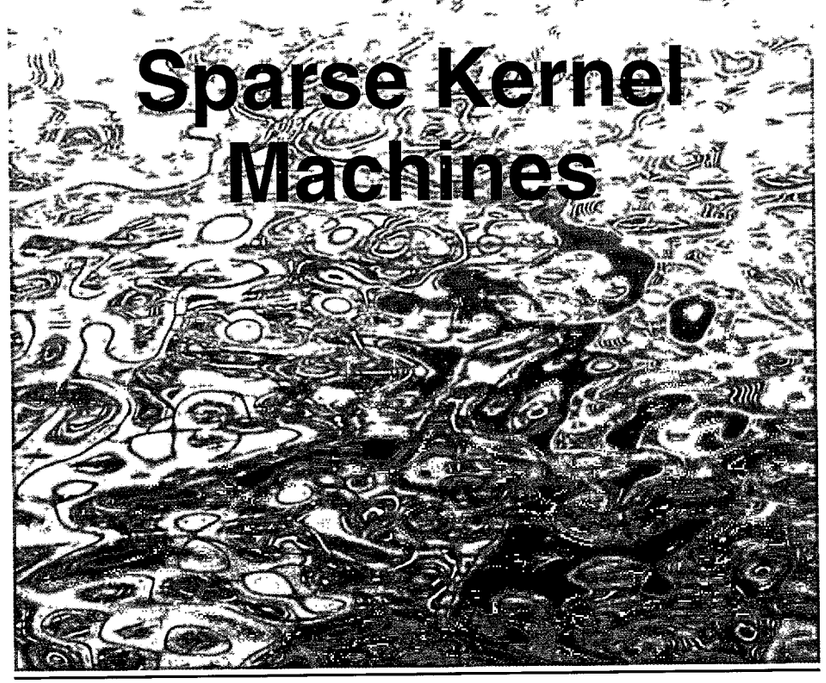
$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n \boldsymbol{\phi}(\mathbf{x}_n) + \mathbf{w}_\perp \tag{6.98}$$

show that the value of $\mathbf{w}$ that minimizes $J(\mathbf{w})$ takes the form of a linear combination of the basis functions $\boldsymbol{\phi}(\mathbf{x}_n)$ for $n = 1, \ldots, N$.

6.17 (⋆⋆) WWW Consider the sum-of-squares error function (6.39) for data having noisy inputs, where $\nu(\boldsymbol{\xi})$ 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).

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

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) + b \tag{7.1}$$

where $\boldsymbol{\phi}(\mathbf{x})$ 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 $\mathbf{x}_1, \ldots, \mathbf{x}_N$, with corresponding target values $t_1, \ldots, t_N$ 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

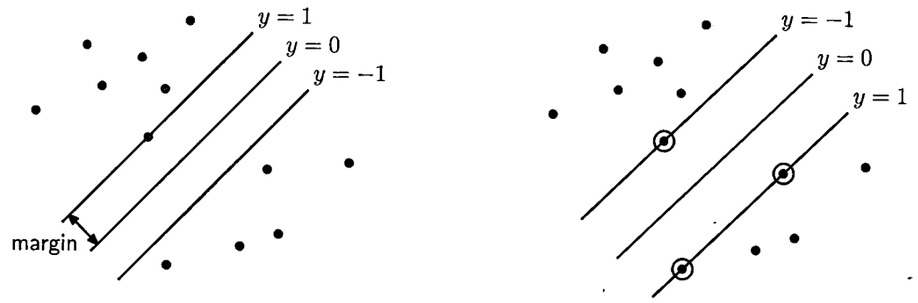


Figure 7.1 The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left figure. Maximizing the margin leads to a particular choice of decision boundary, as shown on the right. The location of this boundary is determined by 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 $\sigma^2 \to 0$, 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

$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}. \tag{7.2}$$

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

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[t_n \left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b \right) \right] \right\} \tag{7.3}$$

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 $\mathbf{w} \to \kappa\mathbf{w}$ and $b \to \kappa b$, 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

$$t_n \left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b\right) = 1 \tag{7.4}$$

for the point that is closest to the surface. In this case, all data points will satisfy the constraints

$$t_n \left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b\right) \geqslant 1, \qquad n = 1, \ldots, N. \tag{7.5}$$

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

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{7.6}$$

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.

Appendix E

In order to solve this constrained optimization problem, we introduce Lagrange multipliers $a_n \geqslant 0$, with one multiplier a_n for each of the constraints in (7.5), giving the Lagrangian function

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{t_n(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) + b) - 1\right\} \tag{7.7}$$

where $\mathbf{a} = (a_1, \ldots, a_N)^{\mathrm{T}}$. 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

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n) \tag{7.8}$$

$$0 = \sum_{n=1}^{N} a_n t_n. \tag{7.9}$$

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

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{7.10}$$

with respect to $\mathbf{a}$ subject to the constraints

$$a_n \geqslant 0, \qquad n = 1, \ldots, N, \tag{7.11}$$

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{7.12}$$

Here the kernel function is defined by $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}')$. 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 $\widetilde{L}(\mathbf{a})$ 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

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b. \tag{7.13}$$

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

$$a_n \geqslant 0 \tag{7.14}$$

$$t_n y(\mathbf{x}_n) - 1 \geqslant 0 \tag{7.15}$$

$$a_n \{t_n y(\mathbf{x}_n) - 1\} = 0. \tag{7.16}$$

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

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1 \tag{7.17}$$

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

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right) \tag{7.18}$$

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

$$\sum_{n=1}^{N} E_\infty(y(\mathbf{x}_n) t_n - 1) + \lambda \|\mathbf{w}\|^2 \tag{7.19}$$

where $E_\infty(z)$ is a function that is zero if $z \geqslant 0$ 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

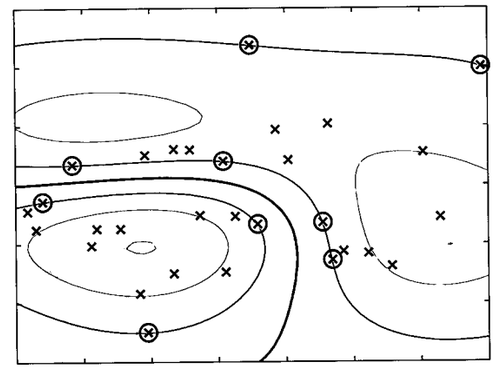


Figure 7.2 Example of synthetic data from two classes in two dimensions showing contours of constant $y(\mathbf{x})$ obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.

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*, $\xi_n \geqslant 0$ where $n = 1, \ldots, N$, 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

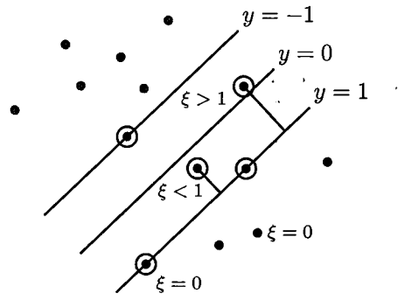


Figure 7.3 Illustration of the slack variables $\xi_n \geqslant 0$. Data points with circles around them are support vectors.

with $\xi_n > 1$ will be misclassified. The exact classification constraints (7.5) are then replaced with

$$t_n y(\mathbf{x}_n) \geqslant 1 - \xi_n, \qquad n = 1, \ldots, N \tag{7.20}$$

in which the slack variables are constrained to satisfy $\xi_n \geqslant 0$. 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 $0 < \xi_n \leqslant 1$ 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

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2}\|\mathbf{w}\|^2 \tag{7.21}$$

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 $C \to \infty$, 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 $\xi_n \geqslant 0$. The corresponding Lagrangian is given by

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} a_n \{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N} \mu_n \xi_n \tag{7.22}$$

Appendix E

where $\{a_n \geqslant 0\}$ and $\{\mu_n \geqslant 0\}$ are Lagrange multipliers. The corresponding set of KKT conditions are given by

$$a_n \geqslant 0 \tag{7.23}$$
$$t_n y(\mathbf{x}_n) - 1 + \xi_n \geqslant 0 \tag{7.24}$$
$$a_n \left(t_n y(\mathbf{x}_n) - 1 + \xi_n\right) = 0 \tag{7.25}$$
$$\mu_n \geqslant 0 \tag{7.26}$$
$$\xi_n \geqslant 0 \tag{7.27}$$
$$\mu_n \xi_n = 0 \tag{7.28}$$

where $n = 1, \ldots, N$.

We now optimize out $\mathbf{w}$, b, and $\{\xi_n\}$ making use of the definition (7.1) of $y(\mathbf{x})$ to give

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n) \tag{7.29}$$

$$\frac{\partial L}{\partial b} = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} a_n t_n = 0 \tag{7.30}$$

$$\frac{\partial L}{\partial \xi_n} = 0 \quad \Rightarrow \quad a_n = C - \mu_n. \tag{7.31}$$

Using these results to eliminate $\mathbf{w}$, b, and $\{\xi_n\}$ from the Lagrangian, we obtain the dual Lagrangian in the form

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{7.32}$$

which is identical to the separable case, except that the constraints are somewhat different. To see what these constraints are, we note that $a_n \geqslant 0$ is required because these are Lagrange multipliers. Furthermore, (7.31) together with $\mu_n \geqslant 0$ implies $a_n \leqslant C$. We therefore have to minimize (7.32) with respect to the dual variables $\{a_n\}$ subject to

$$0 \leqslant a_n \leqslant C \tag{7.33}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{7.34}$$

for $n = 1, \ldots, N$, 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

$$t_n y(\mathbf{x}_n) = 1 - \xi_n. \tag{7.35}$$

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 $\xi_n \leqslant 1$ 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

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1. \tag{7.36}$$

Again, a numerically stable solution is obtained by averaging to give

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right) \tag{7.37}$$

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

$$\widetilde{L}(\mathbf{a}) = -\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{7.38}$$

subject to the constraints

$$0 \leqslant a_n \leqslant 1/N \tag{7.39}$$

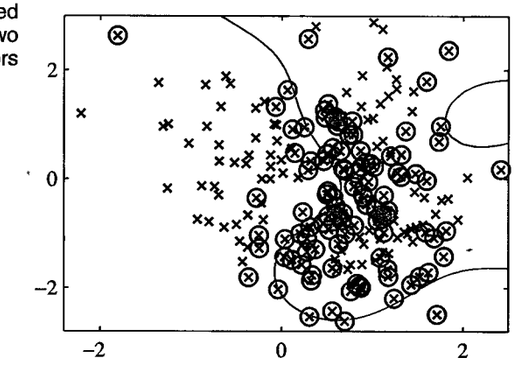
$$\sum_{n=1}^{N} a_n t_n = 0 \tag{7.40}$$

$$\sum_{n=1}^{N} a_n \geqslant \nu. \tag{7.41}$$

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 $\widetilde{L}(\mathbf{a})$ 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

$$\begin{aligned} k(\mathbf{x},\mathbf{z}) &= \left(1+\mathbf{x}^{\mathrm{T}}\mathbf{z}\right)^2 = (1+x_1z_1+x_2z_2)^2 \\ &= 1+2x_1z_1+2x_2z_2+x_1^2z_1^2+2x_1z_1x_2z_2+x_2^2z_2^2 \\ &= (1,\sqrt{2}x_1,\sqrt{2}x_2,x_1^2,\sqrt{2}x_1x_2,x_2^2)(1,\sqrt{2}z_1,\sqrt{2}z_2,z_1^2,\sqrt{2}z_1z_2,z_2^2)^{\mathrm{T}} \\ &= \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{z}). \end{aligned} \tag{7.42}$$

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 $\boldsymbol{\phi}(\mathbf{x})$. 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

$$p(t=1|\mathbf{x}) = \sigma\left(Ay(\mathbf{x})+B\right) \tag{7.43}$$

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).

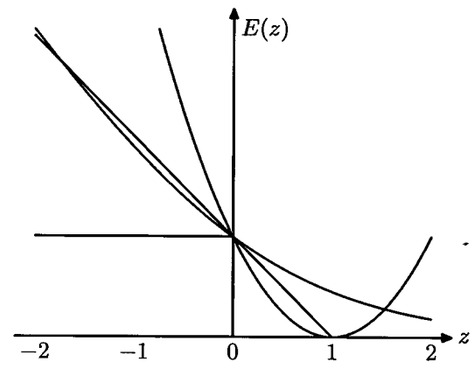
7.1.2 Relation to logistic regression

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.

Section 4.3.2

We have seen that for data points that are on the correct side of the margin boundary, and which therefore satisfy $y_nt_n \geqslant 1$, we have $\xi_n = 0$, and for the

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 $\xi_n = 1 - y_nt_n$. Thus the objective function (7.21) can be written (up to an overall multiplicative constant) in the form

$$\sum_{n=1}^{N} E_{\mathrm{SV}}(y_nt_n) + \lambda\|\mathbf{w}\|^2 \tag{7.44}$$

where $\lambda = (2C)^{-1}$, and $E_{\mathrm{SV}}(\cdot)$ is the *hinge* error function defined by

$$E_{\mathrm{SV}}(y_nt_n) = [1-y_nt_n]_+ \tag{7.45}$$

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

$$p(t|y) = \sigma(yt). \tag{7.46}$$

Exercise 7.6

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

$$\sum_{n=1}^{N} E_{\mathrm{LR}}(y_nt_n) + \lambda\|\mathbf{w}\|^2. \tag{7.47}$$

where

$$E_{\mathrm{LR}}(yt) = \ln\left(1+\exp(-yt)\right). \tag{7.48}$$

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 $E_{\mathrm{SV}}(yt)$ 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 $\mathcal{C}_k$ 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

$$y(\mathbf{x}) = \max_k y_k(\mathbf{x}). \tag{7.49}$$

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

Section 3.1.4

We now extend support vector machines to regression problems while at the same time preserving the property of sparseness. In simple linear regression, we

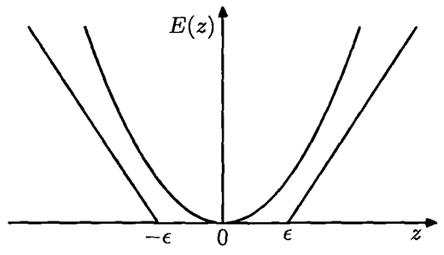


Figure 7.6 Plot of an ϵ-insensitive error function (in red) in which the error increases linearly with distance beyond the insensitive region. Also shown for comparison is the quadratic error function (in green).

minimize a regularized error function given by

$$\frac{1}{2}\sum_{n=1}^{N}\{y_n - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2. \tag{7.50}$$

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

$$E_\epsilon(y(\mathbf{x}) - t) = \begin{cases} 0, & \text{if } |y(\mathbf{x}) - t| < \epsilon; \\ |y(\mathbf{x}) - t| - \epsilon, & \text{otherwise} \end{cases} \tag{7.51}$$

and is illustrated in Figure 7.6.

We therefore minimize a regularized error function given by

$$C\sum_{n=1}^{N} E_\epsilon(y(\mathbf{x}_n) - t_n) + \frac{1}{2}\|\mathbf{w}\|^2. \tag{7.52}$$

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 $\xi_n \geqslant 0$ and $\widehat{\xi}_n \geqslant 0$, where $\xi_n > 0$ corresponds to a point for which $t_n > y(\mathbf{x}_n) + \epsilon$, and $\widehat{\xi}_n > 0$ 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 $y_n - \epsilon \leqslant t_n \leqslant y_n + \epsilon$, 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

$$t_n \leqslant y(\mathbf{x}_n) + \epsilon + \xi_n \tag{7.53}$$

$$t_n \geqslant y(\mathbf{x}_n) - \epsilon - \widehat{\xi}_n. \tag{7.54}$$

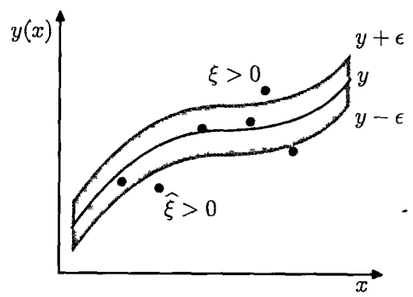


Figure 7.7 Illustration of SVM regression, showing the regression curve together with the ϵ-insensitive 'tube'. Also shown are examples of the slack variables ξ and $\widehat{\xi}$. Points above the ϵ-tube have $\xi > 0$ and $\widehat{\xi} = 0$, points below the ϵ-tube have $\xi = 0$ and $\widehat{\xi} > 0$, and points inside the ϵ-tube have $\xi = \widehat{\xi} = 0$.

The error function for support vector regression can then be written as

$$C\sum_{n=1}^{N}(\xi_n + \widehat{\xi}_n) + \frac{1}{2}\|\mathbf{w}\|^2 \tag{7.55}$$

which must be minimized subject to the constraints $\xi_n \geqslant 0$ and $\widehat{\xi}_n \geqslant 0$ as well as (7.53) and (7.54). This can be achieved by introducing Lagrange multipliers $a_n \geqslant 0$, $\widehat{a}_n \geqslant 0$, $\mu_n \geqslant 0$, and $\widehat{\mu}_n \geqslant 0$ and optimizing the Lagrangian

$$\begin{aligned} L = {} & C\sum_{n=1}^{N}(\xi_n + \widehat{\xi}_n) + \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N}(\mu_n\xi_n + \widehat{\mu}_n\widehat{\xi}_n) \\ & - \sum_{n=1}^{N} a_n(\epsilon + \xi_n + y_n - t_n) - \sum_{n=1}^{N}\widehat{a}_n(\epsilon + \widehat{\xi}_n - y_n + t_n). \end{aligned} \tag{7.56}$$

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 $\widehat{\xi}_n$ to zero, giving

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{n=1}^{N}(a_n - \widehat{a}_n)\boldsymbol{\phi}(\mathbf{x}_n) \tag{7.57}$$

$$\frac{\partial L}{\partial b} = 0 \quad \Rightarrow \quad \sum_{n=1}^{N}(a_n - \widehat{a}_n) = 0 \tag{7.58}$$

$$\frac{\partial L}{\partial \xi_n} = 0 \quad \Rightarrow \quad a_n + \mu_n = C \tag{7.59}$$

$$\frac{\partial L}{\partial \widehat{\xi}_n} = 0 \quad \Rightarrow \quad \widehat{a}_n + \widehat{\mu}_n = C. \tag{7.60}$$

Exercise 7.7

Using these results to eliminate the corresponding variables from the Lagrangian, we see that the dual problem involves maximizing

$$\widetilde{L}(\mathbf{a}, \widehat{\mathbf{a}}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) - \epsilon\sum_{n=1}^{N}(a_n + \widehat{a}_n) + \sum_{n=1}^{N}(a_n - \widehat{a}_n)t_n \tag{7.61}$$

with respect to $\{a_n\}$ and $\{\widehat{a}_n\}$, where we have introduced the kernel $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}')$. Again, this is a constrained maximization, and to find the constraints we note that $a_n \geqslant 0$ and $\widehat{a}_n \geqslant 0$ are both required because these are Lagrange multipliers. Also $\mu_n \geqslant 0$ and $\widehat{\mu}_n \geqslant 0$ together with (7.59) and (7.60), require $a_n \leqslant C$ and $\widehat{a}_n \leqslant C$, and so again we have the box constraints

$$0 \leqslant a_n \leqslant C \tag{7.62}$$

$$0 \leqslant \widehat{a}_n \leqslant C \tag{7.63}$$

together with the condition (7.58).

Substituting (7.57) into (7.1), we see that predictions for new inputs can be made using

$$y(\mathbf{x}) = \sum_{n=1}^{N}(a_n - \widehat{a}_n)k(\mathbf{x}, \mathbf{x}_n) + b \tag{7.64}$$

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

$$a_n(\epsilon + \xi_n + y_n - t_n) = 0 \tag{7.65}$$

$$\widehat{a}_n(\epsilon + \widehat{\xi}_n - y_n + t_n) = 0 \tag{7.66}$$

$$(C - a_n)\xi_n = 0 \tag{7.67}$$

$$(C - \widehat{a}_n)\widehat{\xi}_n = 0. \tag{7.68}$$

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 $\widehat{a}_n$ implies $\epsilon + \widehat{\xi}_n - y_n + t_n = 0$, 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 $\epsilon + \widehat{\xi}_n - y_n + t_n = 0$ are incompatible, as is easily seen by adding them together and noting that ξ_n and $\widehat{\xi}_n$ are nonnegative while ϵ is strictly positive, and so for every data point $\mathbf{x}_n$, either a_n or $\widehat{a}_n$ (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 $\widehat{a}_n \neq 0$. These are points that lie on the boundary of the ϵ-tube or outside the tube. All points within the tube have

$a_n = \widehat{a}_n = 0$. 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

$$b = t_n - \epsilon - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) = t_n - \epsilon - \sum_{m=1}^{N}(a_m - \widehat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) \tag{7.69}$$

where we have used (7.57). We can obtain an analogous result by considering a point for which $0 < \widehat{a}_n < C$. 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

$$\widetilde{L}(\mathbf{a}, \widehat{\mathbf{a}}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) + \sum_{n=1}^{N}(a_n - \widehat{a}_n)t_n \tag{7.70}$$

subject to the constraints

$$0 \leqslant a_n \leqslant C/N \tag{7.71}$$

$$0 \leqslant \widehat{a}_n \leqslant C/N \tag{7.72}$$

$$\sum_{n=1}^{N}(a_n - \widehat{a}_n) = 0 \tag{7.73}$$

$$\sum_{n=1}^{N}(a_n + \widehat{a}_n) \leqslant \nu C. \tag{7.74}$$

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.

Appendix A

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.

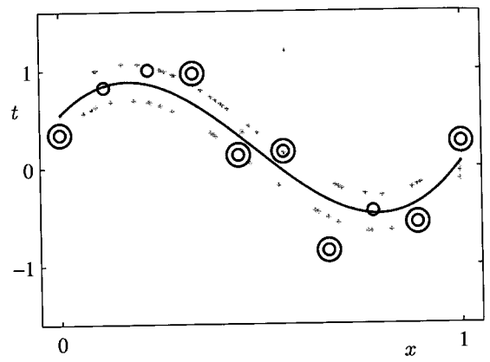


Figure 7.8 Illustration of the ν-SVM for regression applied to the sinusoidal synthetic data set using Gaussian kernels. The predicted regression curve is shown by the red line, and the ϵ-insensitive tube corresponds to the shaded region. Also, the data points are shown in green, and those with support 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

$$\mathbb{E}_{\mathbf{x},\mathbf{t}} \left[I \left(\mathbf{f}(\mathbf{x}; \mathcal{D}) \neq \mathbf{t} \right) \right] < \epsilon \tag{7.75}$$

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 $p(\mathbf{x}, \mathbf{t})$, so long as both the training and the test examples are drawn (independently) from the same distribution, and for *any* choice for the function $\mathbf{f}(\mathbf{x})$ 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 $p(\mathbf{x}, \mathbf{t})$, 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

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}), \beta^{-1}) \tag{7.76}$$

where $\beta = \sigma^{-2}$ is the noise precision (inverse noise variance), and the mean is given by a linear model of the form

$$y(\mathbf{x}) = \sum_{i=1}^{M} w_i \phi_i(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}) \tag{7.77}$$

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

$$y(\mathbf{x}) = \sum_{n=1}^{N} w_n k(\mathbf{x}, \mathbf{x}_n) + b \tag{7.78}$$

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 $\mathbf{x}_n^{\mathrm{T}}$ with $n = 1, \ldots, N$. The corresponding target values are given by $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$. Thus, the likelihood function is given by

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta^{-1}). \tag{7.79}$$

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

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{i=1}^{M} \mathcal{N}(w_i|0, \alpha_i^{-1}) \tag{7.80}$$

where α_i represents the precision of the corresponding parameter w_i, and $\boldsymbol{\alpha}$ denotes $(\alpha_1, \ldots, \alpha_M)^{\mathrm{T}}$. 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

$$p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \boldsymbol{\alpha}, \beta) = \mathcal{N}(\mathbf{w}|\mathbf{m}, \boldsymbol{\Sigma}) \tag{7.81}$$

where the mean and covariance are given by

$$\mathbf{m} = \beta \boldsymbol{\Sigma} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{t} \tag{7.82}$$

$$\boldsymbol{\Sigma} = \left(\mathbf{A} + \beta \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi}\right)^{-1} \tag{7.83}$$

where $\boldsymbol{\Phi}$ is the $N \times M$ design matrix with elements $\Phi_{ni} = \phi_i(\mathbf{x}_n)$, and $\mathbf{A} = \mathrm{diag}(\alpha_i)$. 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 $k(\mathbf{x}_n, \mathbf{x}_m)$.

Section 3.5 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

$$p(\mathbf{t}|\mathbf{X}, \boldsymbol{\alpha}, \beta) = \int p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) p(\mathbf{w}|\boldsymbol{\alpha}) \, \mathrm{d}\mathbf{w}. \tag{7.84}$$

Exercise 7.10 Because this represents the convolution of two Gaussians, it is readily evaluated to give the log marginal likelihood in the form

$$\begin{aligned} \ln p(\mathbf{t}|\mathbf{X}, \boldsymbol{\alpha}, \beta) &= \ln \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C}) \\ &= -\frac{1}{2}\left\{N \ln(2\pi) + \ln|\mathbf{C}| + \mathbf{t}^{\mathrm{T}} \mathbf{C}^{-1} \mathbf{t}\right\} \end{aligned} \tag{7.85}$$

where $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$, and we have defined the $N \times N$ matrix $\mathbf{C}$ given by

$$\mathbf{C} = \beta^{-1} \mathbf{I} + \boldsymbol{\Phi} \mathbf{A}^{-1} \boldsymbol{\Phi}^{\mathrm{T}}. \tag{7.86}$$

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 *Exercise 7.12*

$$\alpha_i^{\text{new}} = \frac{\gamma_i}{m_i^2} \tag{7.87}$$

$$(\beta^{\text{new}})^{-1} = \frac{\|\mathbf{t} - \boldsymbol{\Phi}\mathbf{m}\|^2}{N - \sum_i \gamma_i} \tag{7.88}$$

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*

$$\gamma_i = 1 - \alpha_i \Sigma_{ii} \tag{7.89}$$

in which Σ_{ii} is the i^{th} diagonal component of the posterior covariance $\boldsymbol{\Sigma}$ given by (7.83). Learning therefore proceeds by choosing initial values for $\boldsymbol{\alpha}$ 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

Exercise 9.23 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

Section 7.2.2 $\{\alpha_i\}$ 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 $\boldsymbol{\alpha}^\star$ and $\beta^\star$ for the hyperparameters that maximize the marginal likelihood, we can evaluate the predictive distribution over t for a new

Exercise 7.14 input $\mathbf{x}$. Using (7.76) and (7.81), this is given by

$$\begin{aligned} p(t|\mathbf{x}, \mathbf{X}, \mathbf{t}, \boldsymbol{\alpha}^\star, \beta^\star) &= \int p(t|\mathbf{x}, \mathbf{w}, \beta^\star) p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \boldsymbol{\alpha}^\star, \beta^\star)\, \mathrm{d}\mathbf{w} \\ &= \mathcal{N}\left(t|\mathbf{m}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}), \sigma^2(\mathbf{x})\right). \end{aligned} \tag{7.90}$$

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

$$\sigma^2(\mathbf{x}) = (\beta^\star)^{-1} + \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{\phi}(\mathbf{x}) \tag{7.91}$$

where $\boldsymbol{\Sigma}$ is given by (7.83) in which $\boldsymbol{\alpha}$ and β are set to their optimized values $\boldsymbol{\alpha}^\star$ and $\beta^\star$. 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

Section 6.4.2 is undesirable. The predictive distribution in Gaussian process regression does not

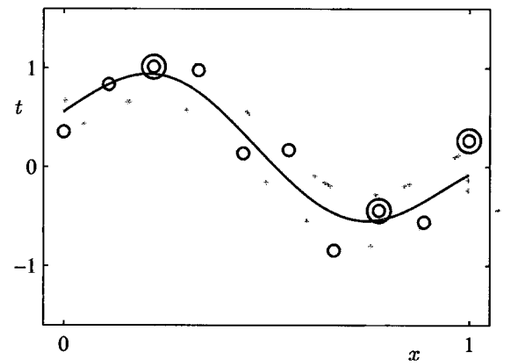


Figure 7.9 Illustration of RVM regression using the same data set, and the same Gaussian kernel functions, as used in Figure 7.8 for the ν-SVM regression model. The mean of the predictive distribution for the RVM is shown by the red line, and the one standard-deviation predictive distribution is shown 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

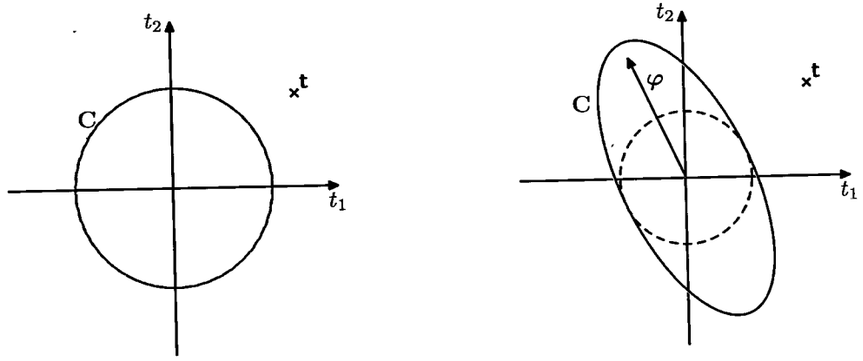


Figure 7.10 Illustration of the mechanism for sparsity in a Bayesian linear regression model, showing a training set vector of target values given by $\mathbf{t} = (t_1, t_2)^{\mathrm{T}}$, indicated by the cross, for a model with one basis vector $\boldsymbol{\varphi} = (\phi(\mathbf{x}_1), \phi(\mathbf{x}_2))^{\mathrm{T}}$, 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 contrition 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

$$\mathbf{C} = \frac{1}{\beta}\mathbf{I} + \frac{1}{\alpha}\boldsymbol{\varphi}\boldsymbol{\varphi}^{\mathrm{T}} \tag{7.92}$$

where $\boldsymbol{\varphi}$ denotes the N-dimensional vector $(\phi(\mathbf{x}_1), \phi(\mathbf{x}_2))^{\mathrm{T}}$, and similarly $\mathbf{t} = (t_1, t_2)^{\mathrm{T}}$. 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 $\alpha^\star$ and $\beta^\star$ by maximizing the marginal likelihood. We see from Figure 7.10 that, if there is a poor alignment between the direction of $\boldsymbol{\varphi}$ 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 $\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_M$ 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

$$\begin{aligned} \mathbf{C} &= \beta^{-1}\mathbf{I} + \sum_{j \neq i} \alpha_j^{-1}\boldsymbol{\varphi}_j\boldsymbol{\varphi}_j^{\mathrm{T}} + \alpha_i^{-1}\boldsymbol{\varphi}_i\boldsymbol{\varphi}_i^{\mathrm{T}} \\ &= \mathbf{C}_{-i} + \alpha_i^{-1}\boldsymbol{\varphi}_i\boldsymbol{\varphi}_i^{\mathrm{T}} \end{aligned} \tag{7.93}$$

where $\boldsymbol{\varphi}_i$ denotes the i^{th} column of $\boldsymbol{\Phi}$, in other words the N-dimensional vector with elements $(\phi_i(\mathbf{x}_1), \ldots, \phi_i(\mathbf{x}_N))$, in contrast to $\boldsymbol{\phi}_n$, which denotes the n^{th} row of $\boldsymbol{\Phi}$. 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

$$|\mathbf{C}| = |\mathbf{C}_{-i}||1 + \alpha_i^{-1}\boldsymbol{\varphi}_i^{\mathrm{T}}\mathbf{C}_{-i}^{-1}\boldsymbol{\varphi}_i| \tag{7.94}$$

$$\mathbf{C}^{-1} = \mathbf{C}_{-i}^{-1} - \frac{\mathbf{C}_{-i}^{-1}\boldsymbol{\varphi}_i\boldsymbol{\varphi}_i^{\mathrm{T}}\mathbf{C}_{-i}^{-1}}{\alpha_i + \boldsymbol{\varphi}_i^{\mathrm{T}}\mathbf{C}_{-i}^{-1}\boldsymbol{\varphi}_i}. \tag{7.95}$$

Exercise 7.15

Using these results, we can then write the log marginal likelihood function (7.85) in the form

$$L(\boldsymbol{\alpha}) = L(\boldsymbol{\alpha}_{-i}) + \lambda(\alpha_i) \tag{7.96}$$

where $L(\boldsymbol{\alpha}_{-i})$ is simply the log marginal likelihood with basis function $\boldsymbol{\varphi}_i$ omitted, and the quantity $\lambda(\alpha_i)$ is defined by

$$\lambda(\alpha_i) = \frac{1}{2}\left[\ln \alpha_i - \ln(\alpha_i + s_i) + \frac{q_i^2}{\alpha_i + s_i}\right] \tag{7.97}$$

and contains all of the dependence on α_i. Here we have introduced the two quantities

$$s_i = \boldsymbol{\varphi}_i^{\mathrm{T}}\mathbf{C}_{-i}^{-1}\boldsymbol{\varphi}_i \tag{7.98}$$

$$q_i = \boldsymbol{\varphi}_i^{\mathrm{T}}\mathbf{C}_{-i}^{-1}\mathbf{t}. \tag{7.99}$$

Here s_i is called the *sparsity* and q_i is known as the *quality* of $\boldsymbol{\varphi}_i$, and as we shall see, a large value of s_i relative to the value of q_i means that the basis function $\boldsymbol{\varphi}_i$

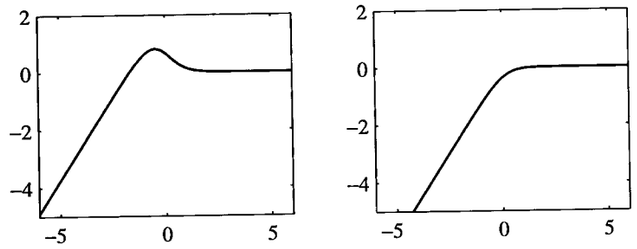


Figure 7.11 Plots of the log marginal likelihood $\lambda(\alpha_i)$ versus $\ln \alpha_i$ showing on the left, the single maximum at a finite α_i for $q_i^2 = 4$ and $s_i = 1$ (so that $q_i^2 > s_i$) and on the right, the maximum at $\alpha_i = \infty$ for $q_i^2 = 1$ and $s_i = 2$ (so that $q_i^2 < s_i$).

is more likely to be pruned from the model. The 'sparsity' measures the extent to which basis function $\boldsymbol{\varphi}_i$ overlaps with the other basis vectors in the model, and the 'quality' represents a measure of the alignment of the basis vector $\boldsymbol{\varphi}_n$ with the error between the training set values $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$ and the vector $\mathbf{y}_{-i}$ of predictions that would result from the model with the vector $\boldsymbol{\varphi}_i$ excluded (Tipping and Faul, 2003).

The stationary points of the marginal likelihood with respect to α_i occur when the derivative

$$\frac{\mathrm{d}\lambda(\alpha_i)}{\mathrm{d}\alpha_i} = \frac{\alpha_i^{-1} s_i^2 - (q_i^2 - s_i)}{2(\alpha_i + s_i)^2} \tag{7.100}$$

is equal to zero. There are two possible forms for the solution. Recalling that $\alpha_i \geqslant 0$, we see that if $q_i^2 < s_i$, then $\alpha_i \to \infty$ provides a solution. Conversely, if $q_i^2 > s_i$, we can solve for α_i to obtain

$$\alpha_i = \frac{s_i^2}{q_i^2 - s_i}. \tag{7.101}$$

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 $\lambda(\alpha_i)$.

Exercise 7.16

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 $\boldsymbol{\varphi}_1$, with hyperparameter α_1 set using (7.101), with the remaining hyperparameters α_j for $j \neq i$ initialized to infinity, so that only $\boldsymbol{\varphi}_1$ is included in the model.

3. Evaluate $\boldsymbol{\Sigma}$ and $\mathbf{m}$, along with q_i and s_i for all basis functions.
4. Select a candidate basis function $\boldsymbol{\varphi}_i$.
5. If $q_i^2 > s_i$, and $\alpha_i < \infty$, so that the basis vector $\boldsymbol{\varphi}_i$ 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 $\boldsymbol{\varphi}_i$ to the model, and evaluate hyperparameter α_i using (7.101).
7. If $q_i^2 \leqslant s_i$, and $\alpha_i < \infty$ then remove basis function $\boldsymbol{\varphi}_i$ 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 $q_i^2 \leqslant s_i$ and $\alpha_i = \infty$, then the basis function $\boldsymbol{\varphi}_i$ is already excluded from the model and no action is required.

In practice, it is convenient to evaluate the quantities

$$Q_i = \boldsymbol{\varphi}_i^{\mathrm{T}} \mathbf{C}^{-1} \mathbf{t} \tag{7.102}$$

$$S_i = \boldsymbol{\varphi}_i^{\mathrm{T}} \mathbf{C}^{-1} \boldsymbol{\varphi}_i. \tag{7.103}$$

The quality and sparseness variables can then be expressed in the form

$$q_i = \frac{\alpha_i Q_i}{\alpha_i - S_i} \tag{7.104}$$

$$s_i = \frac{\alpha_i S_i}{\alpha_i - S_i}. \tag{7.105}$$

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

$$Q_i = \beta \boldsymbol{\varphi}_i^{\mathrm{T}} \mathbf{t} - \beta^2 \boldsymbol{\varphi}_i^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Sigma} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{t} \tag{7.106}$$

$$S_i = \beta \boldsymbol{\varphi}_i^{\mathrm{T}} \boldsymbol{\varphi}_i - \beta^2 \boldsymbol{\varphi}_i^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Sigma} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\varphi}_i \tag{7.107}$$

where $\boldsymbol{\Phi}$ and $\boldsymbol{\Sigma}$ 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

$$y(\mathbf{x}, \mathbf{w}) = \sigma\left(\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})\right) \tag{7.108}$$

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.

Section 4.4

We begin by initializing the hyperparameter vector $\boldsymbol{\alpha}$. For this given value of $\boldsymbol{\alpha}$, 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 $\boldsymbol{\alpha}$, and the process is repeated until convergence.

Let us consider the Laplace approximation for this model in more detail. For a fixed value of $\boldsymbol{\alpha}$, the mode of the posterior distribution over $\mathbf{w}$ is obtained by maximizing

$$\begin{aligned}\ln p(\mathbf{w}|\mathbf{t},\boldsymbol{\alpha}) &= \ln\{p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})\} - \ln p(\mathbf{t}|\boldsymbol{\alpha}) \\ &= \sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} - \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{A}\mathbf{w} + \text{const} \quad (7.109)\end{aligned}$$

where $\mathbf{A} = \mathrm{diag}(\alpha_i)$. 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

Exercise 7.18

$$\nabla \ln p(\mathbf{w}|\mathbf{t},\boldsymbol{\alpha}) = \boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{t}-\mathbf{y}) - \mathbf{A}\mathbf{w} \quad (7.110)$$

$$\nabla\nabla \ln p(\mathbf{w}|\mathbf{t},\boldsymbol{\alpha}) = -\left(\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{B}\boldsymbol{\Phi} + \mathbf{A}\right) \quad (7.111)$$

where $\mathbf{B}$ is an $N \times N$ diagonal matrix with elements $b_n = y_n(1-y_n)$, the vector $\mathbf{y} = (y_1, \ldots, y_N)^{\mathrm{T}}$, and $\boldsymbol{\Phi}$ 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

$$\mathbf{w}^\star = \mathbf{A}^{-1}\boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{t}-\mathbf{y}) \quad (7.112)$$

$$\boldsymbol{\Sigma} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{B}\boldsymbol{\Phi} + \mathbf{A}\right)^{-1}. \quad (7.113)$$

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 approximation, we have

$$\begin{aligned}p(\mathbf{t}|\boldsymbol{\alpha}) &= \int p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})\,\mathrm{d}\mathbf{w} \\ &\simeq p(\mathbf{t}|\mathbf{w}^\star)p(\mathbf{w}^\star|\boldsymbol{\alpha})(2\pi)^{M/2}|\boldsymbol{\Sigma}|^{1/2}. \quad (7.114)\end{aligned}$$

If we substitute for $p(\mathbf{t}|\mathbf{w}^\star)$ and $p(\mathbf{w}^\star|\boldsymbol{\alpha})$ and then set the derivative of the marginal likelihood with respect to α_i equal to zero, we obtain

Exercise 7.19

$$-\frac{1}{2}(w_i^\star)^2 + \frac{1}{2\alpha_i} - \frac{1}{2}\Sigma_{ii} = 0. \quad (7.115)$$

Defining $\gamma_i = 1 - \alpha_i\Sigma_{ii}$ and rearranging then gives

$$\alpha_i^{\text{new}} = \frac{\gamma_i}{(w_i^\star)^2} \quad (7.116)$$

which is identical to the re-estimation formula (7.87) obtained for the regression RVM.

If we define

$$\widehat{\mathbf{t}} = \boldsymbol{\Phi}\mathbf{w}^\star + \mathbf{B}^{-1}(\mathbf{t}-\mathbf{y}) \quad (7.117)$$

we can write the approximate log marginal likelihood in the form

$$\ln p(\mathbf{t}|\boldsymbol{\alpha},\beta) = -\frac{1}{2}\left\{N\ln(2\pi) + \ln|\mathbf{C}| + \widehat{\mathbf{t}}^{\mathrm{T}}\mathbf{C}^{-1}\widehat{\mathbf{t}}\right\} \quad (7.118)$$

where

$$\mathbf{C} = \mathbf{B} + \boldsymbol{\Phi}\mathbf{A}\boldsymbol{\Phi}^{\mathrm{T}}. \quad (7.119)$$

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.

Appendix A

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 $\boldsymbol{\varphi}_i$ 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).

Section 13.3

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

$$a_k = \mathbf{w}_k^{\mathrm{T}}\mathbf{x} \quad (7.120)$$

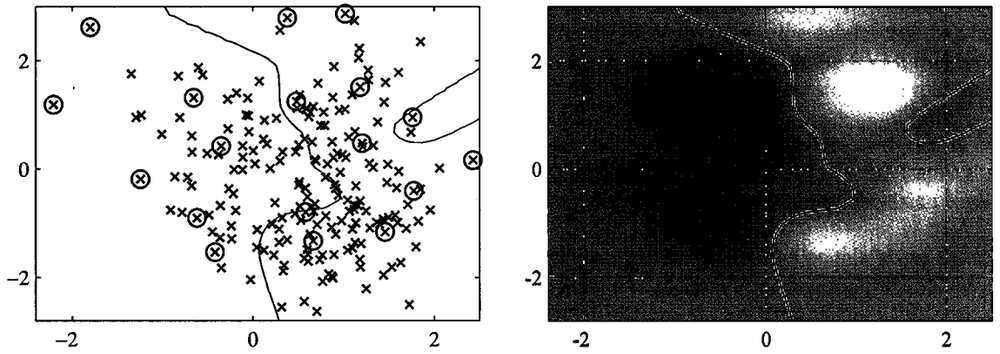


Figure 7.12 Example of the relevance vector machine applied to a synthetic data set, in which the left-hand plot shows the decision boundary and the data points, with the relevance vectors indicated by circles. Comparison with the results shown in Figure 7.4 for the corresponding support vector machine 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

$$y_k(\mathbf{x}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}. \tag{7.121}$$

The log likelihood function is then given by

$$\ln p(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}} \tag{7.122}$$

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 $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{x}'$, 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 $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}')$, that the classification is based on the closest mean in the feature space $\boldsymbol{\phi}(\mathbf{x})$.

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

$$\frac{1}{\rho^2} = \sum_{n=1}^{N} a_n \tag{7.123}$$

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

$$\frac{1}{\rho^2} = 2\widetilde{L}(\mathbf{a}) \tag{7.124}$$

where $\widetilde{L}(\mathbf{a})$ is defined by (7.10). Similarly, show that

$$\frac{1}{\rho^2} = \|\mathbf{w}\|^2. \tag{7.125}$$

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 $\widehat{\xi}_n$ to zero and then back substituting to eliminate the corresponding variables, show that the dual Lagrangian is given by (7.61).

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

$$p(\mathcal{D}) \simeq \prod_j Z_j. \tag{10.243}$$

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 $\widetilde{f}_0(\boldsymbol{\theta})$ is initialized to be $f_0(\boldsymbol{\theta})$, then an EP update to refine $\widetilde{f}_0(\boldsymbol{\theta})$ leaves $\widetilde{f}_0(\boldsymbol{\theta})$ 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).

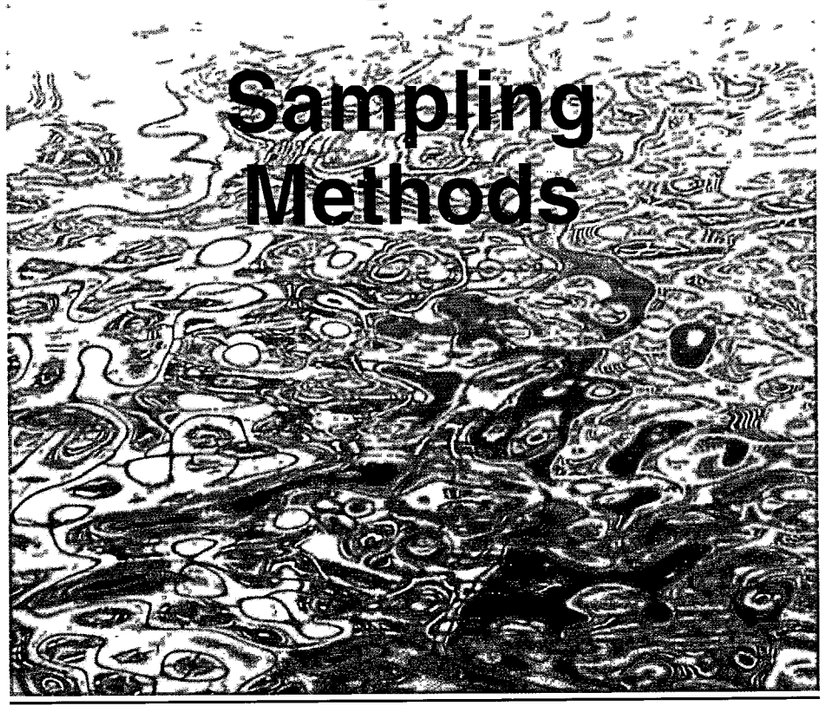
10.39 (⋆⋆⋆) Show that the mean and variance of $q^{\text{new}}(\boldsymbol{\theta})$ for EP applied to the clutter problem are given by (10.217) and (10.218). To do this, first prove the following results for the expectations of $\boldsymbol{\theta}$ and $\boldsymbol{\theta}\boldsymbol{\theta}^{\mathrm{T}}$ under $q^{\text{new}}(\boldsymbol{\theta})$

$$\mathbb{E}[\boldsymbol{\theta}] = \mathbf{m}^{\backslash n} + v^{\backslash n}\nabla_{\mathbf{m}^{\backslash n}} \ln Z_n \tag{10.244}$$

$$\mathbb{E}[\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\theta}] = 2(v^{\backslash n})^2\nabla_{v^{\backslash n}} \ln Z_n + 2\mathbb{E}[\boldsymbol{\theta}]^{\mathrm{T}}\mathbf{m}^{\backslash n} - \|\mathbf{m}^{\backslash n}\|^2 \tag{10.245}$$

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

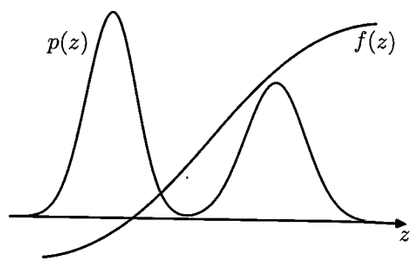


Figure 11.1 Schematic illustration of a function $f(z)$ whose expectation is to be evaluated with respect to a distribution $p(z)$.

variables, we wish to evaluate the expectation

$$\mathbb{E}[f] = \int f(\mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z} \tag{11.1}$$

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 $l = 1, \ldots, L$) drawn independently from the distribution $p(\mathbf{z})$. This allows the expectation (11.1) to be approximated by a finite sum

$$\widehat{f} = \frac{1}{L}\sum_{l=1}^{L} f(\mathbf{z}^{(l)}). \tag{11.2}$$

As long as the samples $\mathbf{z}^{(l)}$ are drawn from the distribution $p(\mathbf{z})$, then $\mathbb{E}[\widehat{f}] = \mathbb{E}[f]$ and so the estimator $\widehat{f}$ has the correct mean. The variance of the estimator is given by

Exercise 11.1

$$\mathrm{var}[\widehat{f}] = \frac{1}{L}\mathbb{E}\left[(f - \mathbb{E}[f])^2\right] \tag{11.3}$$

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

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

$$p(\mathbf{z}) = \prod_{i=1}^{M} p(\mathbf{z}_i|\mathrm{pa}_i) \tag{11.4}$$

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 $\mathbf{z}_1, \ldots, \mathbf{z}_M$ sampling from the conditional distributions $p(\mathbf{z}_i|\mathrm{pa}_i)$. 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

$$p(y) = p(z)\left|\frac{dz}{dy}\right| \tag{11.5}$$

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

$$z = h(y) \equiv \int_{-\infty}^{y} p(\widehat{y})\,\mathrm{d}\widehat{y} \tag{11.6}$$

Exercise 11.2

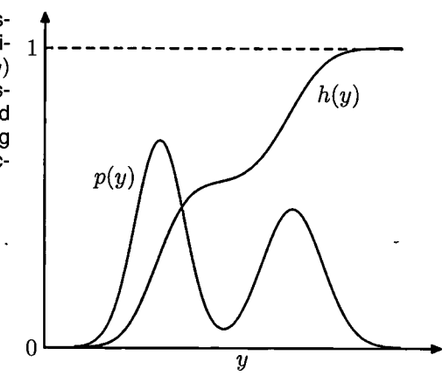
which is the indefinite integral of $p(y)$. Thus, $y = h^{-1}(z)$, and so we have to transform the uniformly distributed random numbers using a function which is the inverse of the indefinite integral of the desired distribution. This is illustrated in Figure 11.2.

Consider for example the *exponential distribution*

$$p(y) = \lambda \exp(-\lambda y) \tag{11.7}$$

where $0 \leqslant y < \infty$. 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.

Figure 11.2 Geometrical interpretation of the transformation method for generating nonuniformly distributed random numbers. $h(y)$ 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

$$p(y) = \frac{1}{\pi}\frac{1}{1+y^2}. \tag{11.8}$$

Exercise 11.3

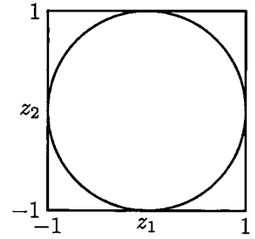
In this case, the inverse of the indefinite integral can be expressed in terms of the 'tan' function.

The generalization to multiple variables is straightforward and involves the Jacobian of the change of variables, so that

$$p(y_1, \ldots, y_M) = p(z_1, \ldots, z_M)\left|\frac{\partial(z_1, \ldots, z_M)}{\partial(y_1, \ldots, y_M)}\right|. \tag{11.9}$$

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 $z \to 2z - 1$. Next we discard each pair unless it satisfies $z_1^2 + z_2^2 \leqslant 1$. 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.



$$y_1 = z_1\left(\frac{-2\ln z_1}{r^2}\right)^{1/2} \tag{11.10}$$

$$y_2 = z_2\left(\frac{-2\ln z_2}{r^2}\right)^{1/2} \tag{11.11}$$

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

$$p(y_1, y_2) = p(z_1, z_2)\left|\frac{\partial(z_1, z_2)}{\partial(y_1, y_2)}\right| = \left[\frac{1}{\sqrt{2\pi}}\exp(-y_1^2/2)\right]\left[\frac{1}{\sqrt{2\pi}}\exp(-y_2^2/2)\right] \tag{11.12}$$

and so y_1 and y_2 are independent and each has a Gaussian distribution with zero mean and unit variance.

If y has a Gaussian distribution with zero mean and unit variance, then $\sigma y + \mu$ will have a Gaussian distribution with mean μ and variance σ^2. To generate vector-valued variables having a multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, we can make use of the *Cholesky decomposition*, which takes the form $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$ (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 $\mathbf{y} = \boldsymbol{\mu} + \mathbf{L}\mathbf{z}$ will have mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.

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

$$p(z) = \frac{1}{Z_p}\widetilde{p}(z) \tag{11.13}$$

where $\widetilde{p}(z)$ 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.

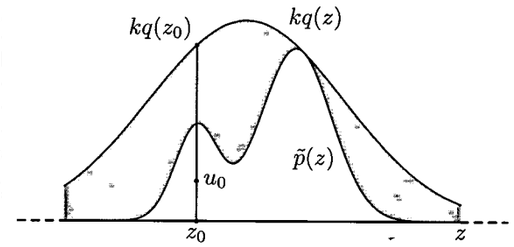


Figure 11.4 In the rejection sampling method, samples are drawn from a simple distribution $q(z)$ and rejected if they fall in the grey area between the unnormalized distribution $\widetilde{p}(z)$ and the scaled distribution $kq(z)$. The resulting samples are distributed according to $p(z)$, which is the normalized version of $\widetilde{p}(z)$.

We next introduce a constant k whose value is chosen such that $kq(z) \geqslant \widetilde{p}(z)$ 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 $u_0 > \widetilde{p}(z_0)$ 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 $\widetilde{p}(z)$, and hence the corresponding z values are distributed according to $p(z)$, as desired.

Exercise 11.6

The original values of z are generated from the distribution $q(z)$, and these samples are then accepted with probability $\widetilde{p}(z)/kq(z)$, and so the probability that a sample will be accepted is given by

$$p(\text{accept}) = \int \{\widetilde{p}(z)/kq(z)\}\, q(z)\, \mathrm{d}z = \frac{1}{k}\int \widetilde{p}(z)\, \mathrm{d}z. \tag{11.14}$$

Thus the fraction of points that are rejected by this method depends on the ratio of the area under the unnormalized distribution $\widetilde{p}(z)$ 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 $\widetilde{p}(z)$.

As an illustration of the use of rejection sampling, consider the task of sampling from the gamma distribution

$$\mathrm{Gam}(z|a, b) = \frac{b^a z^{a-1}\exp(-bz)}{\Gamma(a)} \tag{11.15}$$

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.

Exercise 11.7

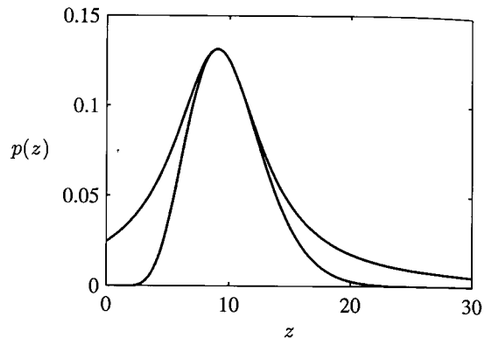


Figure 11.5 Plot showing the gamma distribution given by (11.15) as the green curve, with a scaled Cauchy proposal distribution shown by the red curve. Samples from the gamma distribution can be obtained by sampling from the Cauchy and then applying the rejection sampling criterion.

$$q(z) = \frac{k}{1 + (z - c)^2/b^2}. \tag{11.16}$$

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 $kq(z) \geqslant \widetilde{p}(z)$. 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. *Exercise 11.9* This is straightforward because the log of the envelope distribution is a succession

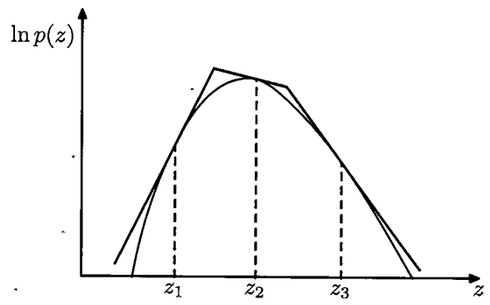


Figure 11.6 In the case of distributions that are log concave, an envelope function for use in rejection sampling can be constructed using the tangent lines computed at a set of grid points. If a sample point is rejected, it is added to the set of grid points and used to refine the envelope distribution.

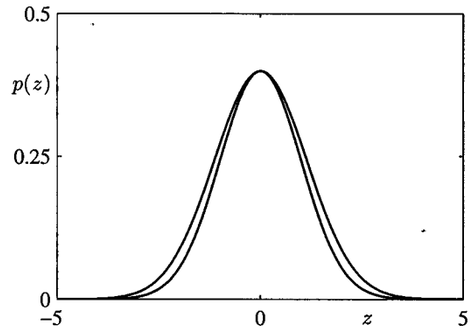


Figure 11.7 Illustrative example of rejection sampling involving sampling from a Gaussian distribution $p(z)$ shown by the green curve, by using rejection sampling from a proposal distribution $q(z)$ that is also Gaussian and whose scaled version $kq(z)$ is shown by the red curve.

of linear functions, and hence the envelope distribution itself comprises a piecewise exponential distribution of the form

$$q(z) = k_i \lambda_i \exp\{-\lambda_i(z - z_{i-1})\} \qquad z_{i-1} < z \leqslant z_i. \tag{11.17}$$

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 $\sigma_q^2 \geqslant \sigma_p^2$ in order that there exists a k such that $kq(z) \geqslant p(z)$. 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.

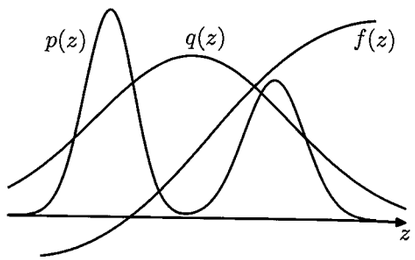


Figure 11.8 Importance sampling addresses the problem of evaluating the expectation of a function $f(z)$ with respect to a distribution $p(z)$ from which it is difficult to draw samples directly. Instead, samples $\{z^{(l)}\}$ are drawn from a simpler distribution $q(z)$, and the corresponding terms 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

$$\mathbb{E}[f] \simeq \sum_{l=1}^{L} p(\mathbf{z}^{(l)}) f(\mathbf{z}^{(l)}). \tag{11.18}$$

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 $q(\mathbf{z})$

$$\begin{aligned}\mathbb{E}[f] &= \int f(\mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z} \\ &= \int f(\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})\,\mathrm{d}\mathbf{z} \\ &\simeq \frac{1}{L}\sum_{l=1}^{L}\frac{p(\mathbf{z}^{(l)})}{q(\mathbf{z}^{(l)})}f(\mathbf{z}^{(l)}).\end{aligned} \tag{11.19}$$

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 $p(\mathbf{z}) = \widetilde{p}(\mathbf{z})/Z_p$ where $\widetilde{p}(\mathbf{z})$ can be evaluated easily, whereas Z_p is unknown. Similarly, we may wish to use an importance sampling distribution $q(\mathbf{z}) = \widetilde{q}(\mathbf{z})/Z_q$, which has the same property. We then have

$$\begin{aligned}\mathbb{E}[f] &= \int f(\mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z} \\ &= \frac{Z_q}{Z_p}\int f(\mathbf{z})\frac{\widetilde{p}(\mathbf{z})}{\widetilde{q}(\mathbf{z})}q(\mathbf{z})\,\mathrm{d}\mathbf{z} \\ &\simeq \frac{Z_q}{Z_p}\frac{1}{L}\sum_{l=1}^{L}\widetilde{r}_l f(\mathbf{z}^{(l)}).\end{aligned} \tag{11.20}$$

where $\widetilde{r}_l = \widetilde{p}(\mathbf{z}^{(l)})/\widetilde{q}(\mathbf{z}^{(l)})$. We can use the same sample set to evaluate the ratio Z_p/Z_q with the result

$$\begin{aligned}\frac{Z_p}{Z_q} &= \frac{1}{Z_q}\int \widetilde{p}(\mathbf{z})\,\mathrm{d}\mathbf{z} = \int \frac{\widetilde{p}(\mathbf{z})}{\widetilde{q}(\mathbf{z})}q(\mathbf{z})\,\mathrm{d}\mathbf{z} \\ &\simeq \frac{1}{L}\sum_{l=1}^{L}\widetilde{r}_l\end{aligned} \tag{11.21}$$

and hence

$$\mathbb{E}[f] \simeq \sum_{l=1}^{L} w_l f(\mathbf{z}^{(l)}) \tag{11.22}$$

where we have defined

$$w_l = \frac{\widetilde{r}_l}{\sum_m \widetilde{r}_m} = \frac{\widetilde{p}(\mathbf{z}^{(l)})/q(\mathbf{z}^{(l)})}{\sum_m \widetilde{p}(\mathbf{z}^{(m)})/q(\mathbf{z}^{(m)})}. \tag{11.23}$$

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 $\widetilde{q}(\mathbf{z})$ is uniform over the possible choices for $\mathbf{z}$, and that $\widetilde{p}(\mathbf{z}|\mathbf{x}) = \widetilde{p}(\mathbf{z})$, 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 $p(\mathbf{z}_i|\mathrm{pa}_i)$ 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

$$r(\mathbf{z}) = \prod_{\mathbf{z}_i \notin \mathbf{e}} \frac{p(\mathbf{z}_i|\mathrm{pa}_i)}{p(\mathbf{z}_i|\mathrm{pa}_i)} \prod_{\mathbf{z}_i \in \mathbf{e}} \frac{p(\mathbf{z}_i|\mathrm{pa}_i)}{1} = \prod_{\mathbf{z}_i \in \mathbf{e}} p(\mathbf{z}_i|\mathrm{pa}_i). \tag{11.24}$$

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 $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(L)}$ are drawn from $q(\mathbf{z})$. Then in the second stage, weights $w_1, \ldots, w_L$ are constructed using (11.23). Finally, a second set of L samples is drawn from the discrete distribution $(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(L)})$ with probabilities given by the weights $(w_1, \ldots, w_L)$.

The resulting L samples are only approximately distributed according to $p(\mathbf{z})$, but the distribution becomes correct in the limit $L \to \infty$. To see this, consider the univariate case, and note that the cumulative distribution of the resampled values is given by

$$\begin{aligned} p(z \leqslant a) &= \sum_{l: z^{(l)} \leqslant a} w_l \\ &= \frac{\sum_l I(z^{(l)} \leqslant a)\widetilde{p}(z^{(l)})/q(z^{(l)})}{\sum_l \widetilde{p}(z^{(l)})/q(z^{(l)})} \end{aligned} \tag{11.25}$$

where $I(.)$ is the indicator function (which equals 1 if its argument is true and 0 otherwise). Taking the limit $L \to \infty$, and assuming suitable regularity of the distributions, we can replace the sums by integrals weighted according to the original sampling distribution $q(z)$

$$\begin{aligned} p(z \leqslant a) &= \frac{\int I(z \leqslant a)\,\{\widetilde{p}(z)/q(z)\}\,q(z)\,\mathrm{d}z}{\int \{\widetilde{p}(z)/q(z)\}\,q(z)\,\mathrm{d}z} \\ &= \frac{\int I(z \leqslant a)\widetilde{p}(z)\,\mathrm{d}z}{\int \widetilde{p}(z)\,\mathrm{d}z} \\ &= \int I(z \leqslant a)p(z)\,\mathrm{d}z \end{aligned} \tag{11.26}$$

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 $(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(L)})$ 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

$$\begin{aligned}\mathbb{E}[f(\mathbf{z})] &= \int f(\mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z} \\ &= \frac{\int f(\mathbf{z})[\widetilde{p}(\mathbf{z})/q(\mathbf{z})]q(\mathbf{z})\,\mathrm{d}\mathbf{z}}{\int [\widetilde{p}(\mathbf{z})/q(\mathbf{z})]q(\mathbf{z})\,\mathrm{d}\mathbf{z}} \\ &\simeq \sum_{l=1}^{L} w_l f(\mathbf{z}_l). \end{aligned} \tag{11.27}$$

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

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \int p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{Z}, \mathbf{X}|\boldsymbol{\theta})\,\mathrm{d}\mathbf{Z}. \tag{11.28}$$

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 $\boldsymbol{\theta}$. In principle, we would like to draw samples from the joint posterior $p(\boldsymbol{\theta}, \mathbf{Z}|\mathbf{X})$, but we shall suppose that this is computationally difficult. Suppose further that it is relatively straightforward to sample from the complete-data parameter posterior $p(\boldsymbol{\theta}|\mathbf{Z}, \mathbf{X})$. 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

$$p(\mathbf{Z}|\mathbf{X}) = \int p(\mathbf{Z}|\boldsymbol{\theta}, \mathbf{X})p(\boldsymbol{\theta}|\mathbf{X})\,\mathrm{d}\boldsymbol{\theta} \tag{11.30}$$

and hence for $l = 1, \ldots, L$ we first draw a sample $\boldsymbol{\theta}^{(l)}$ from the current estimate for $p(\boldsymbol{\theta}|\mathbf{X})$, and then use this to draw a sample $\mathbf{Z}^{(l)}$ from $p(\mathbf{Z}|\boldsymbol{\theta}^{(l)}, \mathbf{X})$.

P-step. Given the relation

$$p(\boldsymbol{\theta}|\mathbf{X}) = \int p(\boldsymbol{\theta}|\mathbf{Z}, \mathbf{X})p(\mathbf{Z}|\mathbf{X})\,\mathrm{d}\mathbf{Z} \tag{11.31}$$

we use the samples $\{\mathbf{Z}^{(l)}\}$ obtained from the I-step to compute a revised estimate of the posterior distribution over $\boldsymbol{\theta}$ given by

$$p(\boldsymbol{\theta}|\mathbf{X}) \simeq \frac{1}{L}\sum_{l=1}^{L} p(\boldsymbol{\theta}|\mathbf{Z}^{(l)}, \mathbf{X}). \tag{11.32}$$

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 $\boldsymbol{\theta}$ 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 $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots$ forms a Markov chain. Again, if we write $p(\mathbf{z}) = \widetilde{p}(\mathbf{z})/Z_p$, we will assume that $\widetilde{p}(\mathbf{z})$ 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 $\mathbf{z}^\star$ from the proposal distribution and then accept the sample according to an appropriate criterion.

Section 11.2.1

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

$$A(\mathbf{z}^\star, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{\widetilde{p}(\mathbf{z}^\star)}{\widetilde{p}(\mathbf{z}^{(\tau)})}\right). \tag{11.33}$$

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 $A(\mathbf{z}^\star, \mathbf{z}^{(\tau)}) > u$. Note that if the step from $\mathbf{z}^{(\tau)}$ to $\mathbf{z}^\star$ 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 $\mathbf{z}^{(\tau+1)} = \mathbf{z}^\star$, otherwise the candidate point $\mathbf{z}^\star$ 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 $\tau \to \infty$. It should be emphasized, however, that the sequence $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots$ 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

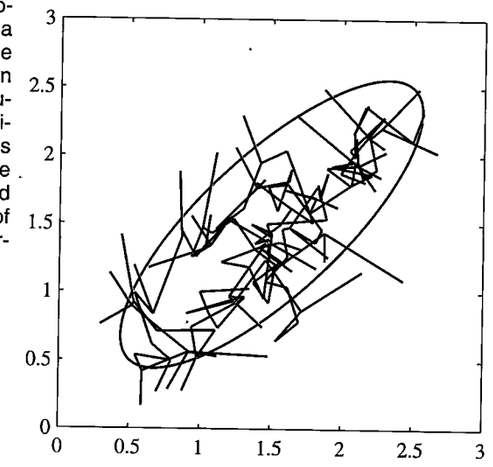


Figure 11.9 A simple illustration using Metropolis algorithm to sample from a Gaussian distribution whose one standard-deviation contour is shown by the ellipse. The proposal distribution is an isotropic Gaussian distribution whose standard deviation is 0.2. Steps that are accepted are shown as green 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

$$p(z^{(\tau+1)} = z^{(\tau)}) = 0.5 \tag{11.34}$$

$$p(z^{(\tau+1)} = z^{(\tau)} + 1) = 0.25 \tag{11.35}$$

$$p(z^{(\tau+1)} = z^{(\tau)} - 1) = 0.25 \tag{11.36}$$

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.

Exercise 11.10

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 $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(M)}$ such that the following conditional independence property holds for $m \in \{1, \ldots, M-1\}$

$$p(\mathbf{z}^{(m+1)}|\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}) = p(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)}). \tag{11.37}$$

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

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

$$p(\mathbf{z}^{(m+1)}) = \sum_{\mathbf{z}^{(m)}} p(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)})p(\mathbf{z}^{(m)}). \tag{11.38}$$

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 $p^\star(\mathbf{z})$ is invariant if

$$p^\star(\mathbf{z}) = \sum_{\mathbf{z}'} T(\mathbf{z}', \mathbf{z})p^\star(\mathbf{z}'). \tag{11.39}$$

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

$$p^\star(\mathbf{z})T(\mathbf{z}, \mathbf{z}') = p^\star(\mathbf{z}')T(\mathbf{z}', \mathbf{z}) \tag{11.40}$$

for the particular distribution $p^\star(\mathbf{z})$. It is easily seen that a transition probability that satisfies detailed balance with respect to a particular distribution will leave that distribution invariant, because

$$\sum_{\mathbf{z}'} p^\star(\mathbf{z}')T(\mathbf{z}', \mathbf{z}) = \sum_{\mathbf{z}'} p^\star(\mathbf{z})T(\mathbf{z}, \mathbf{z}') = p^\star(\mathbf{z}) \sum_{\mathbf{z}'} p(\mathbf{z}'|\mathbf{z}) = p^\star(\mathbf{z}). \tag{11.41}$$

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 $m \to \infty$, the distribution $p(\mathbf{z}^{(m)})$ converges to the required invariant distribution $p^\star(\mathbf{z})$, 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 $B_1, \ldots, B_K$. This can be achieved through a mixture distribution of the form

$$T(\mathbf{z}', \mathbf{z}) = \sum_{k=1}^{K} \alpha_k B_k(\mathbf{z}', \mathbf{z}) \tag{11.42}$$

for some set of mixing coefficients $\alpha_1, \ldots, \alpha_K$ satisfying $\alpha_k \geqslant 0$ and $\sum_k \alpha_k = 1$. Alternatively, the base transitions may be combined through successive application, so that

$$T(\mathbf{z}', \mathbf{z}) = \sum_{\mathbf{z}_1} \ldots \sum_{\mathbf{z}_{n-1}} B_1(\mathbf{z}', \mathbf{z}_1) \ldots B_{K-1}(\mathbf{z}_{K-2}, \mathbf{z}_{K-1})B_K(\mathbf{z}_{K-1}, \mathbf{z}). \tag{11.43}$$

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 $B_1, B_2, \ldots, B_K, B_K, \ldots, B_2, B_1$, 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 $\mathbf{z}^\star$ from the distribution $q_k(\mathbf{z}|\mathbf{z}^{(\tau)})$ and then accept it with probability $A_k(\mathbf{z}^\star, \mathbf{z}_\tau)$ where

$$A_k(\mathbf{z}^\star, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{\widetilde{p}(\mathbf{z}^\star)q_k(\mathbf{z}^{(\tau)}|\mathbf{z}^\star)}{\widetilde{p}(\mathbf{z}^{(\tau)})q_k(\mathbf{z}^\star|\mathbf{z}^{(\tau)})}\right). \tag{11.44}$$

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 $p(\mathbf{z}) = \widetilde{p}(\mathbf{z})/Z_p$. 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

$$\begin{aligned} p(\mathbf{z})q_k(\mathbf{z}|\mathbf{z}')A_k(\mathbf{z}', \mathbf{z}) &= \min\left(p(\mathbf{z})q_k(\mathbf{z}|\mathbf{z}'), p(\mathbf{z}')q_k(\mathbf{z}'|\mathbf{z})\right) \\ &= \min\left(p(\mathbf{z}')q_k(\mathbf{z}'|\mathbf{z}), p(\mathbf{z})q_k(\mathbf{z}|\mathbf{z}')\right) \\ &= p(\mathbf{z}')q_k(\mathbf{z}'|\mathbf{z})A_k(\mathbf{z}, \mathbf{z}') \end{aligned} \tag{11.45}$$

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

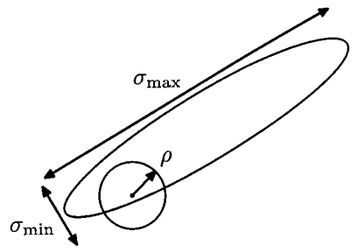


Figure 11.10 Schematic illustration of the use of an isotropic Gaussian proposal distribution (blue circle) to sample from a correlated multivariate Gaussian distribution (red ellipse) having very different standard deviations in different directions, using the Metropolis-Hastings algorithm. In order to keep the rejection rate low, the scale ρ of the proposal distribution should be on the order of the smallest standard deviation $\sigma_{\min}$, 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 $\sigma_{\max}$ 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 $\sigma_{\min}$. 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 $p(\mathbf{z}) = p(z_1, \ldots, z_M)$ 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 $z_1, \ldots, z_M$ 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

$$p(z_1|z_2^{(\tau)}, z_3^{(\tau)}). \tag{11.46}$$

Next we replace $z_2^{(\tau)}$ by a value $z_2^{(\tau+1)}$ obtained by sampling from the conditional distribution

$$p(z_2|z_1^{(\tau+1)}, z_3^{(\tau)}) \tag{11.47}$$

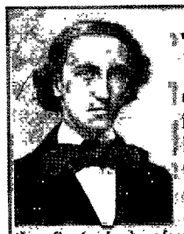
so that the new value for z_1 is used straight away in subsequent sampling steps. Then we update z_3 with a sample $z_3^{(\tau+1)}$ drawn from

$$p(z_3|z_1^{(\tau+1)}, z_2^{(\tau+1)}) \tag{11.48}$$

and so on, cycling through the three variables in turn.

Gibbs Sampling

1. Initialize $\{z_i : i = 1, \ldots, M\}$
2. For $\tau = 1, \ldots, T$:
   - Sample $z_1^{(\tau+1)} \sim p(z_1|z_2^{(\tau)}, z_3^{(\tau)}, \ldots, z_M^{(\tau)})$.
   - Sample $z_2^{(\tau+1)} \sim p(z_2|z_1^{(\tau+1)}, z_3^{(\tau)}, \ldots, z_M^{(\tau)})$.
   - $\vdots$
   - Sample $z_j^{(\tau+1)} \sim p(z_j|z_1^{(\tau+1)}, \ldots, z_{j-1}^{(\tau+1)}, z_{j+1}^{(\tau)}, \ldots, z_M^{(\tau)})$.
   - $\vdots$
   - Sample $z_M^{(\tau+1)} \sim p(z_M|z_1^{(\tau+1)}, z_2^{(\tau+1)}, \ldots, z_{M-1}^{(\tau+1)})$.



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 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 $\mathbf{z}^\star$ is given by $q_k(\mathbf{z}^\star|\mathbf{z}) = p(z_k^\star|\mathbf{z}_{\setminus k})$. We note that $\mathbf{z}_{\setminus k}^\star = \mathbf{z}_{\setminus k}$ 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

$$A(\mathbf{z}^\star, \mathbf{z}) = \frac{p(\mathbf{z}^\star)q_k(\mathbf{z}|\mathbf{z}^\star)}{p(\mathbf{z})q_k(\mathbf{z}^\star|\mathbf{z})} = \frac{p(z_k^\star|\mathbf{z}_{\setminus k}^\star)p(\mathbf{z}_{\setminus k}^\star)p(z_k|\mathbf{z}_{\setminus k}^\star)}{p(z_k|\mathbf{z}_{\setminus k})p(\mathbf{z}_{\setminus k})p(z_k^\star|\mathbf{z}_{\setminus k})} = 1 \tag{11.49}$$

where we have used $\mathbf{z}_{\setminus k}^\star = \mathbf{z}_{\setminus k}$. 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

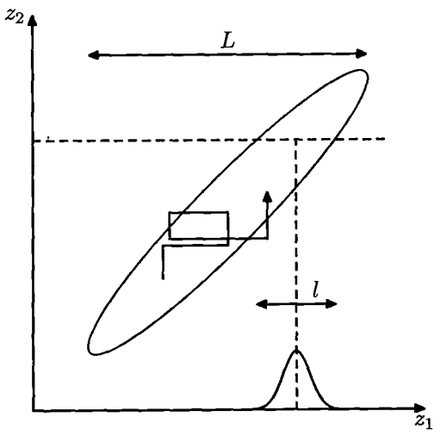


Figure 11.11 Illustration of Gibbs sampling by alternate updates of two variables whose distribution is a correlated Gaussian. The step size is governed by the standard deviation of the conditional distribution (green curve), and is $O(l)$, leading to slow progress in the direction of elongation of the 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 $p(z, y) \propto \exp(-z^2y^2)$ 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

$$z_i' = \mu_i + \alpha(z_i - \mu_i) + \sigma_i(1 - \alpha_i^2)^{1/2}\nu \tag{11.50}$$

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-

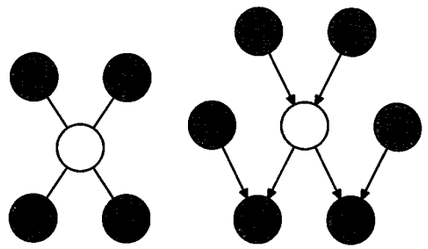


Figure 11.12 The Gibbs sampling method requires samples to be drawn from the conditional distribution of a variable conditioned on the remaining variables. For graphical models, this conditional distribution is a function only of the states of the nodes in the Markov blanket. For an undirected graph 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 $\widetilde{p}(\mathbf{z})$.

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

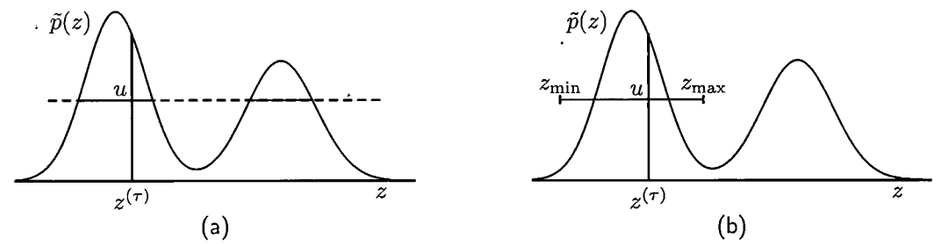


Figure 11.13 Illustration of slice sampling. (a) For a given value $z^{(\tau)}$, a value of u is chosen uniformly in the region $0 \leqslant u \leqslant \widetilde{p}(z^{(\tau)})$, which then defines a 'slice' through the distribution, shown by the solid horizontal lines. (b) Because it is infeasible to sample directly from a slice, a new sample of z is drawn from a region $z_{\min} \leqslant z \leqslant z_{\max}$, which contains the previous value $z^{(\tau)}$.

given by

$$\widehat{p}(z, u) = \begin{cases} 1/Z_p & \text{if } 0 \leqslant u \leqslant \widetilde{p}(z) \\ 0 & \text{otherwise} \end{cases} \tag{11.51}$$

where $Z_p = \int \widetilde{p}(z)\,\mathrm{d}z$. The marginal distribution over z is given by

$$\int \widehat{p}(z, u)\,\mathrm{d}u = \int_0^{\widetilde{p}(z)} \frac{1}{Z_p}\,\mathrm{d}u = \frac{\widetilde{p}(z)}{Z_p} = p(z) \tag{11.52}$$

and so we can sample from $p(z)$ by sampling from $\widehat{p}(z, u)$ and then ignoring the u values. This can be achieved by alternately sampling z and u. Given the value of z we evaluate $\widetilde{p}(z)$ and then sample u uniformly in the range $0 \leqslant u \leqslant \widetilde{p}(z)$, which is straightforward. Then we fix u and sample z uniformly from the 'slice' through the distribution defined by $\{z : \widetilde{p}(z) > u\}$. 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 $\widehat{p}(z, u)$ 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 $z_{\min} \leqslant z \leqslant z_{\max}$ 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

$$r_i = \frac{\mathrm{d}z_i}{\mathrm{d}\tau} \tag{11.53}$$

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

$$p(\mathbf{z}) = \frac{1}{Z_p} \exp\left(-E(\mathbf{z})\right) \tag{11.54}$$

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

$$\frac{\mathrm{d}r_i}{\mathrm{d}\tau} = -\frac{\partial E(\mathbf{z})}{\partial z_i}. \tag{11.55}$$

It is convenient to reformulate this dynamical system using the Hamiltonian framework. To do this, we first define the *kinetic energy* by

$$K(\mathbf{r}) = \frac{1}{2}\|\mathbf{r}\|^2 = \frac{1}{2}\sum_i r_i^2. \tag{11.56}$$

The total energy of the system is then the sum of its potential and kinetic energies

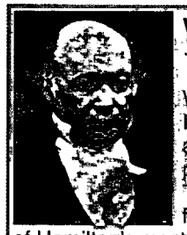
$$H(\mathbf{z}, \mathbf{r}) = E(\mathbf{z}) + K(\mathbf{r}) \tag{11.57}$$

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

Exercise 11.15

$$\frac{\mathrm{d}z_i}{\mathrm{d}\tau} = \frac{\partial H}{\partial r_i} \tag{11.58}$$

$$\frac{\mathrm{d}r_i}{\mathrm{d}\tau} = -\frac{\partial H}{\partial z_i}. \tag{11.59}$$



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

$$\begin{aligned}\frac{\mathrm{d}H}{\mathrm{d}\tau} &= \sum_i \left\{ \frac{\partial H}{\partial z_i}\frac{\mathrm{d}z_i}{\mathrm{d}\tau} + \frac{\partial H}{\partial r_i}\frac{\mathrm{d}r_i}{\mathrm{d}\tau} \right\} \\ &= \sum_i \left\{ \frac{\partial H}{\partial z_i}\frac{\partial H}{\partial r_i} - \frac{\partial H}{\partial r_i}\frac{\partial H}{\partial z_i} \right\} = 0. \end{aligned} \tag{11.60}$$

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

$$\mathbf{V} = \left(\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\tau}, \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\tau} \right) \tag{11.61}$$

and that the divergence of this field vanishes

$$\begin{aligned}\mathrm{div}\, \mathbf{V} &= \sum_i \left\{ \frac{\partial}{\partial z_i}\frac{\mathrm{d}z_i}{\mathrm{d}\tau} + \frac{\partial}{\partial r_i}\frac{\mathrm{d}r_i}{\mathrm{d}\tau} \right\} \\ &= \sum_i \left\{ -\frac{\partial}{\partial z_i}\frac{\partial H}{\partial r_i} + \frac{\partial}{\partial r_i}\frac{\partial H}{\partial z_i} \right\} = 0. \end{aligned} \tag{11.62}$$

Now consider the joint distribution over phase space whose total energy is the Hamiltonian, i.e., the distribution given by

$$p(\mathbf{z}, \mathbf{r}) = \frac{1}{Z_H} \exp(-H(\mathbf{z}, \mathbf{r})). \tag{11.63}$$

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.

Exercise 11.16

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 $\widehat{\mathbf{z}}$ and $\widehat{\mathbf{r}}$ to the position and momentum variables using

$$\widehat{r}_i(\tau + \epsilon/2) = \widehat{r}_i(\tau) - \frac{\epsilon}{2}\frac{\partial E}{\partial z_i}(\widehat{\mathbf{z}}(\tau)) \tag{11.64}$$

$$\widehat{z}_i(\tau + \epsilon) = \widehat{z}_i(\tau) + \epsilon \widehat{r}_i(\tau + \epsilon/2) \tag{11.65}$$

$$\widehat{r}_i(\tau + \epsilon) = \widehat{r}_i(\tau + \epsilon/2) - \frac{\epsilon}{2}\frac{\partial E}{\partial z_i}(\widehat{\mathbf{z}}(\tau + \epsilon)). \tag{11.66}$$

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 $\epsilon \to 0$. 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.

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 $(\mathbf{z}^\star, \mathbf{r}^\star)$ is the state after the leapfrog integration, then this candidate state is accepted with probability

$$\min\left(1, \exp\{H(\mathbf{z}, \mathbf{r}) - H(\mathbf{z}^\star, \mathbf{r}^\star)\}\right). \tag{11.67}$$

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

$$\frac{1}{Z_H}\exp(-H(\mathcal{R}))\delta V \frac{1}{2}\min\left\{1, \exp(-H(\mathcal{R}) + H(\mathcal{R}'))\right\}. \tag{11.68}$$

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

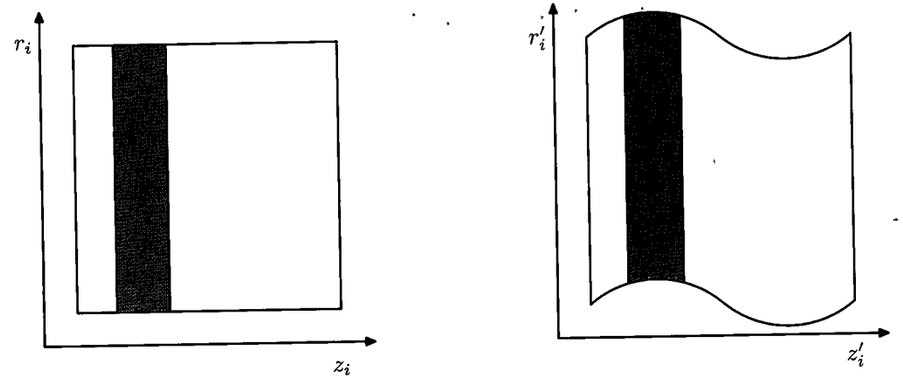


Figure 11.14 Each step of the leapfrog algorithm (11.64)–(11.66) modifies either a position variable z_i or a momentum variable r_i. Because the change to one variable is a function only of the other, any region in phase space will be sheared without change of volume.

region $\mathcal{R}'$ and integrating backwards in time to end up in region $\mathcal{R}$ is given by

$$\frac{1}{Z_H}\exp(-H(\mathcal{R}'))\delta V \frac{1}{2}\min\left\{1, \exp(-H(\mathcal{R}') + H(\mathcal{R}))\right\}. \tag{11.69}$$

Exercise 11.17 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

$$H(\mathbf{z}, \mathbf{r}) = \frac{1}{2}\sum_i \frac{1}{\sigma_i^2} z_i^2 + \frac{1}{2}\sum_i r_i^2. \tag{11.70}$$

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

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 $\sigma_{\min}$. 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 $\sigma_{\min}$. 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

$$p_E(\mathbf{z}) = \frac{1}{Z_E}\exp(-E(\mathbf{z})) \tag{11.71}$$

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 $G(\mathbf{z})$

$$\begin{aligned}\frac{Z_E}{Z_G} &= \frac{\sum_{\mathbf{z}}\exp(-E(\mathbf{z}))}{\sum_{\mathbf{z}}\exp(-G(\mathbf{z}))} \\ &= \frac{\sum_{\mathbf{z}}\exp(-E(\mathbf{z})+G(\mathbf{z}))\exp(-G(\mathbf{z}))}{\sum_{\mathbf{z}}\exp(-G(\mathbf{z}))} \\ &= \mathbb{E}_{G(\mathbf{z})}[\exp(-E+G)] \\ &\simeq \sum_l \exp(-E(\mathbf{z}^{(l)})+G(\mathbf{z}^{(l)}))\end{aligned} \tag{11.72}$$

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 $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(L)}$, then the sampling distribution can be written as

$$\frac{1}{Z_G}\exp(-G(\mathbf{z})) = \sum_{l=1}^{L} T(\mathbf{z}^{(l)}, \mathbf{z}) \tag{11.73}$$

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 $p_2, \ldots, p_{M-1}$ 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

$$\frac{Z_M}{Z_1} = \frac{Z_2}{Z_1}\frac{Z_3}{Z_2}\cdots\frac{Z_M}{Z_{M-1}} \tag{11.74}$$

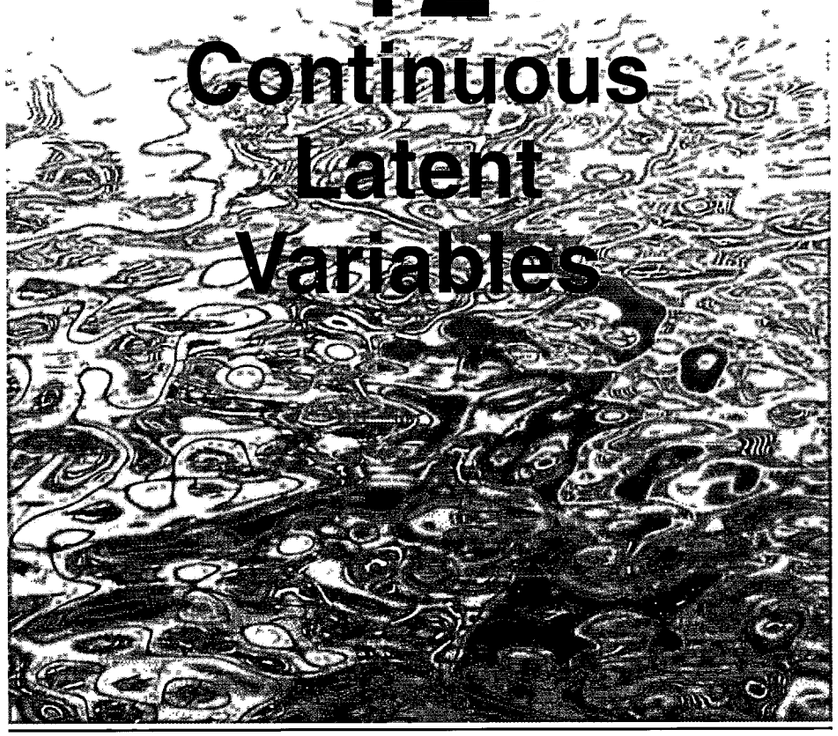
in which the intermediate ratios can be determined using Monte Carlo methods as discussed above. One way to construct such a sequence of intermediate systems is to use an energy function containing a continuous parameter $0 \leqslant \alpha \leqslant 1$ that interpolates between the two distributions

$$E_\alpha(\mathbf{z}) = (1-\alpha)E_1(\mathbf{z}) + \alpha E_M(\mathbf{z}). \tag{11.75}$$

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.

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



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

Appendix A

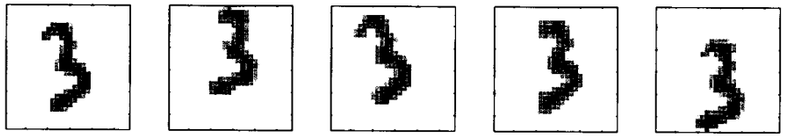


Figure 12.1 A synthetic data set obtained by taking one of the off-line digit images and creating multiple copies in each of which the digit has undergone a random displacement and rotation within some larger image field. The resulting images each have $100 \times 100 = 10,000$ pixels.

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.

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 8.1.4

Section 12.1

In this chapter w will begin with a standard, nonprobabilistic treatment of PCA, and then we show how PCA arises naturally as the maximum likelihood solution to

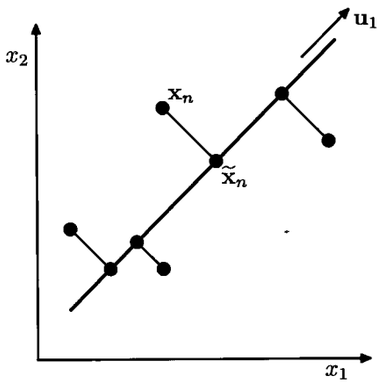


Figure 12.2 Principal component analysis seeks a space of lower dimensionality, known as the principal subspace and denoted by the magenta line, such that the orthogonal projection of the data points (red dots) onto this subspace maximizes the variance of the projected points (green dots). An alternative 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-Loève* 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 $n = 1, \ldots, N$, 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 $\mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1 = 1$ (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 $\mathbf{u}_1^{\mathrm{T}}\mathbf{x}_n$. The mean of the projected data is $\mathbf{u}_1^{\mathrm{T}}\overline{\mathbf{x}}$ where $\overline{\mathbf{x}}$ is the sample set mean given by

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \tag{12.1}$$

and the variance of the projected data is given by

$$\frac{1}{N}\sum_{n=1}^{N}\left\{\mathbf{u}_1^{\mathrm{T}}\mathbf{x}_n - \mathbf{u}_1^{\mathrm{T}}\overline{\mathbf{x}}\right\}^2 = \mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1 \tag{12.2}$$

where $\mathbf{S}$ is the data covariance matrix defined by

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^{\mathrm{T}}. \tag{12.3}$$

We now maximize the projected variance $\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1$ with respect to $\mathbf{u}_1$. Clearly, this has to be a constrained maximization to prevent $\|\mathbf{u}_1\| \to \infty$. The appropriate constraint comes from the normalization condition $\mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1 = 1$. To enforce this constraint, we introduce a Lagrange multiplier that we shall denote by λ_1, and then make an unconstrained maximization of

Appendix E

$$\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1 + \lambda_1\left(1 - \mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1\right). \tag{12.4}$$

By setting the derivative with respect to $\mathbf{u}_1$ equal to zero, we see that this quantity will have a stationary point when

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1 \tag{12.5}$$

which says that $\mathbf{u}_1$ must be an eigenvector of $\mathbf{S}$. If we left-multiply by $\mathbf{u}_1^{\mathrm{T}}$ and make use of $\mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1 = 1$, we see that the variance is given by

$$\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1 = \lambda_1 \tag{12.6}$$

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 $\mathbf{u}_1, \ldots, \mathbf{u}_M$ of the data covariance matrix $\mathbf{S}$ corresponding to the M largest eigenvalues $\lambda_1, \ldots, \lambda_M$. This is easily shown using proof by induction.

Exercise 12.1

To summarize, principal component analysis involves evaluating the mean $\overline{\mathbf{x}}$ 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.

Section 12.2.2

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 $i = 1, \ldots, D$ that satisfy

Appendix C

$$\mathbf{u}_i^{\mathrm{T}}\mathbf{u}_j = \delta_{ij}. \tag{12.7}$$

Because this basis is complete, each data point can be represented exactly by a linear combination of the basis vectors

$$\mathbf{x}_n = \sum_{i=1}^{D}\alpha_{ni}\mathbf{u}_i \tag{12.8}$$

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 $\{x_{n1}, \ldots, x_{nD}\}$ are replaced by an equivalent set $\{\alpha_{n1}, \ldots, \alpha_{nD}\}$. Taking the inner product with $\mathbf{u}_j$, and making use of the orthonormality property, we obtain $\alpha_{nj} = \mathbf{x}_n^{\mathrm{T}}\mathbf{u}_j$, and so without loss of generality we can write

$$\mathbf{x}_n = \sum_{i=1}^{D}\left(\mathbf{x}_n^{\mathrm{T}}\mathbf{u}_i\right)\mathbf{u}_i. \tag{12.9}$$

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

$$\widetilde{\mathbf{x}}_n = \sum_{i=1}^{M}z_{ni}\mathbf{u}_i + \sum_{i=M+1}^{D}b_i\mathbf{u}_i \tag{12.10}$$

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 $\widetilde{\mathbf{x}}_n$, averaged over the data set, so that our goal is to minimize

$$J = \frac{1}{N}\sum_{n=1}^{N}\|\mathbf{x}_n - \widetilde{\mathbf{x}}_n\|^2. \tag{12.11}$$

Consider first of all the minimization with respect to the quantities $\{z_{ni}\}$. Substituting for $\widetilde{\mathbf{x}}_n$, setting the derivative with respect to z_{nj} to zero, and making use of the orthonormality conditions, we obtain

$$z_{nj} = \mathbf{x}_n^{\mathrm{T}}\mathbf{u}_j \tag{12.12}$$

where $j = 1, \ldots, M$. Similarly, setting the derivative of J with respect to b_i to zero, and again making use of the orthonormality relations, gives

$$b_j = \overline{\mathbf{x}}^{\mathrm{T}}\mathbf{u}_j \tag{12.13}$$

where $j = M+1, \ldots, D$. If we substitute for z_{ni} and b_i, and make use of the general expansion (12.9), we obtain

$$\mathbf{x}_n - \widetilde{\mathbf{x}}_n = \sum_{i=M+1}^{D}\left\{(\mathbf{x}_n - \overline{\mathbf{x}})^{\mathrm{T}}\mathbf{u}_i\right\}\mathbf{u}_i \tag{12.14}$$

from which we see that the displacement vector from $\mathbf{x}_n$ to $\widetilde{\mathbf{x}}_n$ lies in the space orthogonal to the principal subspace, because it is a linear combination of $\{\mathbf{u}_i\}$ for $i = M+1, \ldots, D$, as illustrated in Figure 12.2. This is to be expected because the projected points $\widetilde{\mathbf{x}}_n$ 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

$$J = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}\left(\mathbf{x}_n^{\mathrm{T}}\mathbf{u}_i - \overline{\mathbf{x}}^{\mathrm{T}}\mathbf{u}_i\right)^2 = \sum_{i=M+1}^{D}\mathbf{u}_i^{\mathrm{T}}\mathbf{S}\mathbf{u}_i. \tag{12.15}$$

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 $J = \mathbf{u}_2^{\mathrm{T}}\mathbf{S}\mathbf{u}_2$, subject to the normalization constraint $\mathbf{u}_2^{\mathrm{T}}\mathbf{u}_2 = 1$. Using a Lagrange multiplier λ_2 to enforce the constraint, we consider the minimization of

$$\widetilde{J} = \mathbf{u}_2^{\mathrm{T}}\mathbf{S}\mathbf{u}_2 + \lambda_2\left(1 - \mathbf{u}_2^{\mathrm{T}}\mathbf{u}_2\right). \tag{12.16}$$

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

$$\mathbf{S}\mathbf{u}_i = \lambda_i\mathbf{u}_i \tag{12.17}$$

where $i = 1, \ldots, D$, and as usual the eigenvectors $\{\mathbf{u}_i\}$ are chosen to be orthonormal. The corresponding value of the distortion measure is then given by

$$J = \sum_{i=M+1}^{D}\lambda_i \tag{12.18}$$

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

Appendix A

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

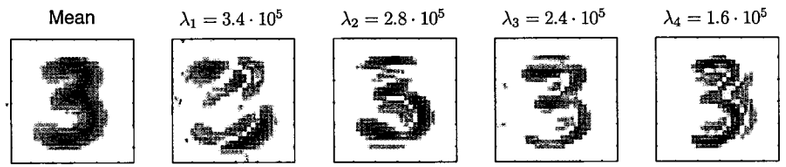


Figure 12.3 The mean vector $\overline{\mathbf{x}}$ along with the first four PCA eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_4$ for the off-line digits data set, together with the corresponding eigenvalues.

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

$$\widetilde{\mathbf{x}}_n = \sum_{i=1}^{M} (\mathbf{x}_n^{\mathrm{T}}\mathbf{u}_i)\mathbf{u}_i + \sum_{i=M+1}^{D} (\overline{\mathbf{x}}^{\mathrm{T}}\mathbf{u}_i)\mathbf{u}_i \tag{12.19}$$

$$= \overline{\mathbf{x}} + \sum_{i=1}^{M} \left(\mathbf{x}_n^{\mathrm{T}}\mathbf{u}_i - \overline{\mathbf{x}}^{\mathrm{T}}\mathbf{u}_i\right)\mathbf{u}_i \tag{12.20}$$

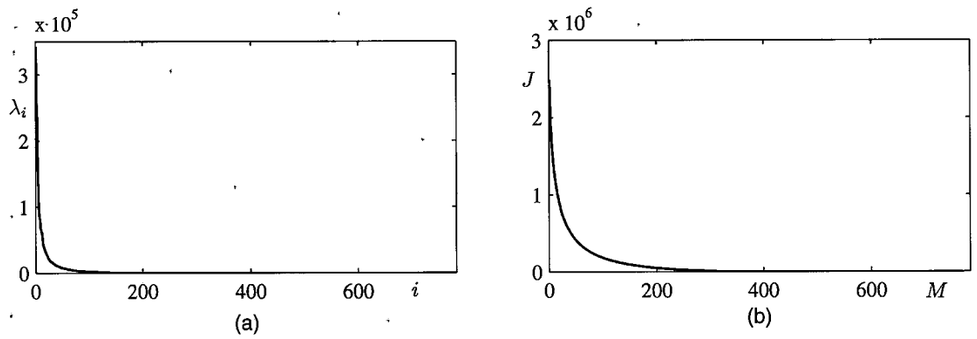


Figure 12.4 (a) Plot of the eigenvalue spectrum for the off-line digits data set. (b) Plot of the sum of the discarded eigenvalues, which represents the sum-of-squares distortion J introduced by projecting the data onto a principal component subspace of dimensionality M.

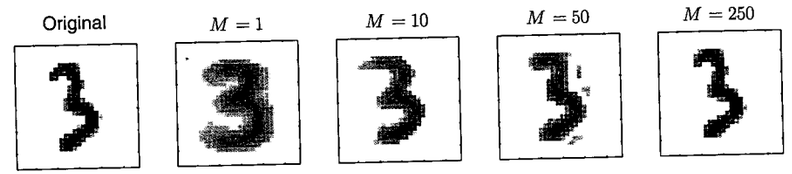


Figure 12.5 An original example from the off-line digits data set together with its PCA reconstructions obtained by retaining M principal components for various values of M. As M increases the reconstruction becomes more accurate and would become perfect when $M = D = 28 \times 28 = 784$.

where we have made use of the relation

$$\overline{\mathbf{x}} = \sum_{i=1}^{D} \left(\overline{\mathbf{x}}^{\mathrm{T}}\mathbf{u}_i\right)\mathbf{u}_i \tag{12.21}$$

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 $\left(\mathbf{x}_n^{\mathrm{T}}\mathbf{u}_i - \overline{\mathbf{x}}^{\mathrm{T}}\mathbf{u}_i\right)$. 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

Appendix A

Section 9.1

$$\rho_{ij} = \frac{1}{N}\sum_{n=1}^{N} \frac{(x_{ni} - \overline{x}_i)}{\sigma_i}\frac{(x_{nj} - \overline{x}_j)}{\sigma_j} \tag{12.22}$$

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

$$\mathbf{SU} = \mathbf{UL} \tag{12.23}$$

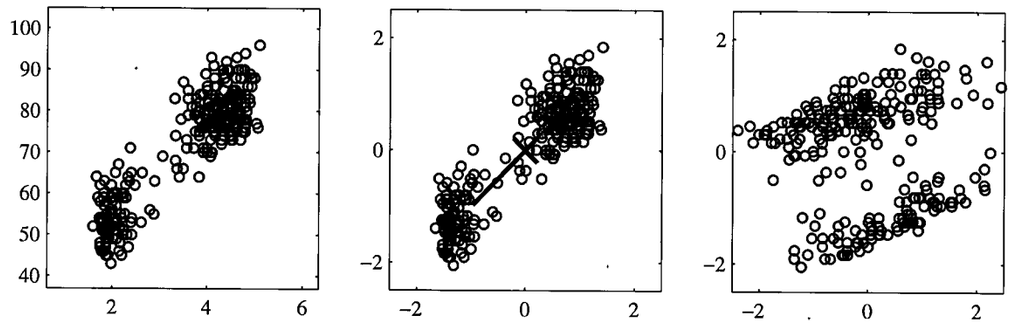


Figure 12.6 Illustration of the effects of linear pre-processing applied to the Old Faithful data set. The plot on the left shows the original data. The centre plot shows the result of standardizing the individual variables to zero mean and unit variance. Also shown are the principal axes of this normalized 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

$$\mathbf{y}_n = \mathbf{L}^{-1/2}\mathbf{U}^{\mathrm{T}}(\mathbf{x}_n - \overline{\mathbf{x}}) \tag{12.24}$$

where $\overline{\mathbf{x}}$ 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

$$\begin{aligned}\frac{1}{N}\sum_{n=1}^{N}\mathbf{y}_n\mathbf{y}_n^{\mathrm{T}} &= \frac{1}{N}\sum_{n=1}^{N}\mathbf{L}^{-1/2}\mathbf{U}^{\mathrm{T}}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^{\mathrm{T}}\mathbf{U}\mathbf{L}^{-1/2} \\ &= \mathbf{L}^{-1/2}\mathbf{U}^{\mathrm{T}}\mathbf{S}\mathbf{U}\mathbf{L}^{-1/2} = \mathbf{L}^{-1/2}\mathbf{L}\mathbf{L}^{-1/2} = \mathbf{I}. \end{aligned} \tag{12.25}$$

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 $\mathbf{x}_n^{\mathrm{T}}\mathbf{u}_1$ and $\mathbf{x}_n^{\mathrm{T}}\mathbf{u}_2$, 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.

Appendix A

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.

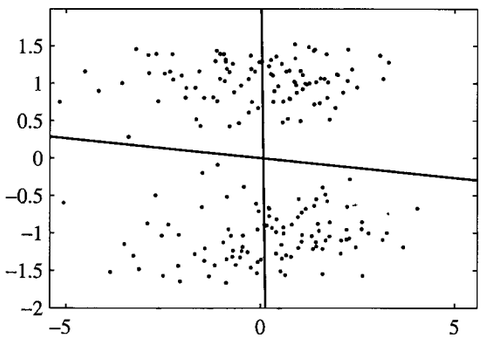
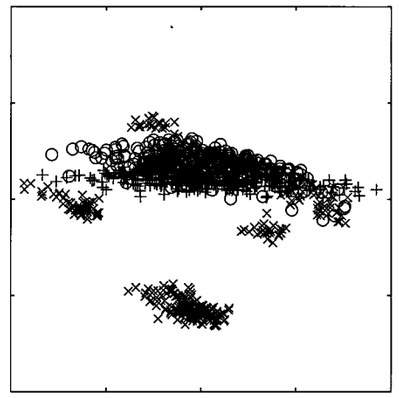


Figure 12.8 Visualization of the oil flow data set obtained by projecting the data onto the first two principal components. The red, blue, and green points correspond to the 'laminar', 'homogeneous', and 'annular' flow configurations respectively.



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 $(\mathbf{x}_n - \overline{\mathbf{x}})^{\mathrm{T}}$. The covariance matrix (12.3) can then be written as $\mathbf{S} = N^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}$, and the corresponding eigenvector equation becomes

$$\frac{1}{N}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{u}_i = \lambda_i \mathbf{u}_i. \tag{12.26}$$

Now pre-multiply both sides by $\mathbf{X}$ to give

$$\frac{1}{N}\mathbf{X}\mathbf{X}^{\mathrm{T}}(\mathbf{X}\mathbf{u}_i) = \lambda_i(\mathbf{X}\mathbf{u}_i). \tag{12.27}$$

If we now define $\mathbf{v}_i = \mathbf{X}\mathbf{u}_i$, we obtain

$$\frac{1}{N}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{12.28}$$

which is an eigenvector equation for the $N \times N$ matrix $N^{-1}\mathbf{X}\mathbf{X}^{\mathrm{T}}$. 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 $\mathbf{X}^{\mathrm{T}}$ to give

$$\left(\frac{1}{N}\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)(\mathbf{X}^{\mathrm{T}}\mathbf{v}_i) = \lambda_i(\mathbf{X}^{\mathrm{T}}\mathbf{v}_i) \tag{12.29}$$

from which we see that $(\mathbf{X}^{\mathrm{T}}\mathbf{v}_i)$ 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 $\mathbf{u}_i \propto \mathbf{X}^{\mathrm{T}}\mathbf{v}_i$ by a constant such that $\|\mathbf{u}_i\| = 1$, which, assuming $\mathbf{v}_i$ has been normalized to unit length, gives

$$\mathbf{u}_i = \frac{1}{(N\lambda_i)^{1/2}}\mathbf{X}^{\mathrm{T}}\mathbf{v}_i. \tag{12.30}$$

In summary, to apply this approach we first evaluate $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ 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.

Section 12.2.2

- 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.
- 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.

Section 12.2.3

- 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.
- 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

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}). \tag{12.31}$$

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

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I}) \tag{12.32}$$

Section 8.2.2

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

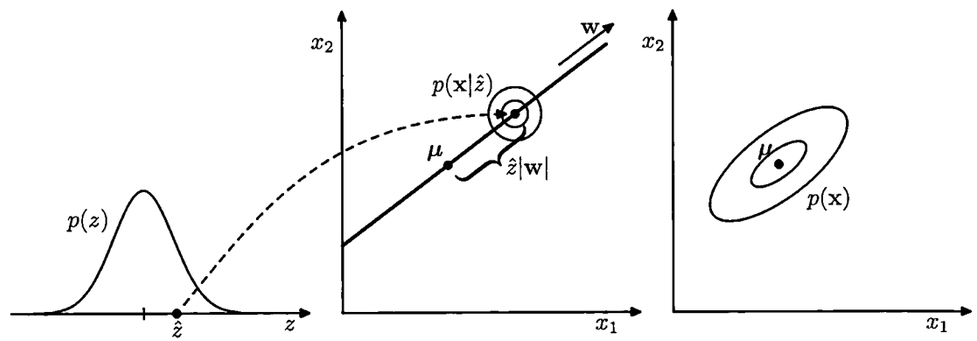


Figure 12.9 An illustration of the generative view of the probabilistic PCA model for a two-dimensional data space and a one-dimensional latent space. An observed data point $\mathbf{x}$ is generated by first drawing a value $\widehat{z}$ for the latent variable from its prior distribution $p(z)$ and then drawing a value for $\mathbf{x}$ from an isotropic Gaussian distribution (illustrated by the red circles) having mean $\mathbf{w}\widehat{z} + \boldsymbol{\mu}$ 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

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \tag{12.33}$$

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

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z}. \tag{12.34}$$

Because this corresponds to a linear-Gaussian model, this marginal distribution is again Gaussian, and is given by

Exercise 12.7

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{C}) \tag{12.35}$$

where the $D \times D$ covariance matrix $\mathbf{C}$ is defined by

$$\mathbf{C} = \mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I}. \tag{12.36}$$

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

$$\begin{aligned} \mathbb{E}[\mathbf{x}] &= \mathbb{E}[\mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}] = \boldsymbol{\mu} && (12.37) \\ \mathrm{cov}[\mathbf{x}] &= \mathbb{E}\left[(\mathbf{W}\mathbf{z} + \boldsymbol{\epsilon})(\mathbf{W}\mathbf{z} + \boldsymbol{\epsilon})^{\mathrm{T}}\right] \\ &= \mathbb{E}\left[\mathbf{W}\mathbf{z}\mathbf{z}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\right] + \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^{\mathrm{T}}] = \mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I} && (12.38) \end{aligned}$$

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 $\widetilde{\mathbf{W}} = \mathbf{W}\mathbf{R}$ where $\mathbf{R}$ is an orthogonal matrix. Using the orthogonality property $\mathbf{R}\mathbf{R}^{\mathrm{T}} = \mathbf{I}$, we see that the quantity $\widetilde{\mathbf{W}}\widetilde{\mathbf{W}}^{\mathrm{T}}$ that appears in the covariance matrix $\mathbf{C}$ takes the form

$$\widetilde{\mathbf{W}}\widetilde{\mathbf{W}}^{\mathrm{T}} = \mathbf{W}\mathbf{R}\mathbf{R}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}} = \mathbf{W}\mathbf{W}^{\mathrm{T}} \tag{12.39}$$

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

$$\mathbf{C}^{-1} = \sigma^{-1}\mathbf{I} - \sigma^{-2}\mathbf{W}\mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}} \tag{12.40}$$

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

$$\mathbf{M} = \mathbf{W}^{\mathrm{T}}\mathbf{W} + \sigma^2\mathbf{I}. \tag{12.41}$$

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

Exercise 12.8

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}|\mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}(\mathbf{x} - \boldsymbol{\mu}), \sigma^{-2}\mathbf{M}\right). \tag{12.42}$$

Note that the posterior mean depends on $\mathbf{x}$, whereas the posterior covariance is independent of $\mathbf{x}$.

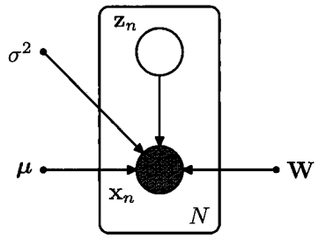


Figure 12.10 The probabilistic PCA model for a data set of N observations of $\mathbf{x}$ can be expressed as a directed graph in which each observation $\mathbf{x}_n$ is associated with a value $\mathbf{z}_n$ of the latent variable.

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

$$\begin{aligned}\ln p(\mathbf{X}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2) &= \sum_{n=1}^{N} \ln p(\mathbf{x}_n|\mathbf{W}, \boldsymbol{\mu}, \sigma^2) \\ &= -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\mathbf{C}| - \frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n - \boldsymbol{\mu})^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}). \quad (12.43)\end{aligned}$$

Setting the derivative of the log likelihood with respect to $\boldsymbol{\mu}$ equal to zero gives the expected result $\boldsymbol{\mu} = \overline{\mathbf{x}}$ where $\overline{\mathbf{x}}$ is the data mean defined by (12.1). Back-substituting we can then write the log likelihood function in the form

$$\ln p(\mathbf{X}|\mathbf{W}, \boldsymbol{\mu}, \sigma^2) = -\frac{N}{2}\left\{D\ln(2\pi) + \ln|\mathbf{C}| + \mathrm{Tr}\left(\mathbf{C}^{-1}\mathbf{S}\right)\right\} \quad (12.44)$$

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

$$\mathbf{W}_{\mathrm{ML}} = \mathbf{U}_M(\mathbf{L}_M - \sigma^2\mathbf{I})^{1/2}\mathbf{R} \quad (12.45)$$

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 $\mathbf{u}_1, \ldots, \mathbf{u}_M$. 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

$$\sigma^2_{\mathrm{ML}} = \frac{1}{D-M}\sum_{i=M+1}^{D}\lambda_i \quad (12.46)$$

so that σ^2_{ML} 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 $\mathbf{R}\mathbf{R}^{\mathrm{T}} = \mathbf{I}$, 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 $\mathbf{v}^{\mathrm{T}}\mathbf{v} = 1$, which is given by $\mathbf{v}^{\mathrm{T}}\mathbf{C}\mathbf{v}$. 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 $\mathbf{v}^{\mathrm{T}}\mathbf{U} = \mathbf{0}$ and hence $\mathbf{v}^{\mathrm{T}}\mathbf{C}\mathbf{v} = \sigma^2$. 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 $\mathbf{v}^{\mathrm{T}}\mathbf{C}\mathbf{v} = (\lambda_i - \sigma^2) + \sigma^2 = \lambda_i$. 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).

Section 12.2.2

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 $\mathbf{U}\mathbf{U}^{\mathrm{T}} = \mathbf{I}$ and $\mathbf{R}\mathbf{R}^{\mathrm{T}} = \mathbf{I}$, we see that the covariance $\mathbf{C}$ of the marginal distribution for $\mathbf{x}$ becomes

$$\mathbf{C} = \mathbf{U}(\mathbf{L} - \sigma^2\mathbf{I})^{1/2}\mathbf{R}\mathbf{R}^{\mathrm{T}}(\mathbf{L} - \sigma^2\mathbf{I})^{1/2}\mathbf{U}^{\mathrm{T}} + \sigma^2\mathbf{I} = \mathbf{U}\mathbf{L}\mathbf{U}^{\mathrm{T}} = \mathbf{S} \tag{12.47}$$

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

$$\mathbb{E}[\mathbf{z}|\mathbf{x}] = \mathbf{M}^{-1}\mathbf{W}_{\mathrm{ML}}^{\mathrm{T}}(\mathbf{x} - \overline{\mathbf{x}}) \tag{12.48}$$

where $\mathbf{M}$ is given by (12.41). This projects to a point in data space given by

$$\mathbf{W}\mathbb{E}[\mathbf{z}|\mathbf{x}] + \boldsymbol{\mu}. \tag{12.49}$$

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 $\sigma^2 \to 0$, then the posterior mean reduces to

$$(\mathbf{W}_{\mathrm{ML}}^{\mathrm{T}}\mathbf{W}_{\mathrm{ML}})^{-1}\mathbf{W}_{\mathrm{ML}}^{\mathrm{T}}(\mathbf{x} - \overline{\mathbf{x}}) \tag{12.50}$$

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 12.11

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

$$DM + 1 - M(M-1)/2. \tag{12.51}$$

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

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

Section 9.4

are assumed independent, the complete-data log likelihood function takes the form

$$\ln p\left(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2\right) = \sum_{n=1}^{N} \{\ln p(\mathbf{x}_n|\mathbf{z}_n) + \ln p(\mathbf{z}_n)\} \tag{12.52}$$

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 $\overline{\mathbf{x}}$ 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

$$\begin{aligned}\mathbb{E}[\ln p\left(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2\right)] = -\sum_{n=1}^{N} \Bigg\{ & \frac{D}{2}\ln(2\pi\sigma^2) + \frac{1}{2}\mathrm{Tr}\left(\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}]\right) \\ & + \frac{1}{2\sigma^2}\|\mathbf{x}_n - \boldsymbol{\mu}\|^2 - \frac{1}{\sigma^2}\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}(\mathbf{x}_n - \boldsymbol{\mu}) \\ & + \frac{1}{2\sigma^2}\mathrm{Tr}\left(\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}]\mathbf{W}^{\mathrm{T}}\mathbf{W}\right)\Bigg\}. \end{aligned} \tag{12.53}$$

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

$$\mathbb{E}[\mathbf{z}_n] = \mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}(\mathbf{x}_n - \overline{\mathbf{x}}) \tag{12.54}$$

$$\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}] = \sigma^2\mathbf{M}^{-1} + \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}} \tag{12.55}$$

which follow directly from the posterior distribution (12.42) together with the standard result $\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}] = \mathrm{cov}[\mathbf{z}_n] + \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}}$. 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

Exercise 12.15

$$\mathbf{W}_{\text{new}} = \left[\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}}\right]\left[\sum_{n=1}^{N}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}]\right]^{-1} \tag{12.56}$$

$$\begin{aligned}\sigma^2_{\text{new}} = \frac{1}{ND}\sum_{n=1}^{N} \Big\{ & \|\mathbf{x}_n - \overline{\mathbf{x}}\|^2 - 2\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}}\mathbf{W}_{\text{new}}^{\mathrm{T}}(\mathbf{x}_n - \overline{\mathbf{x}}) \\ & + \mathrm{Tr}\left(\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}]\mathbf{W}_{\text{new}}^{\mathrm{T}}\mathbf{W}_{\text{new}}\right)\Big\}. \end{aligned} \tag{12.57}$$

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

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 $\sigma^2 \to 0$, 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 $\mathbf{M} = \mathbf{W}^{\mathrm{T}}\mathbf{W}$. To emphasize the simplicity of the algorithm, let us define $\widetilde{\mathbf{X}}$ to be a matrix of size $N \times D$ whose n^{th} row is given by the vector $\mathbf{x}_n - \overline{\mathbf{x}}$ and similarly define $\boldsymbol{\Omega}$ 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

$$\boldsymbol{\Omega} = (\mathbf{W}_{\text{old}}^{\mathrm{T}}\mathbf{W}_{\text{old}})^{-1}\mathbf{W}_{\text{old}}^{\mathrm{T}}\widetilde{\mathbf{X}} \tag{12.58}$$

and the M step (12.56) takes the form

$$\mathbf{W}_{\text{new}} = \widetilde{\mathbf{X}}^{\mathrm{T}}\boldsymbol{\Omega}^{\mathrm{T}}(\boldsymbol{\Omega}\boldsymbol{\Omega}^{\mathrm{T}})^{-1}. \tag{12.59}$$

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

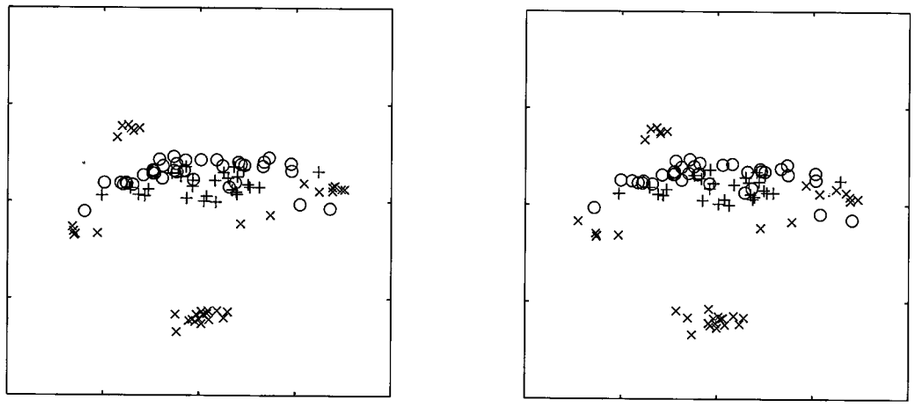


Figure 12.11 Probabilistic PCA visualization of a portion of the oil flow data set for the first 100 data points. The left-hand plot shows the posterior mean projections of the data points on the principal subspace. The right-hand plot is obtained by first randomly omitting 30% of the variable values 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.

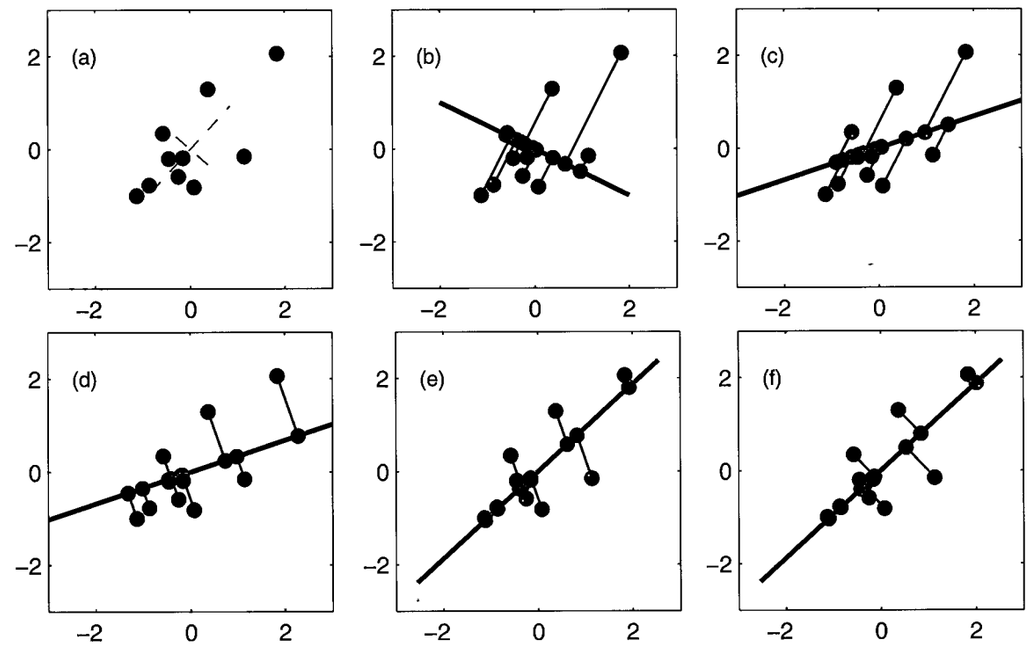


Figure 12.12 Synthetic data illustrating the EM algorithm for PCA defined by (12.58) and (12.59). (a) A data set $\mathbf{X}$ with the data points shown in green, together with the true principal components (shown as eigenvectors scaled by the square roots of the eigenvalues). (b) Initial configuration of the principal subspace defined by $\mathbf{W}$, shown in red, together with the projections of the latent points $\mathbf{Z}$ into the data space, given by $\mathbf{Z}\mathbf{W}^{\mathrm{T}}$, shown in cyan. (c) After one M step, the latent space has been updated with $\mathbf{Z}$ held fixed. (d) After the successive E step, the values of $\mathbf{Z}$ have been updated, giving orthogonal projections, with $\mathbf{W}$ 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 $\boldsymbol{\mu}$, $\mathbf{W}$, 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-*

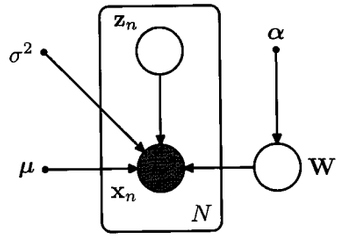


Figure 12.13 Probabilistic graphical model for Bayesian PCA in which the distribution over the parameter matrix $\mathbf{W}$ is governed by a vector $\boldsymbol{\alpha}$ of hyperparameters.

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

$$p(\mathbf{W}|\boldsymbol{\alpha}) = \prod_{i=1}^{M} \left(\frac{\alpha_i}{2\pi}\right)^{D/2} \exp\left\{-\frac{1}{2}\alpha_i \mathbf{w}_i^{\mathrm{T}}\mathbf{w}_i\right\} \tag{12.60}$$

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

$$p(\mathbf{X}|\boldsymbol{\alpha}, \mu, \sigma^2) = \int p(\mathbf{X}|\mathbf{W}, \mu, \sigma^2) p(\mathbf{W}|\boldsymbol{\alpha})\, \mathrm{d}\mathbf{W} \tag{12.61}$$

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

$$\alpha_i^{\text{new}} = \frac{D}{\mathbf{w}_i^{\mathrm{T}}\mathbf{w}_i} \tag{12.62}$$

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

$$\mathbf{W}_{\text{new}} = \left[\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}}\right]\left[\sum_{n=1}^{N}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}] + \sigma^2\mathbf{A}\right]^{-1} \tag{12.63}$$

where $\mathbf{A} = \mathrm{diag}(\alpha_i)$. 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 $\boldsymbol{\alpha}$, 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

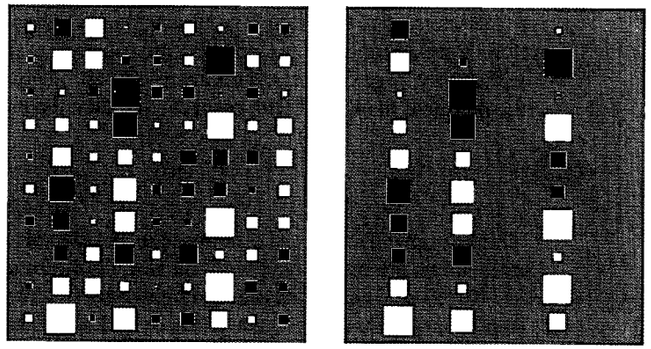


Figure 12.14 'Hinton' diagrams of the matrix $\mathbf{W}$ in which each element of the matrix is depicted as a square (white for positive and black for negative values) whose area is proportional to the magnitude of that element. The synthetic data set comprises 300 data points in $D = 10$ dimensions 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

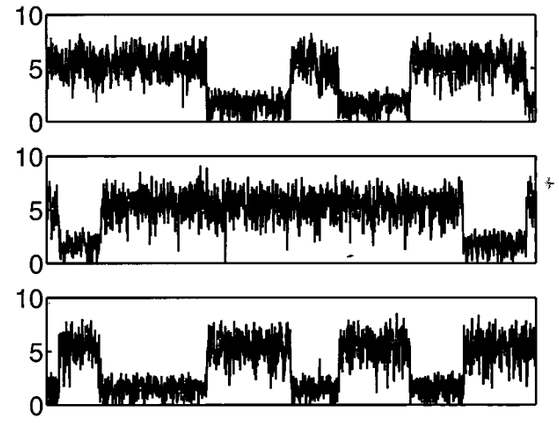
$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \tag{12.64}$$

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 $x_1, \ldots, x_D$ 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

$$\mathbf{C} = \mathbf{W}\mathbf{W}^{\mathrm{T}} + \boldsymbol{\Psi}. \tag{12.65}$$

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.

Section 12.4

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

Exercise 12.21

$$\mathbb{E}[\mathbf{z}_n] = \mathbf{G}\mathbf{W}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}(\mathbf{x}_n - \overline{\mathbf{x}}) \tag{12.66}$$

$$\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}] = \mathbf{G} + \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}} \tag{12.67}$$

where we have defined

$$\mathbf{G} = (\mathbf{I} + \mathbf{W}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\mathbf{W})^{-1}. \tag{12.68}$$

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

to compute in $O(D)$ steps), which is convenient because often $M \ll D$. Similarly, the M-step equations take the form

Exercise 12.22

$$\mathbf{W}^{\text{new}} = \left[\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}}\right]\left[\sum_{n=1}^{N}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}]\right]^{-1} \tag{12.69}$$

$$\mathbf{\Psi}^{\text{new}} = \text{diag}\left\{\mathbf{S} - \mathbf{W}_{\text{new}}\frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[\mathbf{z}_n](\mathbf{x}_n - \overline{\mathbf{x}})^{\mathrm{T}}\right\} \tag{12.70}$$

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 $\mathbf{\Psi}$.

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 $\mathbf{x}^{\mathrm{T}}\mathbf{x}'$ 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 $n = 1, \ldots, N$, 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 $\mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m$. Recall that the principal components are defined by the eigenvectors $\mathbf{u}_i$ of the covariance matrix

$$\mathbf{S}\mathbf{u}_i = \lambda_i\mathbf{u}_i \tag{12.71}$$

where $i = 1, \ldots, D$. Here the $D \times D$ sample covariance matrix $\mathbf{S}$ is defined by

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}, \tag{12.72}$$

and the eigenvectors are normalized such that $\mathbf{u}_i^{\mathrm{T}}\mathbf{u}_i = 1$.

Now consider a nonlinear transformation $\boldsymbol{\phi}(\mathbf{x})$ into an M-dimensional feature space, so that each data point $\mathbf{x}_n$ is thereby projected onto a point $\boldsymbol{\phi}(\mathbf{x}_n)$. We can

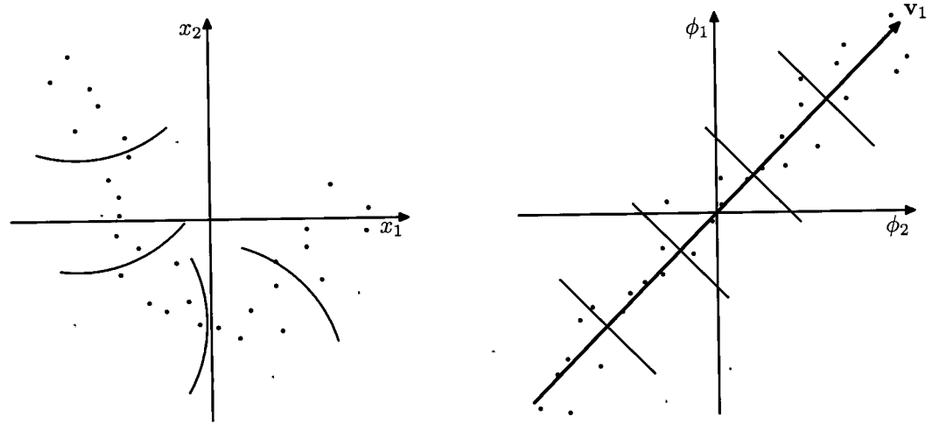


Figure 12.16 Schematic illustration of kernel PCA. A data set in the original data space (left-hand plot) is projected by a nonlinear transformation $\boldsymbol{\phi}(\mathbf{x})$ into a feature space (right-hand plot). By performing PCA in the feature space, we obtain the principal components, of which 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 $\sum_n \boldsymbol{\phi}(\mathbf{x}_n) = \mathbf{0}$. We shall return to this point shortly. The $M \times M$ sample covariance matrix in feature space is given by

$$\mathbf{C} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} \tag{12.73}$$

and its eigenvector expansion is defined by

$$\mathbf{C}\mathbf{v}_i = \lambda_i\mathbf{v}_i \tag{12.74}$$

$i = 1, \ldots, M$. 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

$$\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\left\{\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\mathbf{v}_i\right\} = \lambda_i\mathbf{v}_i \tag{12.75}$$

and so we see that (provided $\lambda_i > 0$) the vector $\mathbf{v}_i$ is given by a linear combination of the $\boldsymbol{\phi}(\mathbf{x}_n)$ and so can be written in the form

$$\mathbf{v}_i = \sum_{n=1}^{N}a_{in}\boldsymbol{\phi}(\mathbf{x}_n). \tag{12.76}$$

Substituting this expansion back into the eigenvector equation, we obtain

$$\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\sum_{m=1}^{N}a_{im}\boldsymbol{\phi}(\mathbf{x}_m)=\lambda_i\sum_{n=1}^{N}a_{in}\boldsymbol{\phi}(\mathbf{x}_n). \tag{12.77}$$

The key step is now to express this in terms of the kernel function $k(\mathbf{x}_n,\mathbf{x}_m)=\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_m)$, which we do by multiplying both sides by $\boldsymbol{\phi}(\mathbf{x}_l)^{\mathrm{T}}$ to give

$$\frac{1}{N}\sum_{n=1}^{N}k(\mathbf{x}_l,\mathbf{x}_n)\sum_{m=1}^{m}a_{im}k(\mathbf{x}_n,\mathbf{x}_m)=\lambda_i\sum_{n=1}^{N}a_{in}k(\mathbf{x}_l,\mathbf{x}_n). \tag{12.78}$$

This can be written in matrix notation as

$$\mathbf{K}^2\mathbf{a}_i=\lambda_i N\mathbf{K}\mathbf{a}_i \tag{12.79}$$

where $\mathbf{a}_i$ is an N-dimensional column vector with elements a_{ni} for $n=1,\ldots,N$. We can find solutions for $\mathbf{a}_i$ by solving the following eigenvalue problem

$$\mathbf{K}\mathbf{a}_i=\lambda_i N\mathbf{a}_i \tag{12.80}$$

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

$$1=\mathbf{v}_i^{\mathrm{T}}\mathbf{v}_i=\sum_{n=1}^{N}\sum_{m=1}^{N}a_{in}a_{im}\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_m)=\mathbf{a}_i^{\mathrm{T}}\mathbf{K}\mathbf{a}_i=\lambda_i N\mathbf{a}_i^{\mathrm{T}}\mathbf{a}_i. \tag{12.81}$$

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

$$y_i(\mathbf{x})=\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{v}_i=\sum_{n=1}^{N}a_{in}\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)=\sum_{n=1}^{N}a_{in}k(\mathbf{x},\mathbf{x}_n) \tag{12.82}$$

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 $\boldsymbol{\phi}(\mathbf{x}_n)$ 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 $\widetilde{\boldsymbol{\phi}}(\mathbf{x}_n)$, are given by

$$\widetilde{\boldsymbol{\phi}}(\mathbf{x}_n)=\boldsymbol{\phi}(\mathbf{x}_n)-\frac{1}{N}\sum_{l=1}^{N}\boldsymbol{\phi}(\mathbf{x}_l) \tag{12.83}$$

and the corresponding elements of the Gram matrix are given by

$$\begin{aligned}\widetilde{K}_{nm}&=\widetilde{\boldsymbol{\phi}}(\mathbf{x}_n)^{\mathrm{T}}\widetilde{\boldsymbol{\phi}}(\mathbf{x}_m)\\&=\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_m)-\frac{1}{N}\sum_{l=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_l)\\&\quad-\frac{1}{N}\sum_{l=1}^{N}\boldsymbol{\phi}(\mathbf{x}_l)^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_m)+\frac{1}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N}\boldsymbol{\phi}(\mathbf{x}_j)^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_l)\\&=k(\mathbf{x}_n,\mathbf{x}_m)-\frac{1}{N}\sum_{l=1}^{N}k(\mathbf{x}_l,\mathbf{x}_m)\\&\quad-\frac{1}{N}\sum_{l=1}^{N}k(\mathbf{x}_n,\mathbf{x}_l)+\frac{1}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N}k(\mathbf{x}_j,\mathbf{x}_l).\end{aligned} \tag{12.84}$$

This can be expressed in matrix notation as

$$\widetilde{\mathbf{K}}=\mathbf{K}-\mathbf{1}_N\mathbf{K}-\mathbf{K}\mathbf{1}_N+\mathbf{1}_N\mathbf{K}\mathbf{1}_N \tag{12.85}$$

where $\mathbf{1}_N$ denotes the $N\times N$ matrix in which every element takes the value $1/N$. Thus we can evaluate $\widetilde{\mathbf{K}}$ using only the kernel function and then use $\widetilde{\mathbf{K}}$ to determine the eigenvalues and eigenvectors. Note that the standard PCA algorithm is recovered as a special case if we use a linear kernel $k(\mathbf{x},\mathbf{x}')=\mathbf{x}^{\mathrm{T}}\mathbf{x}'$. 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

Exercise 12.27

$$k(\mathbf{x},\mathbf{x}')=\exp(-\|\mathbf{x}-\mathbf{x}'\|^2/0.1) \tag{12.86}$$

is applied to a synthetic data set. The lines correspond to contours along which the projection onto the corresponding principal component, defined by

$$\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{v}_i=\sum_{n=1}^{N}a_{in}k(\mathbf{x},\mathbf{x}_n) \tag{12.87}$$

is constant.

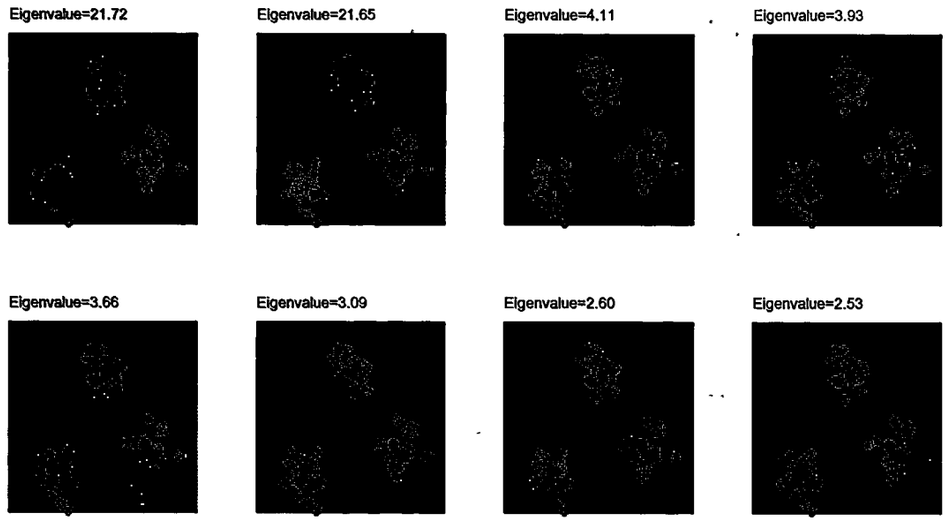


Figure 12.17 Example of kernel PCA, with a Gaussian kernel applied to a synthetic data set in two dimensions, showing the first eight eigenfunctions along with their eigenvalues. The contours are lines along which the projection onto the corresponding principal component is constant. Note how the first 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 $\widetilde{\mathbf{K}}$ 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 $\widehat{\mathbf{x}}_n$ onto the L-dimensional principal subspace, defined by

$$\widehat{\mathbf{x}}_n = \sum_{i=1}^{L} \left(\mathbf{x}_n^{\mathrm{T}}\mathbf{u}_i\right)\mathbf{u}_i. \tag{12.88}$$

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

$$p(\mathbf{z}) = \prod_{j=1}^{M} p(z_j). \tag{12.89}$$

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 $\mathbf{RR}^{\mathrm{T}} = \mathbf{I}$, 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

Exercise 12.29

$$p(z_j) = \frac{1}{\pi \cosh(z_j)} = \frac{1}{\pi(e^{z_j} + e^{-z_j})} \tag{12.90}$$

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

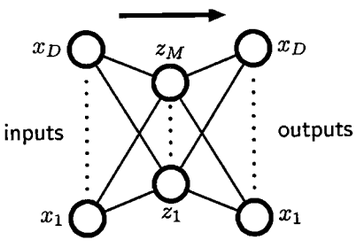


Figure 12.18 An autoassociative multilayer perceptron having two layers of weights. Such a network is trained to map input vectors onto themselves by minimization of a sum-of-squares error. Even with nonlinear units in the hidden layer, such a network is equivalent to linear principal component analysis. 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

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{x}_n\|^2. \tag{12.91}$$

If the hidden units have linear activations 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.

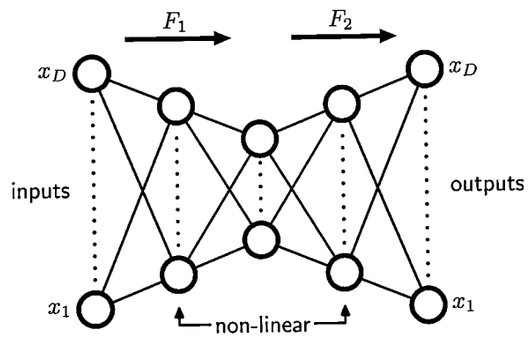


Figure 12.19 Addition of extra hidden layers of nonlinear units gives an autoassociative network which can perform a nonlinear dimensionality reduction.

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.

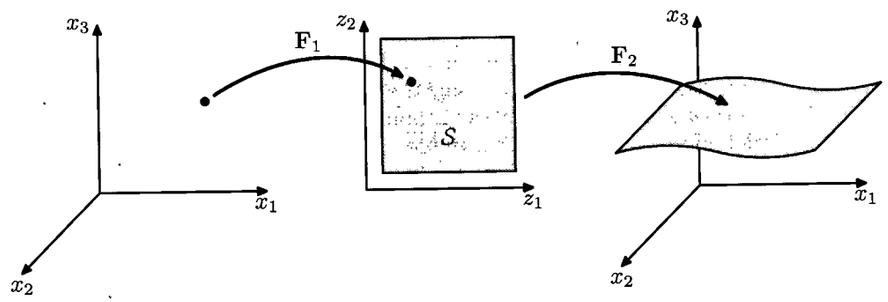


Figure 12.20 Geometrical interpretation of the mappings performed by the network in Figure 12.19 for the case of $D = 3$ inputs and $M = 2$ units in the middle hidden layer. The function $\mathbf{F}_2$ maps from an M-dimensional space $\mathcal{S}$ into a D-dimensional space and therefore defines the 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 $\widehat{\mathbf{x}}$ 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

$$\mathbb{E}\left[\mathbf{x}|g_{\mathbf{f}}(\mathbf{x}) = \lambda\right] = \mathbf{f}(\lambda). \tag{12.92}$$

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 manifold. 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 $\boldsymbol{\phi}_n$ which are parameterized by a single parameter vector $\mathbf{w}$, whereas in the latent space visualization model there is a single latent space variable $\mathbf{x}$ (analogous to $\boldsymbol{\phi}$) and N copies of the latent variable $\mathbf{w}_n$. 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.

Chapter 5

Chapter 11

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.

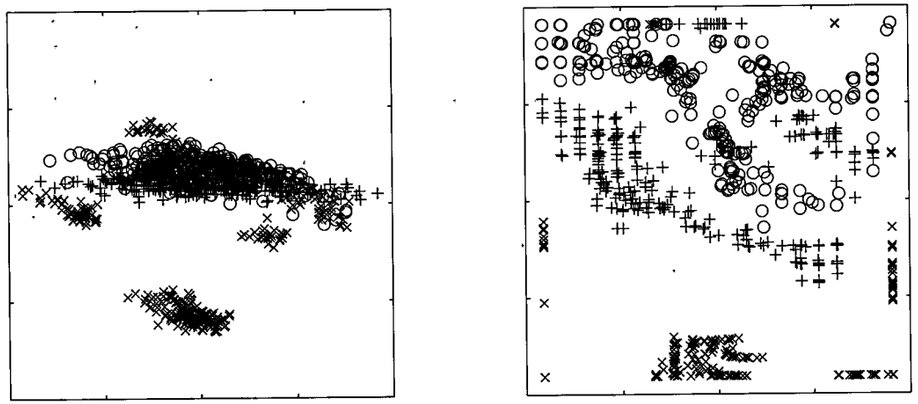


Figure 12.21 Plot of the oil flow data set visualized using PCA on the left and GTM on the right. For the GTM model, each data point is plotted at the mean of its posterior distribution in latent space. The nonlinearity of the GTM model allows the separation between the groups of data points to be seen 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 probabilistic 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).

Section 6.4.

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

12.1 ($\star\star$) **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 $\mathbf{u}_1, \ldots, \mathbf{u}_M$, 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 $\mathbf{u}_1, \ldots, \mathbf{u}_M$ 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.

Appendix E

12.2 ($\star\star$) 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

$$\widetilde{J} = \mathrm{Tr}\left\{\widehat{\mathbf{U}}^{\mathrm{T}}\mathbf{S}\widehat{\mathbf{U}}\right\} + \mathrm{Tr}\left\{\mathbf{H}(\mathbf{I} - \widehat{\mathbf{U}}^{\mathrm{T}}\widehat{\mathbf{U}})\right\} \tag{12.93}$$

where $\widehat{\mathbf{U}}$ is a matrix of dimension $D \times (D - M)$ whose columns are given by $\mathbf{u}_i$. Now minimize $\widetilde{J}$ 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 $\widetilde{J}$ as the specific solution in which the columns of $\widehat{\mathbf{U}}$ are