6

ERROR FUNCTIONS

In previous chapters we have made use of the sum-of-squares error function, which was motivated primarily by analytical simplicity. There are many other possible choices of error function which can also be considered, depending on the particular application. In this chapter we shall describe a variety of different error functions and discuss their relative merits.

For regression problems we shall see that the basic goal is to model the conditional distribution of the output variables, conditioned on the input variables. This motivates the use of a sum-of-squares error function, and several important properties of this error function will be explored in some detail.

For classification problems the goal is to model the posterior probabilities of class membership, again conditioned on the input variables. Although the sum-of-squares error function can be used for classification (and can approximate the posterior probabilities) we shall see that there are other, more appropriate, error functions which can be considered. Generally speaking, Sections 6.1 to 6.4 are concerned with error functions for regression problems, while the remaining sections are concerned primarily with error functions for classification.

As we have stressed several times, the central goal in network training is not to memorize the training data, but rather to model the *underlying generator* of the data, so that the best possible predictions for the output vector $\mathbf{t}$ can be made when the trained network is subsequently presented with a new value for the input vector $\mathbf{x}$. The most general and complete description of the generator of the data is in terms of the probability density $p(\mathbf{x}, \mathbf{t})$ in the joint input-target space. For associative prediction problems of the kind we are considering, it is convenient to decompose the joint probability density into the product of the conditional density of the target data, conditioned on the input data, and the unconditional density of input data, so that

$$p(\mathbf{x}, \mathbf{t}) = p(\mathbf{t}|\mathbf{x})p(\mathbf{x}) \tag{6.1}$$

where $p(\mathbf{t}|\mathbf{x})$ denotes the probability density of $\mathbf{t}$ *given* that $\mathbf{x}$ takes a particular value, while $p(\mathbf{x})$ represents the unconditional density of $\mathbf{x}$ and is given by

$$p(\mathbf{x}) = \int p(\mathbf{t}, \mathbf{x})\, d\mathbf{t}. \tag{6.2}$$

The density $p(\mathbf{x})$ plays an important role in several aspects of neural networks, including procedures for choosing the basis function parameters in a radial basis function network (Section 5.9). However, for the purposes of making predictions of $\mathbf{t}$ for new values of $\mathbf{x}$, it is the conditional density $p(\mathbf{t}|\mathbf{x})$ which we need to model.

Most of the error functions which will be considered in this chapter can be motivated from the principle of maximum likelihood (Section 2.2). For a set of training data $\{\mathbf{x}^n, \mathbf{t}^n\}$, the likelihood can be written as

$$\mathcal{L} = \prod_n p(\mathbf{x}^n, \mathbf{t}^n) = \prod_n p(\mathbf{t}^n|\mathbf{x}^n)p(\mathbf{x}^n) \tag{6.3}$$

where we have assumed that each data point $(\mathbf{x}^n, \mathbf{t}^n)$ is drawn independently from the same distribution, and hence we can multiply the probabilities. Instead of maximizing the likelihood, it is generally more convenient to minimize the negative logarithm of the likelihood. These are equivalent procedures, since the negative logarithm is a monotonic function. We therefore minimize

$$E = -\ln \mathcal{L} = -\sum_n \ln p(\mathbf{t}^n|\mathbf{x}^n) - \sum_n \ln p(\mathbf{x}^n) \tag{6.4}$$

where E is called an *error function*. As we shall see, a feed-forward neural network can be regarded as a framework for modelling the conditional probability density $p(\mathbf{t}|\mathbf{x})$. The second term in (6.4) does not depend on the network parameters, and so represents an additive constant which can be dropped from the error function. We therefore have

$$E = -\sum_n \ln p(\mathbf{t}^n|\mathbf{x}^n). \tag{6.5}$$

Note that the error function takes the form of a sum over patterns of an error term for each pattern separately. This follows from the assumed independence of the data points under the given distribution. Different choices of error function arise from different assumptions about the form of the conditional distribution $p(\mathbf{t}|\mathbf{x})$. For interpolation problems, the targets $\mathbf{t}$ consist of continuous quantities whose values we are trying to predict, while for classification problems they represent labels defining class membership or, more generally, estimates of the probabilities of class membership.

6.1 Sum-of-squares error

Consider the case of c target variables t_k where $k = 1, \ldots, c$, and suppose that the distributions of the different target variables are independent, so that we can

write

$$p(\mathbf{t}|\mathbf{x}) = \prod_{k=1}^{c} p(t_k|\mathbf{x}). \tag{6.6}$$

We shall further assume that the distribution of the target data is Gaussian. More specifically, we assume that the target variable t_k is given by some deterministic function of $\mathbf{x}$ with added Gaussian noise ϵ, so that

$$t_k = h_k(\mathbf{x}) + \epsilon_k. \tag{6.7}$$

We now assume that the errors ϵ_k have a normal distribution with zero mean, and standard a deviation σ which does not depend on $\mathbf{x}$ or on k. Thus, the distribution of ϵ_k is given by

$$p(\epsilon_k) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{\epsilon_k^2}{2\sigma^2}\right). \tag{6.8}$$

We now seek to model the functions $h_k(\mathbf{x})$ by a neural network with outputs $y_k(\mathbf{x};\mathbf{w})$ where $\mathbf{w}$ is the set of weight parameters governing the neural network mapping. Using (6.7) and (6.8) we see that the probability distribution of target variables is given by

do we assume that our model is accurate?

$$p(t_k|\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{\{y_k(\mathbf{x};\mathbf{w}) - t_k\}^2}{2\sigma^2}\right) \tag{6.9}$$

where we have replaced the unknown function $h_k(\mathbf{x})$ by our model $y_k(\mathbf{x};\mathbf{w})$. Together with (6.6) and (6.5) this leads to the following expression for the error function

$$E = \frac{1}{2\sigma^2} \sum_{n=1}^{N} \sum_{k=1}^{c} \{y_k(\mathbf{x}^n;\mathbf{w}) - t_k^n\}^2 + Nc \ln\sigma + \frac{Nc}{2} \ln(2\pi). \tag{6.10}$$

We note that, for the purposes of error minimization, the second and third terms on the right-hand side of (6.10) are independent of the weights $\mathbf{w}$ and so can be omitted. Similarly, the overall factor of $1/\sigma^2$ in the first term can also be omitted. We then finally obtain the familiar expression for the sum-of-squares error function

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} \{y_k(\mathbf{x}^n;\mathbf{w}) - t_k^n\}^2 \tag{6.11}$$

$$= \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}^n; \mathbf{w}) - \mathbf{t}^n\|^2 . \tag{6.12}$$

Having found a set of values $\mathbf{w}^*$ for the weights which minimizes the error, the optimum value for σ can then by found by minimization of E in (6.10) with respect to σ. This minimization is easily performed analytically with the explicit, and intuitive, result

$$\sigma^2 = \frac{1}{Nc} \sum_{n=1}^{N} \sum_{k=1}^{c} \{y_k(\mathbf{x}^n; \mathbf{w}^*) - t_k^n\}^2 \tag{6.13}$$

which says that the optimal value of σ^2 is proportional to the residual value of the sum-of-squares error function at its minimum. We shall return to this result later.

We have derived the sum-of-squares error function from the principle of maximum likelihood on the assumption of Gaussian distributed target data. Of course the use of a sum-of-squares error does not *require* the target data to have a Gaussian distribution. Later in this chapter we shall consider the least-squares solution for an example problem with a strongly non-Gaussian distribution. However, as we shall see, if we use a sum-of-squares error, then the results we obtain cannot distinguish between the true distribution and any other distribution having the same mean and variance.

Note that it is sometimes convenient to assess the performance of networks using a different error function from that used to train them. For instance, in an interpolation problem the networks might be trained using a sum-of-squares error function of the form

$$E = \frac{1}{2} \sum_{n} \|\mathbf{y}(\mathbf{x}^n; \mathbf{w}) - \mathbf{t}^n\|^2 \tag{6.14}$$

where the sum runs over all N patterns in the training set, whereas for network testing it would be more convenient to use a root-mean-square (RMS) error of the form

$$E^{\mathrm{RMS}} = \frac{\sum_n \|\mathbf{y}(\mathbf{x}^n; \mathbf{w}^*) - \mathbf{t}^n\|^2}{\sum_n \|\mathbf{t}^n - \bar{\mathbf{t}}\|^2} \tag{6.15}$$

where $\mathbf{w}^*$ denotes the weight vector of the trained network, and the sums now run over the N' patterns in the test set. Here $\bar{\mathbf{t}}$ is defined to be the average test set target vector

$$\bar{\mathbf{t}} = \frac{1}{N'} \sum_{n=1}^{N'} \mathbf{t}^n. \tag{6.16}$$

The RMS error (6.15) has the advantage, unlike (6.14), that its value does not grow with the size of the data set. If it has a value of unity then the network is predicting the test data 'in the mean' while a value of zero means perfect prediction of the test data.

6.1.1 *Linear output units*

The mapping function of a multi-layer perceptron or a radial basis function network can be written in the form

$$y_k(\mathbf{x}; \mathbf{w}) = g(a_k) \tag{6.17}$$

$$a_k = \sum_{j=0}^{M} w_{kj} z_j(\mathbf{x}; \tilde{\mathbf{w}}) \tag{6.18}$$

where $g(\cdot)$ denotes the activation function of the output units, $\{w_{kj}\}$ denotes the set of weights (and biases) which connect directly to the output units, and $\tilde{\mathbf{w}}$ denotes the set of all other weights (and biases) in the network. The derivative of the sum-of-squares error (6.11) with respect to a_k can be written as

$$\frac{\partial E}{\partial a_k} = \sum_n g'(a_k^n)(y_k^n - t_k^n). \tag{6.19}$$

If we choose the activation function for the output units to be linear, $g(a) = a$, then this derivative takes a particularly simple form

$$\frac{\partial E}{\partial a_k} = \sum_n (y_k^n - t_k^n). \tag{6.20}$$

This allows the minimization with respect to the weights $\{w_{kj}\}$ (with the weights $\tilde{\mathbf{w}}$ held fixed) to be expressed as a linear optimization problem, which can be solved in closed form as discussed in Section 3.4.3. Here we shall follow a similar analysis, except that we shall find it convenient to make the bias parameters explicit and deal with them separately.

We first write the network mapping in the form

$$y_k = \sum_{j=1}^{M} w_{kj} z_j + w_{k0}. \tag{6.21}$$

Minimizing the sum-of-squares error (6.11) with respect to the biases first, we then obtain

$$\frac{\partial E}{\partial w_{k0}} = \sum_{n=1}^{N} \left\{ \sum_{j=1}^{c} w_{kj} z_j^n + w_{k0} - t_k^n \right\} = 0 \tag{6.22}$$

which can be solved explicitly for the biases to give

$$w_{k0} = \bar{t}_k - \sum_{j=1}^{M} w_{kj} \bar{z}_j \tag{6.23}$$

where we have defined the following average quantities:

$$\bar{t}_k = \frac{1}{N} \sum_{n=1}^{N} t_k^n, \qquad \bar{z}_j = \frac{1}{N} \sum_{n=1}^{N} z_j^n. \tag{6.24}$$

The result (6.23) shows that the role of the biases is to compensate for the difference between the averages (over the data set) of the target values, and the weighted sums of the averages of the hidden unit outputs.

If we back-substitute the expression (6.23) into the sum-of-squares error we obtain

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} \left\{ \sum_{j=1}^{M} w_{kj} \tilde{z}_j^n - \tilde{t}_k^n \right\}^2 \tag{6.25}$$

where we have defined

$$\tilde{t}_k^n = t_k^n - \bar{t}_k, \qquad \tilde{z}_j^n = z_j^n - \bar{z}_j. \tag{6.26}$$

We can now minimize this error with respect to the output weights w_{kj} to give

$$\frac{\partial E}{\partial w_{kj}} = \sum_{n=1}^{N} \left\{ \sum_{j'=1}^{M} w_{kj'} \tilde{z}_{j'}^n - \tilde{t}_k^n \right\} \tilde{z}_j^n = 0. \tag{6.27}$$

It is convenient at this point to introduce a matrix notation so that $(\mathbf{T})_{nk} = \tilde{t}_k^n$, $(\mathbf{W})_{kj} = w_{kj}$ and $(\mathbf{Z})_{nj} = \tilde{z}_j^n$. We can then write (6.27) in the form

$$\mathbf{Z}^{\mathrm{T}} \mathbf{Z} \mathbf{W}^{\mathrm{T}} - \mathbf{Z}^{\mathrm{T}} \mathbf{T} = 0 \tag{6.28}$$

where $\mathbf{Z}^{\mathrm{T}}$ denotes the transpose of $\mathbf{Z}$. We can write an explicit solution for the weight matrix as

$$\mathbf{W}^{\mathrm{T}} = \mathbf{Z}^{\dagger}\mathbf{T} = 0 \tag{6.29}$$

where $\mathbf{Z}^{\dagger}$ is the pseudo-inverse of the matrix $\mathbf{Z}$ given by

$$\mathbf{Z}^{\dagger} \equiv (\mathbf{Z}^{\mathrm{T}}\mathbf{Z})^{-1}\mathbf{Z}^{\mathrm{T}}. \tag{6.30}$$

Here we have assumed that the matrix $(\mathbf{Z}^{\mathrm{T}}\mathbf{Z})$ is non-singular. A more general discussion of the properties of the pseudo-inverse can be found in Section 3.4.3. For a single-layer network, this represents the optimal solution for the weights, which can therefore be calculated explicitly. In the present case, however, this expression for the weights depends on the activations of the hidden units which themselves depend on the weights $\tilde{\mathbf{w}}$. Thus, as the weights $\tilde{\mathbf{w}}$ change during learning, so the optimal values for the weights $\{w_{kj}\}$ will also change. Nevertheless, it is still possible to exploit the linear nature of the partial optimization with respect to the output unit weights as part of an overall strategy for error minimization, as discussed in Section 7.3.

6.1.2 *Linear sum rules*

The use of a sum-of-squares error function to determine the weights in a network with linear output units implies an interesting sum rule for the network outputs (Lowe and Webb, 1991). Suppose that the target patterns used to train the network satisfy an exact linear relation, so that for each pattern n we have

$$\mathbf{u}^{\mathrm{T}}\mathbf{t}^n + u_0 = 0 \tag{6.31}$$

where $\mathbf{u}$ and u_0 are constants. We now show that, if the final-layer weights are determined by the optimal least-squares procedure outlined above, then the outputs of the network will satisfy the same linear constraint for arbitrary input patterns.

Summing over all patterns n in (6.31) we find that the average target vector $\bar{\mathbf{t}}$ satisfies the relation $u_0 = -\mathbf{u}^{\mathrm{T}}\bar{\mathbf{t}}$ where the components of $\bar{\mathbf{t}}$ are given by (6.24). Thus, the linear relation (6.31) can be written in the form

$$\mathbf{u}^{\mathrm{T}}\mathbf{t}^n = \mathbf{u}^{\mathrm{T}}\bar{\mathbf{t}}. \tag{6.32}$$

The network outputs, given by (6.21), can be written in vector notation as

$$\mathbf{y} = \mathbf{W}\mathbf{z} + \mathbf{w}_0. \tag{6.33}$$

Similarly, the solution for the optimal biases given by (6.23) can be written as

$$\mathbf{w}_0 = \bar{\mathbf{t}} - \mathbf{W}\bar{\mathbf{z}}. \tag{6.34}$$

Now consider the scalar product of $\mathbf{y}$ with the vector $\mathbf{u}$, for an arbitrary input pattern. Using the optimal weights given by (6.29), together with (6.33) and (6.34), we have

$$\begin{aligned}\mathbf{u}^{\mathrm{T}}\mathbf{y} &= \mathbf{u}^{\mathrm{T}}(\mathbf{w}_0 + \mathbf{W}\mathbf{z}) \\ &= \mathbf{u}^{\mathrm{T}}\bar{\mathbf{t}} + \mathbf{u}^{\mathrm{T}}\mathbf{T}^{\mathrm{T}}(\mathbf{Z}^{\dagger})^{\mathrm{T}}(\mathbf{z} - \bar{\mathbf{z}})\end{aligned} \tag{6.35}$$

where we have used the following property of matrix transposes $(\mathbf{AB})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}$. From (6.32), however, it follows that

$$(\mathbf{u}^{\mathrm{T}}\mathbf{T}^{\mathrm{T}})_n = \mathbf{u}^{\mathrm{T}}\tilde{\mathbf{t}}^n = \mathbf{u}^{\mathrm{T}}(\mathbf{t}^n - \bar{\mathbf{t}}) = 0 \tag{6.36}$$

where we have used the linear constraint (6.32). Combining (6.35) and (6.36) we obtain

$$\mathbf{u}^{\mathrm{T}}\mathbf{y} = \mathbf{u}^{\mathrm{T}}\bar{\mathbf{t}} \tag{6.37}$$

and so the network outputs exactly satisfy the same linear sum rule as the target data. We shall see an application of this result in the next section. More generally, if a set of targets satisfies several linear constraints simultaneously, then so will the outputs of the network (Exercise 6.3).

6.1.3 *Interpretation of network outputs*

We next derive an important result for the interpretation of the outputs of a network trained by minimizing a sum-of-squares error function. In particular, we will show that the outputs approximate the conditional averages of the target data. This is a central result which has several important consequences for practical applications of neural networks. An understanding of its implications can help to avoid some common mistakes, and lead to more effective use of network network techniques.

Consider the limit in which the size N of the training data set goes to infinity. In this limit we can replace the finite sum over patterns in the sum-of-squares error with an integral of the form

$$E = \lim_{N\to\infty} \frac{1}{2N} \sum_{n=1}^{N} \sum_{k} \{y_k(\mathbf{x}^n; \mathbf{w}) - t_k^n\}^2 \tag{6.38}$$

$$= \frac{1}{2} \sum_{k} \iint \{y_k(\mathbf{x}; \mathbf{w}) - t_k\}^2 \, p(t_k, \mathbf{x}) \, dt_k \, d\mathbf{x} \tag{6.39}$$

where we have introduced an extra factor of $1/N$ into the definition of the sum-of-squares error in order to make the limiting process meaningful. We now factor the joint distributions $p(t_k, \mathbf{x})$ into the product of the unconditional density function for the input data $p(\mathbf{x})$, and the target data density conditional on the input vector $p(t_k|\mathbf{x})$, as in (6.1), to give

$$E = \frac{1}{2} \sum_k \int\int \{y_k(\mathbf{x}; \mathbf{w}) - t_k\}^2 \, p(t_k|\mathbf{x}) p(\mathbf{x}) \, dt_k \, d\mathbf{x}. \tag{6.40}$$

Next we define the following conditional averages of the target data

$$\langle t_k|\mathbf{x}\rangle \equiv \int t_k p(t_k|\mathbf{x}) \, dt_k \tag{6.41}$$

$$\langle t_k^2|\mathbf{x}\rangle \equiv \int t_k^2 p(t_k|\mathbf{x}) \, dt_k. \tag{6.42}$$

We now write the term in brackets in (6.40) in the form

$$\{y_k - t_k\}^2 = \{y_k - \langle t_k|\mathbf{x}\rangle + \langle t_k|\mathbf{x}\rangle - t_k\}^2 \tag{6.43}$$

$$= \{y_k - \langle t_k|\mathbf{x}\rangle\}^2 + 2\{y_k - \langle t_k|\mathbf{x}\rangle\}\{\langle t_k|\mathbf{x}\rangle - t_k\} + \{\langle t_k|\mathbf{x}\rangle - t_k\}^2 \tag{6.44}$$

Next we substitute (6.44) into (6.40) and make use of (6.41) and (6.42). The second term on the right-hand side of (6.44) then vanishes as a consequence of the integration over t_k. The sum-of-squares error can then be written in the form

$$E = \frac{1}{2} \sum_k \int \{y_k(\mathbf{x}; \mathbf{w}) - \langle t_k|\mathbf{x}\rangle\}^2 p(\mathbf{x}) \, d\mathbf{x} + \frac{1}{2} \sum_k \int \{\langle t_k^2|\mathbf{x}\rangle - \langle t_k|\mathbf{x}\rangle^2\} p(\mathbf{x}) \, d\mathbf{x}. \tag{6.45}$$

We now note that the second term in (6.45) is independent of the network mapping function $y_k(\mathbf{x}; \mathbf{w})$ and hence is independent of the network weights $\mathbf{w}$. For the purposes of determining the network weights by error minimization, this term can be neglected. Since the integrand in the first term in (6.45) is non-negative, the absolute minimum of the error function occurs when this first term vanishes, which corresponds to the following result for the network mapping

$$y_k(\mathbf{x}; \mathbf{w}^*) = \langle t_k|\mathbf{x}\rangle \tag{6.46}$$

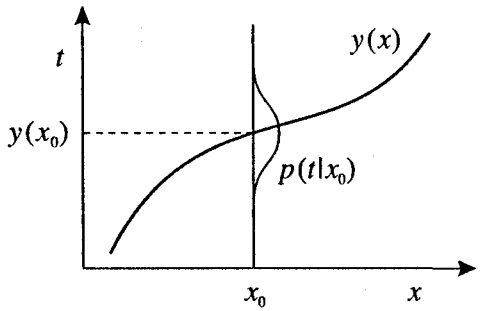

Figure 6.1. A schematic illustration of the property (6.46) that the network mapping which minimizes a sum-of-squares error function is given by the conditional average of the target data. Here we consider a mapping from a single input variable x to a single target variable t. At any given value x_0 of the input variable, the network output $y(x_0)$ is given by the average of t with respect to the distribution $p(t|x_0)$ of the target variable, for that value of x.

where $\mathbf{w}^*$ is the weight vector at the minimum of the error function. Equation (6.46) is a key result and says that the network mapping is given by the conditional average of the target data, in other words by the *regression* of t_k conditioned on $\mathbf{x}$. This result is illustrated schematically in Figure 6.1, and by a simple example in Figure 6.2.

Before discussing the consequences of this important result we note that it is dependent on three key assumptions. First, the data set must be sufficiently large that it approximates an infinite data set. Second, the network function $y_k(\mathbf{x};\mathbf{w})$ must be sufficiently general that there exists a choice of parameters which makes the first term in (6.45) sufficiently small. This second requirement implies that the number of adaptive weights (or equivalently the number of hidden units) must be sufficiently large. It is important that the two limits of large data set and large number of weights must be approached in a coupled way in order to achieve the desired result. This important issue is discussed in Section 9.1 in the context of generalization and the trade-off between bias and variance. The third caveat is that the optimization of the network parameters is performed in such a way as to find the appropriate minimum of the cost function. Techniques for parameter optimization in neural networks are discussed in Chapter 7.

Note that the derivation of the result (6.46) did not depend on the choice of network architecture, or even whether we were using a neural network at all. It only required that the representation for the non-linear mapping be sufficiently general. The importance of neural networks is that they provide a practical framework for approximating arbitrary non-linear multivariate mappings, and can therefore in principle approximate the conditional average to arbitrary accuracy.

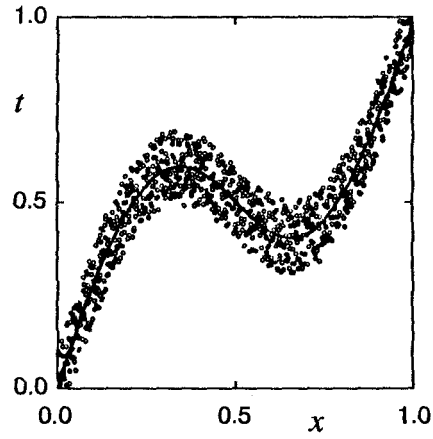

Figure 6.2. A simple example of a network mapping which approximates the *conditional* average of the target data (shown by the circles) generated from the function $t = x + 0.3\sin(2\pi x) + \epsilon$ where ϵ is a random variable drawn from a uniform distribution in the range $(-0.1, 0.1)$. The solid curve shows the result of training a multi-layer perceptron network with five hidden units using a sum-of-squares error function. The network approximates the conditional average of the target data, which gives a good *representation* of the function from which the data was generated.

We can easily see why the minimum of a sum-of-squares error is given by the average value of the target data by considering the simple error function

$$E(y) = (y - a)^2 + (y - b)^2 \tag{6.47}$$

where a and b are constants. Differentiation of $E(y)$ with respect to y shows that the minimum occurs at

$$y^{\min} = (a + b)/2 \tag{6.48}$$

In other words, the minimum is given by the average of the target data. The more general property (6.46) is simply the extension of this result to conditional averages.

We can also derive (6.46) in a more direct way as follows. If we take the sum-of-squares error in the form (6.39) and set the functional derivative (Appendix D) of E with respect to $y_k(\mathbf{x})$ to zero we obtain

$$\frac{\delta E}{\delta y_k(\mathbf{x})} = \int \{y_k(\mathbf{x}) - t_k\} p(t_k|\mathbf{x}) p(\mathbf{x})\, dt_k = 0. \tag{6.49}$$

If we make use of (6.41) we then obtain (6.46) directly. The use of a functional derivative here is equivalent to the earlier assumption that the class of functions $y_k(\mathbf{x})$ is very general.

For many regression problems, the form of network mapping given by the conditional average (6.46) can be regarded as optimal. If the data is generated from a set of deterministic functions $h_k(\mathbf{x})$ with superimposed zero-mean noise ϵ_k then the target data is given by

$$t_k^n = h_k(\mathbf{x}^n) + \epsilon_k^n. \tag{6.50}$$

The network outputs, given by the conditional averages of the target data, then take the form

$$y_k(\mathbf{x}) = \langle t_k|\mathbf{x}\rangle = \langle h_k(\mathbf{x}) + \epsilon_k|\mathbf{x}\rangle = h_k(\mathbf{x}) \tag{6.51}$$

since $\langle \epsilon^n \rangle = 0$. Thus the network has averaged over the noise on the data and discovered the underlying deterministic function. Not all regression problems are as simple as this, however, as we shall see later.

Note that the first integral in (6.45) is weighted by the unconditional density $p(\mathbf{x})$. We therefore see that the network function $y_k(\mathbf{x})$ pays a significant penalty for departing from the conditional average $\langle t_k|\mathbf{x}\rangle$ in regions of input space where the density $p(\mathbf{x})$ of input data is high. In regions where $p(\mathbf{x})$ is small, there is little penalty if the network output is a poor approximation to the conditional average. This forms the basis of a simple procedure for assigning error bars to network predictions, based on an estimate of the density $p(\mathbf{x})$ (Bishop, 1994b).

If we return to (6.45) we see that the second term can be written in the form

$$\frac{1}{2} \sum_k \int \sigma_k^2(\mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \tag{6.52}$$

where $\sigma_k^2(\mathbf{x})$ represents the variance of the target data, as a function of $\mathbf{x}$, and is given by

$$\sigma_k^2(\mathbf{x}) = \langle t_k^2|\mathbf{x}\rangle - \langle t_k|\mathbf{x}\rangle^2 \tag{6.53}$$

$$= \langle (t_k - \langle t_k|\mathbf{x}\rangle)^2|\mathbf{x}\rangle \tag{6.54}$$

$$= \int \{t_k - \langle t_k|\mathbf{x}\rangle\}^2 p(t_k|\mathbf{x})\, dt_k. \tag{6.55}$$

If the network mapping function is given by the conditional average (6.46), so

that the first term in (6.45) vanishes, then the residual error is given by (6.52). The value of the residual error is therefore be a measure of the average variance of the target data. This is equivalent to the earlier result (6.13) obtained for a finite data set. It should be emphasized, however, that these are *biased* estimates of the variance, as discussed in Section 2.2, and so they should be treated with care in practical applications.

We originally derived the sum-of-squares error function from the principle of maximum likelihood by assuming that the distribution of the target data could be described by a Gaussian function with an $\mathbf{x}$-dependent mean, and a single global variance parameter. As we noted earlier, the sum-of-squares error does not require that the distribution of target variables be Gaussian. If a sum-of-squares error is used, however, the quantities which can be determined are the $\mathbf{x}$-dependent mean of the distribution (given by the outputs of the trained network) and a global averaged variance (given by the residual value of the error function at its minimum). Thus, the sum-of-squares error function cannot distinguish between the true distribution, and a Gaussian distribution having the same $\mathbf{x}$-dependent mean and average variance.

6.1.4 *Outer product approximation for the Hessian*

In Section 4.10.2 we discussed a particular approximation to the Hessian matrix (the matrix of second derivatives of the error function with respect to the network weights) for a sum-of-squares error function. This approximation is based on a sum of outer products of first derivatives. Here we show that the approximation is exact in the infinite data limit, provided we are at the global minimum of the error function. Consider the error function in the form (6.45). Taking the second derivatives with respect to two weights w_r and w_s we obtain

$$\frac{\partial^2 E}{\partial w_r \partial w_s} = \sum_k \int \left\{ \frac{\partial y_k}{\partial w_r} \frac{\partial y_k}{\partial w_s} \right\} p(\mathbf{x})\, d\mathbf{x} + \sum_k \int \left\{ \frac{\partial^2 y_k}{\partial w_r \partial w_s} (y_k - \langle t_k | \mathbf{x} \rangle) \right\} p(\mathbf{x})\, d\mathbf{x}. \tag{6.56}$$

Using the result (6.46) that the outputs $y_k(\mathbf{x})$ of the trained network represent the conditional averages of the target data, we see that the second term in (6.56) vanishes. The Hessian is therefore given by an integral of terms involving only the products of first derivatives. For a finite data set, we can write this result in the form

$$\frac{\partial^2 E}{\partial w_r \partial w_s} = \frac{1}{N} \sum_{n=1}^{N} \sum_k \frac{\partial y_k^n}{\partial w_r} \frac{\partial y_k^n}{\partial w_s}. \tag{6.57}$$

6.1.5 *Inverse problems*

The fact that a least-squares solution approximates the conditional average of the target data has an important consequence when neural networks are used to solve *inverse* problems. Many potential applications of neural networks fall into this category. Examples include the analysis of spectral data, tomographic reconstruction, control of industrial plant, and robot kinematics. For such problems there exists a well-defined *forward* problem which is characterized by a *functional* (i.e. single-valued) mapping. Often this corresponds to causality in a physical system. In the case of spectral reconstruction, for example, the forward problem corresponds to the evaluation of the spectrum when the parameters (locations, widths and amplitudes) of the spectral lines are prescribed. In practical applications we generally have to solve the corresponding inverse problem in which the roles of input and output variables are interchanged. In the case of spectral analysis, this corresponds to the determination of the spectral line parameters from an observed spectrum. For inverse problems, the mapping can be often be multi-valued, with values of the inputs for which there are several valid values for the outputs. For example, there may be several choices for the spectral line parameters which give rise to the same observed spectrum. If a least-squares approach is applied to an inverse problem, it will approximate the conditional average of the target data, and this will frequently lead to extremely poor performance (since the average of several solutions is not necessarily itself a solution).

As a simple illustration of this problem, consider the data set shown earlier in Figure 6.2 where we saw how a network which approximates the conditional average of the target data gives a good representation of the underlying generator of the data. Suppose we now reverse the roles of the input and target variables. Figure 6.3 shows the result of training a network of the same type as before on the same data set, but with input and output variables interchanged. The network again tries to approximate the conditional average of the target data, but this time the conditional average gives a very poor description of the generator of the data. The problem can be traced to the intermediate values of x in Figure 6.3 where the target data is multi-valued. Predictions made by the trained network in this region can be very poor. The problem cannot be solved by modifying the network architecture or the training algorithm, since it is a fundamental consequence of using a sum-of-squares error function. For problems involving many input and output variables, where visualization of the data is not straightforward, it can be very difficult to ascertain whether there are regions of input space for which the target data is multi-valued. One approach to such problems is to go beyond the Gaussian description of the distribution of target variables, and to find a more general model for the conditional density, as will be discussed in Section 6.4.

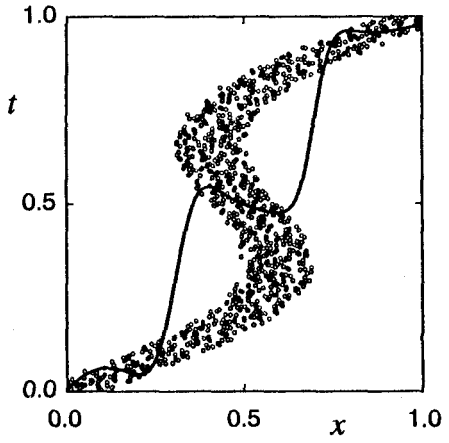

Figure 6.3. An illustration of the problem which can arise when a least-squares approach is applied to an inverse problem. This shows the same data set as in Figure 6.2 but with the roles of input and output variables interchanged. The solid curve shows the result of training the same neural network as in Figure 6.2, again using a sum-of-squares error. This time the network gives a very poor fit to the data, as it again tries to represent the conditional average of the target values.

6.2 Minkowski error

We have derived the sum-of-squares error function from the principle of maximum likelihood on the assumption of a Gaussian distribution of target data. We can obtain more general error functions by considering a generalization of the Gaussian distribution of the form

$$p(\epsilon) = \frac{R\beta^{1/R}}{2\Gamma(1/R)} \exp\left(-\beta|\epsilon|^R\right) \tag{6.58}$$

where $\Gamma(a)$ is the gamma function (defined on page 28), the parameter β controls the variance of the distribution, and the pre-factor in (6.58) ensures that $\int p(\epsilon)\, d\epsilon = 1$. For the case of $R = 2$ this distribution reduces to a Gaussian. We now consider the negative log-likelihood of a data set, given by (6.5) and (6.6), under the distribution (6.58). Omitting irrelevant constants, we obtain an error function of the form

$$E = \sum_n \sum_{k=1}^{c} |y_k(\mathbf{x}^n; \mathbf{w}) - t_k^n|^R \tag{6.59}$$

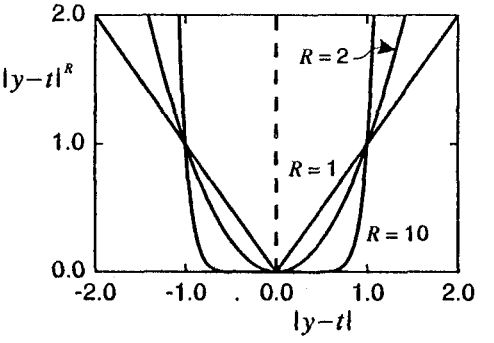

Figure 6.4. Plot of the function $|y-t|^R$ against $|y-t|$ for various values of R. This function forms the basis for the definition of the Minkowski-R error measure.

called the *Minkowski-R* error. This reduces to the usual sum-of-squares error when $R = 2$. For the case of $R = 1$, the distribution function (6.58) is a *Laplacian*, and the corresponding Minkowski-R measure (6.59) is called the *city block* metric (because the distance between two points on a plane measured by this metric is equal to the Euclidean distance covered by moving between the two points along segments of lines parallel to the axes, as if moving along blocks in a city). More generally, the distance metric $|y-t|^R$ is known as the L_R norm. The function $|y-t|^R$ is plotted against $|y-t|$ for various values of R in Figure 6.4.

The derivatives of the Minkowski-R error function with respect to the weights in the network are given by

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \sum_k |y_k(\mathbf{x}^n; \mathbf{w}) - t_k^n|^{R-1} \text{sign}(y_k(\mathbf{x}^n; \mathbf{w}) - t_k^n) \frac{\partial y_k^n}{\partial w_{ji}}. \tag{6.60}$$

These derivatives can be evaluated using the standard back-propagation procedure, discussed in Section 4.8. Examples of the application of the Minkowski-R error to networks trained using back-propagation are given in Hanson and Burr (1988) and Burrascano (1991).

One of the potential difficulties of the standard sum-of-squares error is that it receives the largest contributions from the points which have the largest errors. If there are long tails on the distributions then the solution can be dominated by a very small number of points called *outliers* which have particularly large errors. This is illustrated by a simple example in Figure 6.5.

A similarly severe problem can also arise from incorrectly labelled data. For instance, one single data point for which the target value has been incorrectly labelled by a large amount can completely invalidate the least-squares solution.

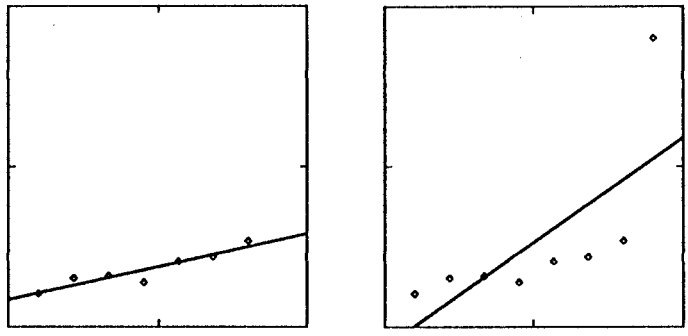

Figure 6.5. Example of fitting a linear polynomial through a set of noisy data points by minimizing a sum-of-squares error. In (a) the line gives a good representation of the systematic aspects of the data. In (b) a single extra data point has been added which lies well away from the other data points, showing how it dominates the fitting of the line.

Techniques which attempt to solve this problem are referred to as *robust statistics*, and a review in the context of conventional statistical methods can be found in Huber (1981). The use of the Minkowski error with an R value less than 2 reduces the sensitivity to outliers. For instance, with $R = 1$, the minimum error solution computes the conditional *median* of the data, rather than the conditional mean (Exercise 6.5). The reason for this can be seen by considering the simple error

$$E(y) = \sum_n |y - t^n|. \tag{6.61}$$

Minimizing $E(y)$ with respect to y gives

$$\sum_n \text{sign}(y - t^n) = 0 \tag{6.62}$$

which is satisfied when y is the median of the points $\{t^n\}$ (i.e. the value for which the same number of points t^n have values greater than y as have values less than y). If one of the t^n is taken to some very large value, this has no effect on the solution for y.

6.3 Input-dependent variance

So far we have assumed that the variance of the target data can be described by a single global parameter σ. In many practical applications, this will be a poor assumption, and we now discuss more general models for the target data distribution. The sum-of-squares error is easily extended to allow each output to be described by its own variance parameter σ_k. More generally, we might wish to determine how the variance of the data depends on the input vector $\mathbf{x}$ (Nix and Weigend, 1994). This can be done by adopting a more general description for the conditional distribution of the target data, and then writing down the negative log-likelihood in order to obtain a suitable error function. Thus, we write the conditional distribution of the target variables in the form

$$p(t_k|\mathbf{x}) = \frac{1}{(2\pi)^{1/2}\sigma_k(\mathbf{x})} \exp\left(-\frac{\{y_k(\mathbf{x};\mathbf{w}) - t_k\}^2}{2\sigma_k^2(\mathbf{x})}\right). \tag{6.63}$$

Forming the negative logarithm of the likelihood function as before, and omitting additive constants, we obtain

$$E = \sum_{n=1}^{N}\sum_{k}\left(\ln\sigma_k(\mathbf{x}^n) + \frac{\{y_k(\mathbf{x}^n) - t_k^n\}^2}{2\sigma_k^2(\mathbf{x}^n)}\right). \tag{6.64}$$

If we now multiply by $1/N$ as before, and take the infinite-data limit, we obtain the error function in the form

$$E = \sum_{k}\int\int\left(\ln\sigma_k(\mathbf{x}) + \frac{\{y_k(\mathbf{x}) - t_k\}^2}{2\sigma_k^2(\mathbf{x})}\right) p(t_k|\mathbf{x})p(\mathbf{x})\,dt_k\,d\mathbf{x}. \tag{6.65}$$

The functions $\sigma_k(\mathbf{x})$ can be modelled by adding further outputs to the neural network. We shall not consider this approach further, as it is a special case of a much more general technique for modelling the full conditional distribution, which will be discussed shortly.

An alternative approach to determining an input-dependent variance (Satchwell, 1994) is based on the result (6.46) that the network mapping which minimizes a sum-of-squares error is given by the conditional expectation of the target data. First a network is trained in the usual way by minimizing a sum-of-squares error in which the t_k^n form the targets. The outputs of this network, when presented with the training data input vectors $\mathbf{x}^n$, correspond to the conditional averages of the target data. These averages are subtracted from the target values and the results are then squared and used as targets for a second network which is also trained using a sum-of-squares error function. The outputs of this network then represent the conditional averages of $\{t_k - \langle t_k|\mathbf{x}\rangle\}^2$ and thus approximate the variances $\sigma_k^2(\mathbf{x})$ given by (6.55).

This procedure can be justified directly as follows. Consider the infinite data

limit again, for which we can write the error function in the form (6.65). If we again assume that the functions $y_k(\mathbf{x})$ and $\sigma_k(\mathbf{x})$ have unlimited flexibility then we can first *minimize* E with respect to the y_k by functional differentiation to give

$$\frac{\delta E}{\delta y_k(\mathbf{x})} = 0 = p(\mathbf{x}) \int \frac{\{y_k(\mathbf{x}) - t_k\}}{\sigma_k^2(\mathbf{x})} p(t_k|\mathbf{x})\, dt_k \tag{6.66}$$

which, after some rearrangement, gives the standard result

$$y_k(\mathbf{x}) = \langle t_k|\mathbf{x}\rangle \tag{6.67}$$

as before. We can similarly minimize E independently with respect to the functions $\sigma_k(\mathbf{x})$ to give

$$\frac{\delta E}{\delta \sigma_k(\mathbf{x})} = 0 = p(\mathbf{x}) \int \left(\frac{1}{\sigma_k(\mathbf{x})} - \frac{\{y_k(\mathbf{x}) - t_k\}^2}{\sigma_k(\mathbf{x})^3} \right) p(t_k|\mathbf{x})\, dt_k \tag{6.68}$$

which is easily solved for $\sigma_k^2(\mathbf{x})$ to give

$$\sigma_k^2(\mathbf{x}) = \langle \{t_k - \langle t_k|\mathbf{x}\rangle\}^2|\mathbf{x}\rangle \tag{6.69}$$

where we have used (6.67). We can then interpret (6.69) in terms of the two-stage two-network approach described above. This technique is simple and can make use of standard neural network software. Its principal limitation is that it still assumes a Gaussian form for the distribution function (since it makes use only of the second-order statistics of the target data).

Since these approaches are based on maximum likelihood, they will give a biased estimate of the variances as discussed above, and so will tend to underestimate the true variance. In extreme cases, such methods can discover pathological solutions in which the variance goes to zero, corresponding to an infinite likelihood, as discussed in the context of unconditional density estimation in Section 2.5.5.

6.4 Modelling conditional distributions

We can view the basic goal in training a feed-forward neural network as that of modelling the statistical properties of the generator of the data, expressed in terms of a conditional distribution function $p(\mathbf{t}|\mathbf{x})$. For the sum-of-squares error function, this corresponds to modelling the conditional distribution of the target data in terms of a Gaussian distribution with a global variance parameter and an $\mathbf{x}$-dependent mean. However, if the data has a complex structure, as for example in Figure 6.3, then this particular choice of distribution can lead to a very poor representation of the data. We therefore seek a general framework for modelling conditional probability distributions.

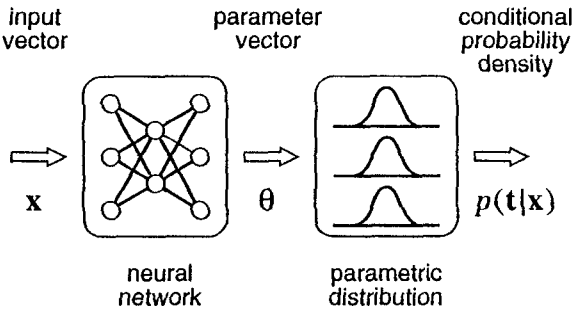

Figure 6.6. We can represent general conditional probability densities $p(\mathbf{t}|\mathbf{x})$ by considering a parametric model for the distribution of $\mathbf{t}$ whose parameters are determined by the outputs of a neural network which takes $\mathbf{x}$ as its input vector.

In Chapter 2 we discussed a number of parametric techniques for modelling unconditional distributions. Suppose we use one of these techniques to model the distribution $p(\mathbf{t}|\boldsymbol{\theta})$ of target variables $\mathbf{t}$, where $\boldsymbol{\theta}$ denotes the set of parameters which govern the model distribution. If we allow the parameters $\boldsymbol{\theta}$ to be functions of the input vector $\mathbf{x}$, then we can model conditional distributions. We can achieve this by letting the components of $\boldsymbol{\theta}(\mathbf{x})$ be given by the outputs of a feed-forward neural network which takes $\mathbf{x}$ as input. This leads to the combined density model and neural network structure shown in Figure 6.6. Provided we consider a sufficiently general density model, and a sufficiently flexible network, we have a framework for approximating arbitrary conditional distributions.

For different choices of the parametric model, we obtain different representations for the conditional densities. For example, a single Gaussian model for $p(\mathbf{t}|\boldsymbol{\theta})$ corresponds to the procedure described above in Section 6.3. Another possibility is to use a linear combination of a fixed set of kernel functions. In this case the outputs of the network represent the coefficients in the linear combination (Bishop and Legleye, 1995), and we must ensure that the coefficients are positive and sum to one in order to preserve the positivity and normalization of the conditional density. We do not discuss this approach further as it is a special case of the more general technique which we consider next.

A powerful, general framework for modelling unconditional distributions, based on the use of *mixture models*, was introduced in Section 2.6. Mixture models represent a distribution in terms of a linear combination of adaptive kernel functions. If we apply this technique to the problem of modelling conditional distributions we have

$$p(\mathbf{t}|\mathbf{x}) = \sum_{j=1}^{M} \alpha_j(\mathbf{x}) \phi_j(\mathbf{t}|\mathbf{x}) \tag{6.70}$$

where M is the number of components, or kernels, in the mixture. The parameters $\alpha_j(\mathbf{x})$ are called *mixing coefficients*, and can be regarded as *prior* probabilities (conditioned on $\mathbf{x}$) of the target vector $\mathbf{t}$ having been generated from the jth component of the mixture. Note that the mixing coefficients are taken to be functions of the input vector $\mathbf{x}$. The function $\phi_j(\mathbf{t}|\mathbf{x})$ represents the conditional density of the target vector $\mathbf{t}$ for the jth kernel. Various choices for the kernel functions are possible. As in Chapter 2, however, we shall restrict attention to kernel functions which are Gaussian of the form

$$\phi_j(\mathbf{t}|\mathbf{x}) = \frac{1}{(2\pi)^{c/2}\sigma_j^c(\mathbf{x})} \exp\left\{-\frac{\|\mathbf{t} - \boldsymbol{\mu}_j(\mathbf{x})\|^2}{2\sigma_j^2(\mathbf{x})}\right\} \tag{6.71}$$

where the vector $\boldsymbol{\mu}_j(\mathbf{x})$ represents the centre of the jth kernel, with components μ_{jk}, and c is the dimensionality of $\mathbf{t}$. In (6.71) we have assumed that the components of the output vector are statistically independent within each of the kernel functions, and can be described by a common variance $\sigma_j^2(\mathbf{x})$. This assumption can be relaxed in a straightforward way by introducing full covariance matrices for each Gaussian kernel, at the expense of a more complex formalism. In principle, however, such a complication is not necessary, since a Gaussian mixture model, with kernels given by (6.71), can approximate any given density function to arbitrary accuracy, provided the mixing coefficients and the Gaussian parameters (means and variances) are correctly chosen (McLachlan and Basford, 1988). Thus, the representation given by (6.70) and (6.71) is completely general. In particular, it does not assume that the components of $\mathbf{t}$ are statistically independent, in contrast to the single-Gaussian representation used in (6.6) and (6.9) to derive the sum-of-squares error.

For any given value of $\mathbf{x}$, the mixture model (6.70) provides a general formalism for modelling an arbitrary conditional density function $p(\mathbf{t}|\mathbf{x})$. We now take the various parameters of the mixture model, namely the mixing coefficients $\alpha_j(\mathbf{x})$, the means $\boldsymbol{\mu}_j(\mathbf{x})$ and the variances $\sigma_j^2(\mathbf{x})$, to be governed by the outputs of a conventional neural network which takes $\mathbf{x}$ as its input. This technique was introduced in the form of the *mixture-of-experts* model (Jacobs *et al.*, 1991) described in Section 9.7, and has since been discussed by other authors (Bishop, 1994a; Liu, 1994; Neuneier *et al.*, 1994). By choosing a mixture model with a sufficient number of kernel functions, and a neural network with a sufficient number of hidden units, this model can approximate as closely as desired any conditional density function $p(\mathbf{t}|\mathbf{x})$. The original motivation for the mixture-of-experts model was to provide a mechanism for partitioning the solution to a problem between several networks. This was achieved by using a separate network to determine the parameters of each kernel function, with a further network to determine the

mixing coefficients. For some applications this modular approach offers a number of advantages, and is discussed further in Section 9.7.

The neural network in Figure 6.6 can be any standard feed-forward network structure with universal approximation capabilities. Here we consider a multilayer perceptron, with a single hidden layer of sigmoidal units and an output layer of linear units. For M components in the mixture model (6.70), the network will have M outputs denoted by z_j^α which determine the mixing coefficients, M outputs denoted by z_j^σ which determine the kernel widths σ_j, and $M \times c$ outputs denoted by z_{jk}^μ which determine the components μ_{jk} of the kernel centres $\boldsymbol{\mu}_j$. The total number of network outputs is given by $(c+2) \times M$, as compared with the usual c outputs for a network used with a sum-of-squares error function.

In order to ensure that the mixing coefficients $\alpha_j(\mathbf{x})$ can be interpreted as probabilities, they must satisfy the constraints

$$\sum_{j=1}^{M} \alpha_j(\mathbf{x}) = 1 \tag{6.72}$$

$$0 \leq \alpha_j(\mathbf{x}) \leq 1. \tag{6.73}$$

The first constraint also ensures that the distribution is correctly normalized, so that $\int p(\mathbf{t}|\mathbf{x})\, d\mathbf{t} = 1$. These constraints can be satisfied by choosing $\alpha_j(\mathbf{x})$ to be related to the corresponding networks outputs by a *softmax* function (Bridle, 1990; Jacobs *et al.*, 1991)

$$\alpha_j = \frac{\exp(z_j^\alpha)}{\sum_{l=1}^{M} \exp(z_l^\alpha)}. \tag{6.74}$$

We shall encounter the softmax function again in the next section when we discuss error functions for classification problems.

The variances σ_j represent *scale* parameters and so it is convenient to represent them in terms of the exponentials of the corresponding network outputs

$$\sigma_j = \exp(z_j^\sigma). \tag{6.75}$$

In a Bayesian framework (Exercise 10.13) this would correspond to the choice of a *non-informative prior*, assuming the corresponding network outputs z_j^σ had uniform probability distributions (Jacobs *et al.*, 1991; Nowlan and Hinton, 1992). This representation also has the additional benefit of helping to avoid pathological configurations in which one or more of the variances goes to zero, since this would require the corresponding $z_j^\sigma \to -\infty$. The possibility of such results is discussed in Section 2.6.1 in the context of mixture models for unconditional density estimation.

The centres $\boldsymbol{\mu}_j$ represent *location* parameters, and again the notion of a non-informative prior (Exercise 10.12) suggests that these be represented directly by the network outputs

$$\mu_{jk} = z_{jk}^{\mu}. \tag{6.76}$$

As before, we can construct an error function from the likelihood by using (6.5) to give

$$E = -\sum_n \ln \left\{ \sum_{j=1}^{M} \alpha_j(\mathbf{x}^n) \phi_j(\mathbf{t}^n | \mathbf{x}^n) \right\} \tag{6.77}$$

with $\phi_j(\mathbf{t}|\mathbf{x})$ given by (6.71). The minimization of this error function with respect to the parameters of the neural network leads to a model for the conditional density of the target data. From this density function, any desired statistic involving the output variables can in principle be computed.

In order to minimize the error function, we need to calculate the derivatives of the error E with respect to the weights in the neural network. These can be evaluated by using the standard back-propagation procedure, provided we obtain suitable expressions for the derivatives of the error with respect to the outputs of the network. Since the error function (6.77) is composed of a sum of terms $E = \sum_n E^n$, one for each pattern, we can consider the derivatives $\delta_k^n = \partial E^n / \partial z_k$ for a particular pattern n and then find the derivatives of E by summing over all patterns. Note that, since the network output units have linear activation functions $g(a) = a$, the quantities δ_k^n can also be written as $\partial E^n / \partial a_k$, and so are equivalent to the 'errors' introduced in the discussion of error back-propagation in Section 4.8. These errors can be back-propagated through the network to find the derivatives with respect to the network weights.

We have already remarked that the ϕ_j can be regarded as conditional density functions, with prior probabilities α_j. As with the mixture models discussed in Section 2.6, it is convenient to introduce the corresponding *posterior* probabilities, which we obtain using Bayes' theorem,

$$\pi_j(\mathbf{x}, \mathbf{t}) = \frac{\alpha_j \phi_j}{\sum_{l=1}^{M} \alpha_l \phi_l}, \tag{6.78}$$

as this leads to some simplification of the subsequent analysis. Note that, from (6.78), the posterior probabilities sum to unity:

$$\sum_{j=1}^{M} \pi_j = 1. \tag{6.79}$$

Consider first the derivatives of E^n with respect to those network outputs which correspond to the mixing coefficients α_j. Using (6.77) and (6.78) we obtain

$$\frac{\partial E^n}{\partial \alpha_k} = -\frac{\pi_k}{\alpha_k}. \tag{6.80}$$

We now note that, as a result of the softmax transformation (6.74), the value of α_k depends on all of the network outputs which contribute to the mixing coefficients, and so differentiating (6.74) we have

$$\frac{\partial \alpha_k}{\partial z_j^\alpha} = \delta_{jk}\alpha_k - \alpha_j\alpha_k. \tag{6.81}$$

From the chain rule we have

$$\frac{\partial E^n}{\partial z_j^\alpha} = \sum_k \frac{\partial E^n}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial z_j^\alpha}. \tag{6.82}$$

Combining (6.80), (6.81) and (6.82) we then obtain

$$\frac{\partial E^n}{\partial z_j^\alpha} = \alpha_j - \pi_j \tag{6.83}$$

where we have used (6.79).

For the derivatives corresponding to the σ_j parameters we make use of (6.77) and (6.78), together with (6.71), to give

$$\frac{\partial E^n}{\partial \sigma_j} = -\pi_j \left\{ \frac{\|\mathbf{t} - \boldsymbol{\mu}_j\|^2}{\sigma_j^3} - \frac{c}{\sigma_j} \right\}. \tag{6.84}$$

Using (6.75) we have

$$\frac{\partial \sigma_j}{\partial z_j^\sigma} = \sigma_j. \tag{6.85}$$

Combining these together we then obtain

$$\frac{\partial E^n}{\partial z_j^\sigma} = -\pi_j \left\{ \frac{\|\mathbf{t} - \boldsymbol{\mu}_j\|^2}{\sigma_j^2} - c \right\}. \tag{6.86}$$

Finally, since the parameters μ_{jk} are given directly by the z_{jk}^μ network outputs, we have, using (6.77) and (6.78), together with (6.71),

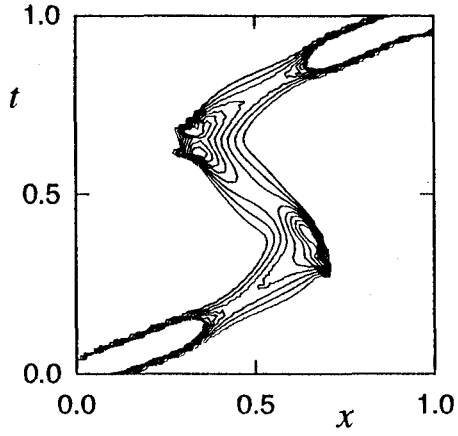

Figure 6.7. Plot of the contours of the conditional probability density of the target data obtained from a multi-layer perceptron network trained using the same data as in Figure 6.3, but using the error function (6.77). The network has three Gaussian kernel functions, and uses a two-layer multi-layer perceptron with five 'tanh' sigmoidal units in the hidden layer, and nine outputs.

$$\frac{\partial E^n}{\partial z_{jk}^{\mu}} = \pi_j \left\{ \frac{\mu_{jk} - t_k}{\sigma_j^2} \right\}. \tag{6.87}$$

An example of the application of these techniques to the estimation of conditional densities is given in Figure 6.7, which shows the contours of conditional density corresponding to the data set shown in Figure 6.3.

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, the model is able to produce a conditional density which is unimodal for some values of x and trimodal for other values, as in Figure 6.7, by modulating the amplitudes of the mixing components, or priors, $\alpha_j(\mathbf{x})$. This can be seen in Figure 6.8 which shows plots of the three priors $\alpha_j(x)$ as functions of x. It can be seen that for $x = 0.2$ and $x = 0.8$ only one of the three kernels has a non-zero prior probability. At $x = 0.5$, however, all three kernels have significant priors.

Once the 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 which 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, given by

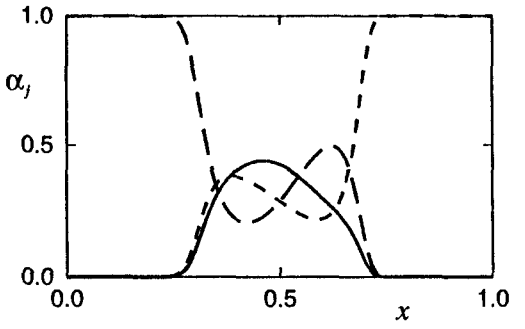

Figure 6.8. Plot of the priors $\alpha_j(x)$ as a function of x for the three kernel functions from the network used to plot Figure 6.7. At both small and large values of x, where the conditional probability density of the target data is unimodal, only one of the kernels has a prior probability which differs significantly from zero. At intermediate values of x, where the conditional density is trimodal, the three kernels have comparable priors.

$$\langle \mathbf{t}|\mathbf{x}\rangle = \int \mathbf{t} p(\mathbf{t}|\mathbf{x})\, d\mathbf{t} \tag{6.88}$$

$$= \sum_j \alpha_j(\mathbf{x}) \int \mathbf{t}\phi_j(\mathbf{t}|\mathbf{x})\, d\mathbf{t} \tag{6.89}$$

$$= \sum_j \alpha_j(\mathbf{x})\boldsymbol{\mu}_j(\mathbf{x}) \tag{6.90}$$

where we have used (6.70) and (6.71). This is equivalent to the function computed by a standard network trained by least squares, and so this network can reproduce the conventional least-squares result as a special case. We can likewise evaluate the variance of the density function about the conditional average, to give

$$s^2(\mathbf{x}) = \left\langle \|\mathbf{t} - \langle \mathbf{t}|\mathbf{x}\rangle\|^2 |\mathbf{x}\right\rangle \tag{6.91}$$

$$= \sum_j \alpha_j(\mathbf{x}) \left\{ \sigma_j(\mathbf{x})^2 + \left\| \boldsymbol{\mu}_j(\mathbf{x}) - \sum_l \alpha_l(\mathbf{x})\boldsymbol{\mu}_l(\mathbf{x}) \right\|^2 \right\} \tag{6.92}$$

where we have used (6.70), (6.71) and (6.90). This is more general than the corresponding least-squares result since this variance is allowed to be a general function of $\mathbf{x}$. Similar results can be obtained for other moments of the condi-

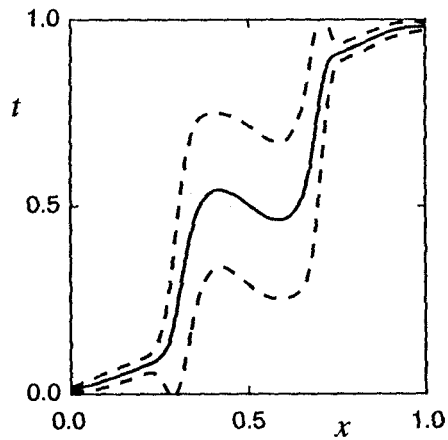

Figure 6.9. This shows a plot of $\langle t|x \rangle$ against x (solid curve) calculated from the conditional density in Figure 6.7 using (6.90), together with corresponding plots of $\langle t|x \rangle \pm s(x)$ (dashed curves) obtained using (6.92).

tional distribution. Plots of the mean and variance, obtained from the conditional distribution in Figure 6.7, are shown in Figure 6.9.

For some applications, the distribution of the target data will consist of a limited number of distinct branches, as is the case for the data shown in Figure 6.3. In such cases we may be interested in finding an output value corresponding to just one of the branches (as would be the case in many control applications for example). The most probable branch is the one which has the greatest associated 'probability mass'. Since each component of the mixture model is normalized, $\int \phi_j(\mathbf{t}|\mathbf{x})\, d\mathbf{t} = 1$, the most probable branch of the solution, assuming the components are well separated and have negligible overlap, is given by

$$\arg\max_j \{\alpha_j(\mathbf{x})\}\,. \tag{6.93}$$

In the mixture-of-experts model (Jacobs *et al.*, 1991) this corresponds to selecting the output of one of the component network modules. The required value of $\mathbf{t}$ is then given by the corresponding centre $\boldsymbol{\mu}_j$. Figure 6.10 shows the most probable branch of the solution, as a function of x, for the same network as used to plot Figure 6.7.

Again, one of the limitations of using maximum likelihood techniques to determine variance-like quantities such as the σ_j, is that it is biased (Section 2.2). In particular, it tends to underestimate the variance in regions where there is limited data.

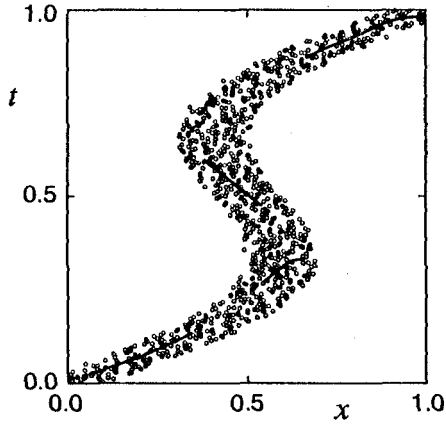

Figure 6.10. Plot of the central value of the most probable kernel as a function of x from the network used to plot Figure 6.7. This gives a discontinuous functional mapping from x to t which at every value of x lies well inside a region of significant probability density. The diagram should be compared with the corresponding continuous mapping in Figure 6.3 obtained from standard least squares.

6.4.1 *Periodic variables*

So far we have considered the problem of 'regression' for variables which live on the real axis $(-\infty, \infty)$. However, a number of applications involve angle-like output variables which live on a finite interval, usually $(0, 2\pi)$ and which are intrinsically periodic. Due to the periodicity, the techniques described so far cannot be applied directly. Here we show how the general framework discussed above can be extended to estimate the conditional distribution $p(\theta|\mathbf{x})$ of a periodic variable θ, conditional on an input vector $\mathbf{x}$ (Bishop and Legleye, 1995).

The approach is again based on a mixture of kernel functions , but in this case the kernel functions themselves are periodic, thereby ensuring that the overall density function will be periodic. To motivate this approach, consider the problem of modelling the distribution of a velocity vector $\mathbf{v}$ in two dimensions. Since $\mathbf{v}$ lives in a Euclidean plane, we can model the density function $p(\mathbf{v})$ using a mixture of conventional spherical Gaussian kernels, where each kernel has the form

$$\phi(v_x, v_y) = \frac{1}{2\pi\sigma^2} \exp\left\{ -\frac{(v_x - \mu_x)^2}{2\sigma^2} - \frac{(v_y - \mu_y)^2}{2\sigma^2} \right\} \tag{6.94}$$

where (v_x, v_y) are the Cartesian components of $\mathbf{v}$, and (μ_x, μ_y) are the components of the centre $\boldsymbol{\mu}$ of the kernel. From this we can extract the conditional

distribution of the polar angle θ of the vector $\mathbf{v}$, given a value for $v = \|\mathbf{v}\|$. This is easily done with the transformation $v_x = v\cos\theta$, $v_y = v\sin\theta$, and defining θ_0 to be the polar angle of $\boldsymbol{\mu}$, so that $\mu_x = \mu\cos\theta_0$ and $\mu_y = \mu\sin\theta_0$, where $\mu = \|\boldsymbol{\mu}\|$. This leads to a distribution which can be written in the form

$$\phi(\theta) = \frac{1}{2\pi I_0(m)} \exp\{m\cos(\theta - \theta_0)\} \tag{6.95}$$

where the normalization coefficient has been expressed in terms of the zeroth-order modified Bessel function of the first kind, $I_0(m)$. The distribution (6.95) is known as a *circular normal* or *von Mises* distribution (Mardia, 1972). The parameter m (which depends on v in our derivation) is analogous to the (inverse) variance parameter in a conventional normal distribution. Since (6.95) is periodic, we can construct a general representation for the conditional density of a periodic variable by considering a mixture of circular normal kernels, with parameters governed by the outputs of a neural network. The weights in the network can again be found by maximizing the likelihood function defined over a set of training data.

An example of the application of these techniques to the determination of wind direction from satellite radar scatterometer data is given in Bishop and Legleye (1995). This is an inverse problem in which the target data is multi-valued. For problems involving periodic variables in which the target data is effectively single-valued with respect to the input vector, then a single circular normal kernel can be used.

An alternative approach to modelling conditional distributions of periodic variables is discussed in Exercise 6.8.

6.5 Estimating posterior probabilities

So far in this chapter we have focused on 'regression' problems in which the target variable are continuous. We now turn to a consideration of error functions for classification problems in which the target variables represent discrete class labels (or, more generally, the probabilities of class membership).

When we use a neural network to solve a classification problem, there are two distinct ways in which we can view the objectives of network training. At the simpler level, we can arrange for the network to represent a non-linear discriminant function so that, when a new input vector is presented to the trained network, the outputs provide a classification directly. The second approach, which is more general and more powerful, is to use the network to model the posterior probabilities of class membership. Typically there is one output unit for each possible class, and the activation of each output unit represents the corresponding posterior probability $p(C_k|\mathbf{x})$, where C_k is the kth class, and $\mathbf{x}$ is the input vector. These probabilities can then be used in a subsequent decision-making stage to arrive at a classification.

By arranging for the network outputs to approximate posterior probabilities, we can exploit a number of results which are not available if the network is

used simply as a non-linear discriminant (Richard and Lippmann, 1991). These include:

Minimum error-rate decisions

From the discussion of optimal classification in Section 1.9 we know that, to minimize the probability of misclassification, a new input vector should be assigned to the class having the largest posterior probability. Note that the network outputs need not be close to 0 or 1 if the class-conditional density functions are overlapping. Heuristic procedures, such as applying extra training using those patterns which fail to generate outputs close to the target values, will be counterproductive, since this alters the distributions and makes it less likely that the network will generate the correct Bayesian probabilities.

Outputs sum to 1

Since the network outputs approximate posterior probabilities they should sum to unity. This can be enforced explicitly as part of the choice of network structure as we shall see. Also, the average of each network output over all patterns in the training set should approximate the corresponding prior class probabilities, since

$$P(\mathcal{C}_k) = \int P(\mathcal{C}_k|\mathbf{x})p(\mathbf{x})\,d\mathbf{x} \simeq \frac{1}{N}\sum_n P(\mathcal{C}_k|\mathbf{x}^n). \tag{6.96}$$

These estimated priors can be compared with the sample estimates of the priors obtained from the fractions of patterns in each class within the training data set. Differences between these two estimates are an indication that the network is not modelling the posterior probabilities accurately (Richard and Lippmann, 1991).

Compensating for different prior probabilities

In some of the conventional approaches to pattern classification discussed in Chapter 1, the posterior probabilities were expressed through Bayes' theorem in the form

$$P(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)P(\mathcal{C}_k)}{p(\mathbf{x})} \tag{6.97}$$

and the prior probabilities $P(\mathcal{C}_k)$ and class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ were estimated separately. The neural network approach, by contrast, provides direct estimates of the posterior probabilities. Sometimes the prior probabilities expected when the network is in use differ from those represented by the training set. It is then it is a simple matter to use Bayes' theorem (6.97) to make the necessary corrections to the network outputs. This is achieved simply by dividing the network outputs by the prior probabilities corresponding to the training set, multiplying them by the new

prior probabilities, and then normalizing the results. Changes in the prior probabilities can therefore be accommodated without re-training the network. The prior probabilities for the training set may be estimated simply by evaluating the fraction of the training set data points in each class. Prior probabilities corresponding to the network's operating environment can often be obtained very straightforwardly since only the class labels are needed and no input data is required. As an example, consider the problem of classifying medical images into 'normal' and 'tumour'. When used for screening purposes, we would expect a very small prior probability of 'tumour'. To obtain a good variety of tumour images in the training set would therefore require huge numbers of training examples. An alternative is to increase artificially the proportion of tumour images in the training set, and then to compensate for the different priors on the test data as described above. The prior probabilities for tumours in the general population can be obtained from medical statistics, without having to collect the corresponding images. Correction of the network outputs is then a simple matter of multiplication and division.

Combining the outputs of several networks

Rather than using a single network to solve a complete problem, there is often benefit in breaking the problem down into smaller parts and treating each part with a separate network. By dividing the network outputs by the prior probabilities used during training, the network outputs become likelihoods scaled by the unconditional density of the input vectors. These scaled likelihoods can be multiplied together on the assumption that the input vectors for the various networks are independent. Since the scaling factor is independent of class, a classifier based on the product of scaled likelihoods will give the same results as one based on the true likelihoods. This approach has been successfully applied to problems in speech recognition (Bourlard and Morgan, 1990; Singer and Lippmann, 1992).

Minimum risk

As discussed in Chapter 1, the goal of a classification system may not always be to minimize the probability of misclassification. Different misclassifications may carry different penalties, and we may wish to minimize the overall loss or risk (Section 1.10). Again the medical screening application provides a good example. It may be far more serious to mis-classify a tumour image as normal than to mis-classify a normal image as that of a tumour. In this case, the posterior probabilities from the network can be combined with a suitable matrix of loss coefficients to allow the minimum risk decision to be made. Again, no network re-training is required to achieve this. However, if the required loss matrix elements are known before the network is trained, then it may be better to modify the error function as will be discussed for the case of a sum-of-squares error in Section 6.6.2.

Rejection thresholds

In Section 1.10.1 we introduced the concept of a rejection threshold, which is such that if all of the posterior probabilities fall below this threshold then no classification decision is made. Alternative classification techniques can then be applied to the rejected cases. This reflects the costs associated with making the wrong decisions balanced against the cost of alternative classification procedures. In the medical image classification problem, for instance, it may be better not to try to classify doubtful images automatically, but instead to have a human expert provide a decision. Rejection of input vectors can be achieved in a principled way, provided the network outputs represent posterior probabilities of class membership.

In subsequent sections of this chapter we show how the outputs of a network can be interpreted as approximations to posterior probabilities, provided the error function used for network training is carefully chosen. We also show that some error functions allow networks to represent non-linear discriminants, even though the output values themselves need not correspond to probabilities.

6.6 Sum-of-squares for classification

In the previous section we showed that, for a network trained by minimizing a sum-of-squares error function, the network outputs approximate the conditional averages of the target data

$$y_k(\mathbf{x}) = \langle t_k | \mathbf{x} \rangle = \int t_k p(t_k|\mathbf{x})\, dt_k. \tag{6.98}$$

In the case of a classification problem, every input vector in the training set is labelled by its class membership, represented by a set of target values t_k^n. The targets can be chosen according to a variety of schemes, but the most convenient is the 1-of-c coding in which, for an input vector $\mathbf{x}^n$ from class $\mathcal{C}_l$, we have $t_k^n = \delta_{kl}$ where δ_{kl} is the Kronecker delta symbol defined on page xiii. In this case the target values are precisely known and the density function in target space becomes singular and can be written as

$$p(t_k|\mathbf{x}) = \sum_{l=1}^{c} \delta(t_k - \delta_{kl}) P(\mathcal{C}_l|\mathbf{x}) \tag{6.99}$$

since $P(\mathcal{C}_l|\mathbf{x})$ is the probability that $\mathbf{x}$ belongs to class $\mathcal{C}_l$. If we now substitute (6.99) into (6.98) we obtain

$$y_k(\mathbf{x}) = P(\mathcal{C}_k|\mathbf{x}) \tag{6.100}$$

so that the outputs of the network correspond to Bayesian posterior probabilities (White, 1989; Richard and Lippmann, 1991).

If the network outputs represent probabilities, then they should lie in the range $(0, 1)$ and should sum to 1. For a network with linear output units, trained by minimizing a sum-of-squares error function, it was shown in Section 6.1.2 that if the target values satisfy a linear constraint, then the network outputs will satisfy the same constraint for an arbitrary input vector. In the case of a 1-of-c coding scheme, the target values sum to unity for each pattern, and so the network outputs will also always sum to unity. However, there is no guarantee that they will lie in the range $(0, 1)$. In fact, the sum-of-squares error function is not the most appropriate for classification problems. It was derived from maximum likelihood on the assumption of Gaussian distributed target data. However, the target values for a 1-of-c coding scheme are binary, and hence far from having a Gaussian distribution. Later we discuss error measures which are more appropriate for classification problems. However, there are advantages in using a sum-of-squares error, including the fact that the determination of the output weights in a network represents a linear optimization problem. The significance of this result for radial basis function networks was described in Chapter 5. We therefore discuss the use of a sum-of-squares error for classification problems in more detail before considering alternative choices of error function.

For a two-class problem, the 1-of-c target coding scheme described above leads to a network with two output units, one for each class, whose activations represent the corresponding probabilities of class membership. An alternative approach, however, is to use a single output y and a target coding which sets $t^n = 1$ if $\mathbf{x}^n$ is from class $\mathcal{C}_1$ and $t^n = 0$ if $\mathbf{x}^n$ is from class $\mathcal{C}_2$. In this case, the distribution of target values is given by

$$p(t_k|\mathbf{x}) = \delta(t-1)P(\mathcal{C}_1|\mathbf{x}) + \delta(t)P(\mathcal{C}_2|\mathbf{x}). \tag{6.101}$$

Substituting this into (6.98) gives

$$y(\mathbf{x}) = P(\mathcal{C}_1|\mathbf{x}) \tag{6.102}$$

and so the network output $y(\mathbf{x})$ represents the posterior probability of the input vector $\mathbf{x}$ belonging to class $\mathcal{C}_1$. The corresponding probability for class $\mathcal{C}_2$ is then given by $P(\mathcal{C}_2|\mathbf{x}) = 1 - y(\mathbf{x})$.

6.6.1 *Interpretation of hidden units*

In Section 6.1.1 we derived the expression (6.29) for the final-layer weights which minimizes a sum-of-squares error, for networks with linear output units. By substituting this result back into the error function we obtain an expression in which the only adaptive parameters are those associated with hidden units, which we denote by $\widetilde{\mathbf{w}}$. This expression sheds light on the nature of the hidden unit representation which a network learns, and indicates why multi-layer non-linear neural networks can be effective as pattern classification systems (Webb and Lowe, 1990).

Writing (6.25) in matrix notation we obtain

$$E = \frac{1}{2}\mathrm{Tr}\{(\mathbf{Z}\mathbf{W}^{\mathrm{T}} - \mathbf{T})(\mathbf{Z}\mathbf{W}^{\mathrm{T}} - \mathbf{T})^{\mathrm{T}}\} \tag{6.103}$$

where $\mathbf{Z}$, $\mathbf{W}$ and $\mathbf{T}$ are defined on page 199. We now substitute the solution (6.29) for the optimal weights into (6.103) to give

$$E = \frac{1}{2}\mathrm{Tr}\{(\mathbf{Z}\mathbf{Z}^{\dagger}\mathbf{T} - \mathbf{T})(\mathbf{Z}\mathbf{Z}^{\dagger}\mathbf{T} - \mathbf{T})^{\mathrm{T}}\}. \tag{6.104}$$

By using some matrix manipulation (Exercise 6.9) we can write this in the form

$$E = \frac{1}{2}\mathrm{Tr}\{\mathbf{T}^{\mathrm{T}}\mathbf{T} - \mathbf{S}_B\mathbf{S}_T^{-1}\} \tag{6.105}$$

Here $\mathbf{S}_T$ is given by

$$\mathbf{S}_T = \mathbf{Z}^{\mathrm{T}}\mathbf{Z} = \sum_n (\mathbf{z}^n - \overline{\mathbf{z}})(\mathbf{z}^n - \overline{\mathbf{z}})^{\mathrm{T}} \tag{6.106}$$

and the components of $\overline{\mathbf{z}}$ are defined by (6.24). We see that this can be interpreted as the total covariance matrix for the activations at the output of the final layer of hidden units with respect to the training data set. Similarly, $\mathbf{S}_B$ in (6.105) is given by

$$\mathbf{S}_B = \mathbf{Z}^{\mathrm{T}}\mathbf{T}\mathbf{T}^{\mathrm{T}}\mathbf{Z} \tag{6.107}$$

which can be interpreted (as we shall see) as a form of between-class covariance matrix.

Since the first term in the curly brackets in (6.105) depends only on the target data it is independent of the remaining weights $\widetilde{\mathbf{w}}$ in the network. Thus, minimizing the sum-of-squares error is equivalent to maximizing a particular discriminant function defined with respect to the activations of the final-layer hidden units given by

$$J = \frac{1}{2}\mathrm{Tr}\{\mathbf{S}_B\mathbf{S}_T^{-1}\}. \tag{6.108}$$

Note that, if the matrix $\mathbf{S}_T$ is ill-conditioned, then the inverse matrix $\mathbf{S}_T^{-1}$ should be replaced by the pseudo-inverse $\mathbf{S}_T^{\dagger}$. The criterion (6.108) has a clear similarity to the Fisher discriminant function which is discussed in Section 3.6. Nothing here is specific to the multi-layer perceptron, or indeed to neural networks. The same result is obtained regardless of the functions $z_j(\mathbf{x}; \widetilde{\mathbf{w}})$ and applies to any generalized linear discriminant in which the basis functions contain adaptive

parameters.

The role played by the hidden units can now be stated as follows. The weights in the final layer are adjusted to produce an optimum discrimination of the classes of input vectors by means of a linear transformation. Minimizing the error of this linear discriminant requires that the input data undergo a non-linear transformation into the space spanned by the activations of the hidden units in such a way as to maximize the discriminant function given by (6.108).

Further insight into the nature of the matrix $\mathbf{S}_B$ is obtained by considering a particular target coding scheme. For the 1-of-c target coding scheme we can write (6.107) in the form (Exercise 6.10)

$$\mathbf{S}_B = \sum_k N_k^2 (\overline{\mathbf{z}}^k - \overline{\mathbf{z}})(\overline{\mathbf{z}}^k - \overline{\mathbf{z}})^{\mathrm{T}} \tag{6.109}$$

where N_k is the number of patterns in class C_k and $\overline{\mathbf{z}}^k$ is the mean activation vector of the hidden units for all training patterns in class C_k, and is defined by

$$\overline{\mathbf{z}}^k = \frac{1}{N_k} \sum_{n \in C_k} \mathbf{z}^n. \tag{6.110}$$

Note that $\mathbf{S}_B$ in (6.109) differs from the conventional between-class covariance matrix introduced in Section 3.6 by having factors of N_k^2 instead of N_k in the sum over classes. This represents a strong weighting of the feature extraction criterion in favour of classes with larger numbers of patterns. If there is a significant difference between the prior probabilities for the training and test data sets, then this effect may be undesirable, and we shall shortly see how to correct for it by modifying the sum-of-squares error measure. As discussed in Section 3.6, there are several ways to generalize Fisher's original two-class discriminant criterion to several classes, all of which reduce to the original Fisher result as a special case. In general, there is no way to decide which of these will yield the best results. For a two-class problem, the between-class covariance matrix given in (6.109) differs from the conventional one only by a multiplicative constant, so in this case the network criterion is equivalent to the original Fisher expression.

In earlier work, Gallinari *et al.* (1988, 1991) showed that, for a network of *linear* processing units with a 1-of-c target coding, the minimization of a sum-of-squares error gave a set of input-to-hidden weights which maximized a criterion which took the form of a ratio of determinants of between-class and total covariance matrices defined at the outputs of the hidden units. The results of Webb and Lowe (1990) contain this result as a special case.

6.6.2 *Weighted sum-of-squares*

We have seen that, for networks with linear output units, minimization of a sum-of-squares error at the network outputs maximizes a particular non-linear feature extraction criterion

$$J = \frac{1}{2}\mathrm{Tr}\{\mathbf{S}_B\mathbf{S}_T^{-1}\} \tag{6.111}$$

at the hidden units. For the 1-of-c coding scheme, the corresponding between-class covariance matrix, given by (6.109), contains coefficients which depend on N_k, the number of patterns in class $\mathcal{C}_k$. Thus, the hidden unit representation obtained by maximizing this discriminant function will only be optimal for a particular set of prior probabilities N_k/N. If the prior probabilities differ between training and test sets, then the feature extraction need not be optimal.

A related difficulty arises if there are different costs associated with different misclassifications, so that a general loss matrix needs to be considered. It has been suggested (Lowe and Webb, 1990, 1991) that modifications to the form of the sum-of-squares error to take account of the loss matrix can lead to improved feature extraction by the hidden layer, and hence to improved classification performance.

To deal with different prior probabilities between the training set and the test set, Lowe and Webb (1990) modify the sum-of-squares error by introducing a weighting factor κ_n for each pattern n so that the error function becomes

$$E = \frac{1}{2}\sum_n \sum_k \kappa_n \{y_k(\mathbf{x}^n) - t^n\}^2 \tag{6.112}$$

where the weighting factors are given by

$$\kappa_n = \frac{\widetilde{P}(\mathcal{C}_k)}{P_k} \qquad \text{for pattern } n \text{ in class } \mathcal{C}_k \tag{6.113}$$

where $\widetilde{P}(\mathcal{C}_k)$ is the prior probability of class $\mathcal{C}_k$ for the *test* data, and $P_k = N_k/N$ is the corresponding (sample estimate of the) prior probability for the training data. It is straightforward to show (Exercise 6.12) that the total covariance matrix $\mathbf{S}_T$ then becomes

$$\mathbf{S}_T = \sum_k \frac{\widetilde{P}(\mathcal{C}_k)}{P_k} \sum_{n \in \mathcal{C}_k} (\mathbf{z}^n - \overline{\mathbf{z}})(\mathbf{z}^n - \overline{\mathbf{z}})^{\mathrm{T}} \tag{6.114}$$

which is the sample-based estimate of the total covariance matrix for data with prior class probabilities $\widetilde{P}(\mathcal{C}_k)$. In (6.114) the $\overline{\mathbf{z}}$ are given by

$$\overline{\mathbf{z}} = \sum_k \frac{\widetilde{P}(\mathcal{C}_k)}{N_k} \sum_{n \in \mathcal{C}_k} \mathbf{z}^n \tag{6.115}$$

which again is the sample-based estimate of the value which $\overline{\mathbf{z}}$ would take for data having the prior probabilities $\widetilde{P}(\mathcal{C}_k)$. Similarly, assuming a 1-of-c target

coding scheme, the between-class covariance matrix is modified to become

$$\mathbf{S}_B = \sum_k N^2 \widetilde{P}(\mathcal{C}_k)^2 (\overline{\mathbf{z}}^k - \overline{\mathbf{z}})(\overline{\mathbf{z}}^k - \overline{\mathbf{z}})^{\mathrm{T}} \tag{6.116}$$

which is the sample-based estimate of the between-class covariance matrix for data with prior probabilities $\widetilde{P}(\mathcal{C}_k)$.

The effects of an arbitrary loss matrix can similarly be taken into account by modifying the target coding scheme so that, for a pattern n which is labelled as belonging to class $\mathcal{C}_l$, the target vector has components $t_k^n = 1 - L_{lk}$, where L_{lk} represents the loss in assigning a pattern from class $\mathcal{C}_l$ to class $\mathcal{C}_k$. The total covariance matrix is unaltered, while the between-class covariance matrix becomes (Exercise 6.13)

$$\mathbf{S}_B = \sum_k \left\{ \sum_l (1 - L_{lk}) N_l (\overline{\mathbf{z}}_l - \overline{\mathbf{z}}) \right\} \left\{ \sum_{l'} (1 - L_{l'k}) N_{l'} (\overline{\mathbf{z}}_{l'} - \overline{\mathbf{z}})^{\mathrm{T}} \right\} \tag{6.117}$$

which reduces to the usual expression when $L_{lk} = 1 - \delta_{lk}$. Examples of the application of these techniques to a problem in medical prognosis are given in Lowe and Webb (1990).

6.7 Cross-entropy for two classes

We have seen that, for a 1-of-c target coding scheme, the outputs of a network trained by minimizing a sum-of-squares error function approximate the posterior probabilities of class membership, conditioned on the input vector. However, the sum-of-squares error was obtained from the maximum likelihood principle by assuming the target data was generated from a smooth deterministic function with added Gaussian noise. This is clearly a sensible starting point for regression problems. For classification problems, however, the targets are binary variables, and the Gaussian noise model does not provide a good description of their distribution. We therefore seek more appropriate choices of error function.

To start with, we consider problems involving two classes. One approach to such problems would be to use a network with two output units, one for each class. This type of representation is discussed in Section 6.9. Here we discuss an alternative approach in which we consider a network with a single output y. We would like the value of y to represent the posterior probability $P(\mathcal{C}_1|\mathbf{x})$ for class $\mathcal{C}_1$. The posterior probability of class $\mathcal{C}_2$ will then by given by $P(\mathcal{C}_2|\mathbf{x}) = 1 - y$. This can be achieved if we consider a target coding scheme for which $t = 1$ if the input vector belongs to class $\mathcal{C}_1$ and $t = 0$ if it belongs to class $\mathcal{C}_2$. We can combine these into a single expression, so that the probability of observing either target value is

$$p(t|\mathbf{x}) = y^t (1 - y)^{1-t} \tag{6.118}$$

which is a particular case of the binomial distribution called the Bernoulli distribution. With this interpretation of the output unit activations, the likelihood of observing the training data set, assuming the data points are drawn independently from this distribution, is then given by

$$\prod_n (y^n)^{t^n}(1-y^n)^{1-t^n}. \tag{6.119}$$

As usual, it is more convenient to minimize the negative logarithm of the likelihood. This leads to the *cross-entropy* error function (Hopfield, 1987; Baum and Wilczek, 1988; Solla *et al.*, 1988; Hinton, 1989; Hampshire and Pearlmutter, 1990) in the form

$$E = -\sum_n \{t^n \ln y^n + (1-t^n)\ln(1-y^n)\}. \tag{6.120}$$

We shall discuss the meaning of the term 'entropy' in Section 6.10. For the moment let us consider some elementary properties of this error function.

Differentiating the error function with respect to y^n we obtain

$$\frac{\partial E}{\partial y^n} = \frac{(y^n - t^n)}{y^n(1-y^n)}. \tag{6.121}$$

The absolute minimum of the error function occurs when

$$y^n = t^n \qquad \text{for all } n. \tag{6.122}$$

In Section 3.1.3 we showed that, for a network with a single output $y = g(a)$ whose value is to be interpreted as a probability, it is appropriate to consider the logistic activation function

$$g(a) = \frac{1}{1+\exp(-a)} \tag{6.123}$$

which has the property

$$g'(a) = g(a)(1-g(a)). \tag{6.124}$$

Combining (6.121) and (6.124) we see that the derivative of the error with respect to a takes the simple form

$$\delta^n \equiv \frac{\partial E}{\partial a^n} = y^n - t^n. \tag{6.125}$$

Here δ^n is the 'error' quantity which is back-propagated through the network in order to compute the derivatives of the error function with respect to the network weights (Section 4.8). Note that (6.125) has the same form as obtained for the sum-of-squares error function and linear output units. We see that there is a natural pairing of error function and output unit activation function which gives rise to this simple form for the derivative. Use of the logistic form of activation function also leads to corresponding simplifications when evaluating the Hessian matrix (the matrix of second derivatives of the error function).

From (6.120) and (6.122), the value of the cross-entropy error function at its minimum is given by

$$E_{\min} = -\sum_n \{t^n \ln t^n + (1 - t^n) \ln(1 - t^n)\}. \tag{6.126}$$

For the 1-of-c coding scheme this vanishes. However, the error function (6.120) is also the correct one to use when t^n is a continuous variable in the range $(0, 1)$ representing the probability of the input vector $\mathbf{x}^n$ belonging to class C_1 (see Section 6.10 and Exercise 6.15). In this case the minimum value (6.126) of the error need not vanish, and so it is convenient to subtract off this value from the original error function to give a modified error of the form

$$E = -\sum_n \left\{ t^n \ln \frac{y^n}{t^n} + (1 - t^n) \ln \frac{(1 - y^n)}{(1 - t^n)} \right\}. \tag{6.127}$$

Since (6.126) is independent of the network outputs this does not affect the location of the minimum and so has no effect on network training. The modified error (6.127) always has its minimum at 0, irrespective of the particular training set.

As a simple illustration of the interpretation of network outputs as probabilities, we consider a simple two-class problem with one input variable in which the class-conditional densities are given by the Gaussian mixture functions shown in Figure 6.11. A multi-layer perceptron with five hidden units having 'tanh' activation functions, and one output unit having a logistic sigmoid activation function, was trained by minimizing a cross-entropy error using 100 cycles of the BFGS quasi-Newton algorithm (Section 7.10). The resulting network mapping function is shown, along with the true posterior probability calculated using Bayes' theorem, in Figure 6.12.

6.7.1 *Sigmoid activation functions*

In Section 3.1.3, the logistic sigmoid activation function was motivated for a single-layer network by the goal of ensuring that the network outputs represent posterior probabilities, with the assumption that the class-conditional densities can be approximated by normal distributions. We can apply a similar argument to the network outputs in the case of multi-layered networks (Rumelhart *et*

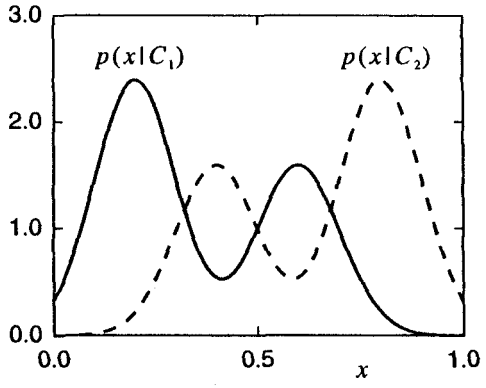

Figure 6.11. Plots of the class-conditional densities used to generate a data set to demonstrate the interpretation of network outputs as posterior probabilities. A total of 2000 data points were generated from these densities, using equal prior probabilities.

al., 1995). In this case we need to consider the distributions of the outputs of the hidden units, represented here by the vector $\mathbf{z}$ for the two classes. We can generalize the discussion by assuming that these class-conditional densities are described by

$$p(\mathbf{z}|\mathcal{C}_k) = \exp\left\{A(\boldsymbol{\theta}_k) + B(\mathbf{z}, \boldsymbol{\phi}) + \boldsymbol{\theta}_k^{\mathrm{T}}\mathbf{z}\right\} \tag{6.128}$$

which is a member of the *exponential family* of distributions (which includes many of the common distributions as special cases such as Gaussian, binomial, Bernoulli, Poisson, and so on). The parameters $\boldsymbol{\theta}_k$ and $\boldsymbol{\phi}$ control the form of the distribution. In writing (6.128) we are implicitly assuming that the distributions differ only in the parameters $\boldsymbol{\theta}_k$ and not in $\boldsymbol{\phi}$. An example would be two Gaussian distributions with different means, but with common covariance matrices.

Using Bayes' theorem, we can write the posterior probability for class $\mathcal{C}_1$ in the form

$$\begin{aligned} P(\mathcal{C}_1|\mathbf{z}) &= \frac{p(\mathbf{z}|\mathcal{C}_1)P(\mathcal{C}_1)}{p(\mathbf{z}|\mathcal{C}_1)P(\mathcal{C}_1) + p(\mathbf{z}|\mathcal{C}_2)P(\mathcal{C}_2)} \\ &= \frac{1}{1+\exp(-a)} \end{aligned} \tag{6.129}$$

which is a logistic sigmoid function, in which

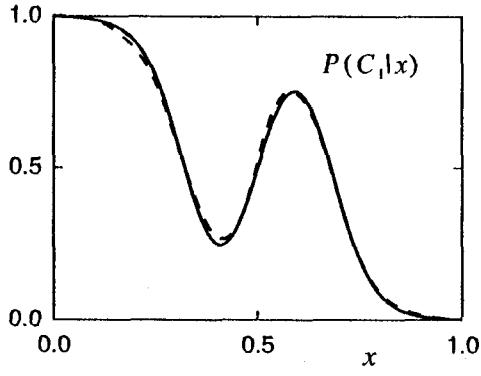

Figure 6.12. The result of training a multi-layer perceptron on data generated from the density functions in Figure 6.11. The solid curve shows the output of the trained network as a function of the input variable x, while the dashed curve shows the true posterior probability $P(C_1|x)$ calculated from the class-conditional densities using Bayes' theorem.

$$a = \ln \frac{p(\mathbf{z}|C_1)P(C_1)}{p(\mathbf{z}|C_2)P(C_2)} \tag{6.130}$$

Using (6.128) we can write this in the form

$$a = \mathbf{w}^{\mathrm{T}}\mathbf{z} + w_0 \tag{6.131}$$

where we have defined

$$\mathbf{w} = \boldsymbol{\theta}_1 - \boldsymbol{\theta}_2 \tag{6.132}$$

$$w_0 = A(\boldsymbol{\theta}_1) - A(\boldsymbol{\theta}_2) + \ln \frac{P(C_1)}{P(C_2)}. \tag{6.133}$$

Thus the network output is given by a logistic sigmoid activation function acting on a weighted linear combination of the outputs of those hidden units which send connections to the output unit.

It is clear that we can apply the above arguments to the activations of hidden units in a network. Provided such units use logistic sigmoid activation functions, we can interpret their outputs as probabilities of the presence of corresponding 'features' conditioned on the inputs to the units.

6.7.2 *Properties of the cross-entropy error*

Suppose we write the network output, for a particular pattern n, in the form $y^n = t^n + \epsilon^n$. Then the cross-entropy error function (6.127) can be written as

$$E = -\sum_n \{t^n \ln(1 + \epsilon^n/t^n) + (1 - t^n)\ln(1 - \epsilon^n/(1 - t^n))\} \tag{6.134}$$

so that the error function depends on the *relative* errors of the network outputs. This should be compared with the sum-of-squares error function which depends on the (squares of the) absolute errors. Minimization of the cross-entropy error function will therefore tend to result in similar relative errors on both small and large target values. By contrast, the sum-of-squares error function tends to give similar absolute errors for each pattern, and will therefore give large relative errors for small output values. This suggests that the cross-entropy error function is likely to perform better than sum-of-squares at estimating small probabilities.

For binary targets, with $t^n = 1$ for an input vector $\mathbf{x}^n$ from class C_1 and $t^n = 0$ for inputs from class C_2, we can write the cross-entropy error function (6.134) in the form

$$E = -\sum_{n \in C_1} \ln(1 + \epsilon^n) - \sum_{n \in C_2} \ln(1 - \epsilon^n) \tag{6.135}$$

where we have used $z \ln z \to 0$ for $z \to 0$. If we suppose that ϵ^n is small, then the error function becomes

$$E \simeq \sum_n |\epsilon^n| \tag{6.136}$$

where we have expanded the logarithms using $\ln(1 + z) \simeq z$ and noted that if $y \in (0, 1)$ then $\epsilon^n < 0$ for inputs from class C_1 and $\epsilon^n > 0$ for inputs from class C_2. The result (6.136) has the form of the Minkowski-R error function for $R = 1$, discussed earlier. Compared to the sum-of-squares error function, this gives much stronger weight to smaller errors.

We have obtained the cross-entropy function by requiring that the network output y represents the probability of an input vector $\mathbf{x}$ belonging to class C_1. We can now confirm the consistency of this requirement by considering the minimum of the error function for an infinitely large data set, for which we can write (6.120) in the form

$$E = -\int\int \{t \ln y(\mathbf{x}) + (1 - t)\ln(1 - y(\mathbf{x}))\}\, p(t|\mathbf{x})p(\mathbf{x})\, dt\, d\mathbf{x}. \tag{6.137}$$

Since the network function $y(\mathbf{x})$ is independent of the target value t we can write (6.137) in the form

$$E = -\int \{\langle t|\mathbf{x}\rangle \ln y(\mathbf{x}) + (1 - \langle t|\mathbf{x}\rangle) \ln(1 - y(\mathbf{x}))\}\, p(\mathbf{x})\, d\mathbf{x} \tag{6.138}$$

where, as before, we have defined the conditional average of the target data as

$$\langle t|\mathbf{x}\rangle = \int t p(t|\mathbf{x})\, dt. \tag{6.139}$$

If we now set the functional derivative (Appendix D) of (6.138) with respect to $y(\mathbf{x})$ to zero we see that the minimum of the error function occurs when

$$y(\mathbf{x}) = \langle t|\mathbf{x}\rangle \tag{6.140}$$

so that, as for the sum-of-squares error, the output of the network approximates the conditional average of the target data for the given input vector. For the target coding scheme which we have adopted we have

$$p(t|\mathbf{x}) = \delta(t-1)P(\mathcal{C}_1|\mathbf{x}) + \delta(t)P(\mathcal{C}_2|\mathbf{x}). \tag{6.141}$$

Substituting (6.141) into (6.139) we find

$$y(\mathbf{x}) = P(\mathcal{C}_1|\mathbf{x}) \tag{6.142}$$

as required.

6.8 Multiple independent attributes

In all of the classification problems which we have considered so far, the aim has been to assign new vectors to one of c mutually exclusive classes. However, in some applications we may wish to use a network to determine the probabilities of the presence or absence of a number of attributes which need not be mutually exclusive. In this case the network has multiple outputs, and the value of the output variable y_k represents the probability that the kth attribute is present. If we treat the attributes as independent, then the distribution of target values will satisfy

$$p(\mathbf{t}|\mathbf{x}) = \prod_{k=1}^{c} p(t_k|\mathbf{x}). \tag{6.143}$$

We can now use (6.118) for each of the conditional distributions to give

$$p(\mathbf{t}|\mathbf{x}) = \prod_{k=1}^{c} y_k^{t_k}(1 - y_k)^{t_k}. \tag{6.144}$$

If we now construct the likelihood function and take the negative logarithm in the usual way, we obtain the error function in the form

$$E = -\sum_n \sum_{k=1}^{c} \{t_k^n \ln y_k^n + (1 - t_k^n) \ln(1 - y_k^n)\}. \tag{6.145}$$

With this choice of error function, the network outputs should each have a logistic sigmoidal activation function of the form (6.123). Again, for binary target variables t_k^n, this error function vanishes at its minimum. If the t_k^n are probabilities in the range $(0, 1)$, the minimum of the error will depend on the particular data set, and so it is convenient to subtract off this minimum value to give

$$E = -\sum_n \sum_{k=1}^{c} \left\{ t_k^n \ln \left(\frac{y_k^n}{t_k^n} \right) + (1 - t_k^n) \ln \left(\frac{1 - y_k^n}{1 - t_k^n} \right) \right\} \tag{6.146}$$

which always has an absolute minimum value with respect to the $\{y_k^n\}$ of zero.

6.9 Cross-entropy for multiple classes

We now return to the conventional classification problem involving mutually exclusive classes, and consider the form which the error function should take when the number of classes is greater than two. Consider a network with one output y_k for each class, and target data which has a 1-of-c coding scheme, so that $t_k^n = \delta_{kl}$ for a pattern n from class $\mathcal{C}_l$. The probability of observing the set of target values $t_k^n = \delta_{kl}$, given an input vector $\mathbf{x}^n$, is just $p(\mathcal{C}_l|\mathbf{x}) = y_l$. The value of the conditional distribution for this pattern can therefore be written as

$$p(\mathbf{t}^n|\mathbf{x}^n) = \prod_{k=1}^{c} (y_k^n)^{t_k^n}. \tag{6.147}$$

If we form the likelihood function, and take the negative logarithm as before, we obtain an error function of the form

$$E = -\sum_n \sum_{k=1}^{c} t_k^n \ln y_k^n. \tag{6.148}$$

The absolute minimum of this error function with respect to the $\{y_k^n\}$ occurs when $y_k^n = t_k^n$ for all values of k and n. At the minimum the error function takes the value

$$E_{\min} = -\sum_n \sum_{k=1}^{c} t_k^n \ln t_k^n. \tag{6.149}$$

For a 1-of-c coding scheme this minimum value is 0. However, the error function (6.148) is still valid, as we shall see, when t_k^n is a continuous variable in the range $(0, 1)$ representing the probability that input $\mathbf{x}^n$ belongs to class C_k. In this case the minimum of the error function need not vanish (it represents the entropy of the distribution of target variables, as will be discussed shortly). It is then convenient to subtract off this minimum value, and hence obtain the error function in the form

$$E = -\sum_n \sum_{k=1}^{c} t_k^n \ln\left(\frac{y_k^n}{t_k^n}\right) \tag{6.150}$$

which is non-negative, and which equals zero when $y_k^n = t_k^n$ for all k and n.

We now consider the corresponding activation function which should be used for the network output units. If the output values are to be interpreted as probabilities they must lie in the range $(0, 1)$, and they must sum to unity. This can be achieved by using a generalization of the logistic sigmoid activation function which takes the form

$$y_k = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})} \tag{6.151}$$

which is known as the normalized exponential, or *softmax* activation function (Bridle, 1990). The term softmax is used because this activation function represents a smooth version of the *winner-takes-all* activation model in which the unit with the largest input has output $+1$ while all other units have output 0. If the exponentials in (6.151) are modified to have the form $\exp(\beta a_k)$, then the winner-takes-all activation is recovered in the limit $\beta \to \infty$. The softmax activation function can be regarded as a generalization of the logistic function, since it can be written in the form

$$y_k = \frac{1}{1 + \exp(-A_k)} \tag{6.152}$$

where A_k is given by

$$A_k = a_k - \ln\left\{\sum_{k' \neq k} \exp(a_{k'})\right\}. \tag{6.153}$$

As with the logistic sigmoid, we can give a very general motivation for the softmax activation function by considering the posterior probability that a hidden unit activation vector $\mathbf{z}$ belongs to class C_k, in which the class-conditional densities are assumed to belong to the family of exponential distributions of the general form

$$p(\mathbf{z}|\mathcal{C}_k) = \exp\left\{A(\boldsymbol{\theta}_k) + B(\mathbf{z}, \boldsymbol{\phi}) + \boldsymbol{\theta}_k^{\mathrm{T}}\mathbf{z}\right\}. \tag{6.154}$$

From Bayes' theorem, the posterior probability of class $\mathcal{C}_k$ is given by

$$p(\mathcal{C}_k|\mathbf{z}) = \frac{p(\mathbf{z}|\mathcal{C}_k)P(\mathcal{C}_k)}{\sum_{k'} p(\mathbf{z}|\mathcal{C}_{k'})P(\mathcal{C}_{k'})}. \tag{6.155}$$

Substituting (6.154) into (6.155) and re-arranging we obtain

$$p(\mathcal{C}_k|\mathbf{z}) = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})} \tag{6.156}$$

where

$$a_k = \mathbf{w}_k^{\mathrm{T}}\mathbf{z} + w_{k0} \tag{6.157}$$

and we have defined

$$\mathbf{w}_k = \boldsymbol{\theta}_k \tag{6.158}$$

$$w_{k0} = A(\boldsymbol{\theta}_k) + \ln P(\mathcal{C}_k). \tag{6.159}$$

The result (6.156) represents the final layer of a network with softmax activation functions, and shows that (provided the distribution (6.154) is appropriate) the outputs can be interpreted as probabilities of class membership, conditioned on the outputs of the hidden units.

In evaluating the derivatives of the softmax error function we need to consider the inputs to all output units, and so we have (for pattern n)

$$\frac{\partial E^n}{\partial a_k} = \sum_{k'} \frac{\partial E^n}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial a_k}. \tag{6.160}$$

From (6.151) we have

$$\frac{\partial y_{k'}}{\partial a_k} = y_{k'}\delta_{kk'} - y_{k'}y_k \tag{6.161}$$

while from (6.150) we have

$$\frac{\partial E^n}{\partial y_{k'}} = -\frac{t_{k'}}{y_{k'}}. \tag{6.162}$$

Substituting (6.161) and (6.162) into (6.160) we find

$$\frac{\partial E^n}{\partial a_k} = y_k - t_k \tag{6.163}$$

which is the same result as found for both the sum-of-squares error (with a linear activation function) and the two-class cross-entropy error (with a logistic activation function). Again, we see that there is a natural pairing of error function and activation function.

6.10 Entropy

The concept of entropy was originally developed by physicists in the context of equilibrium thermodynamics and later extended through the development of statistical mechanics. It was introduced into information theory by Shannon (1948). An understanding of basic information theory leads to further insights into the entropy-based error measures discussed in this section. It also paves the way for an introduction to the minimum description length framework in Section 10.10. Here we consider two distinct but related interpretations of entropy, the first based on *degree of disorder* and the second based on *information content.*

Consider a probability density function $p(x)$ for a single random variable x. It is convenient to represent the density function as a histogram in which the x-axis has been divided into bins labelled by the integer i. Imagine constructing the histogram by putting a total of N identical discrete objects into the bins, such that the ith bin contains N_i objects. We wish to count the number of distinct ways in which objects can be arranged, while still giving rise to the same histogram. Since there are N ways of choosing the first object, $(N-1)$ ways of choosing the second object, and so on, there a total of $N!$ ways to select the N objects. However, we do not wish to count rearrangements of objects within a single bin. For the ith bin there are $N_i!$ such rearrangements and so the total number of distinct ways to arrange the objects, known as the multiplicity, is given by

$$W = \frac{N!}{\prod_i N_i!}. \tag{6.164}$$

The entropy is defined as (a constant times) the negative logarithm of the multiplicity

$$S = -\frac{1}{N} \ln W = -\frac{1}{N} \{\ln N! - \sum_i \ln N_i!\}. \tag{6.165}$$

We now consider the limit $N \to \infty$, and make use of Stirling's approximation $\ln N! \simeq N \ln N - N$ together with the relation $\sum_i N_i = N$, to give

$$S = -\lim_{N \to \infty} \sum_i \left(\frac{N_i}{N}\right) \ln \left(\frac{N_i}{N}\right) = -\sum_i p_i \ln p_i \tag{6.166}$$

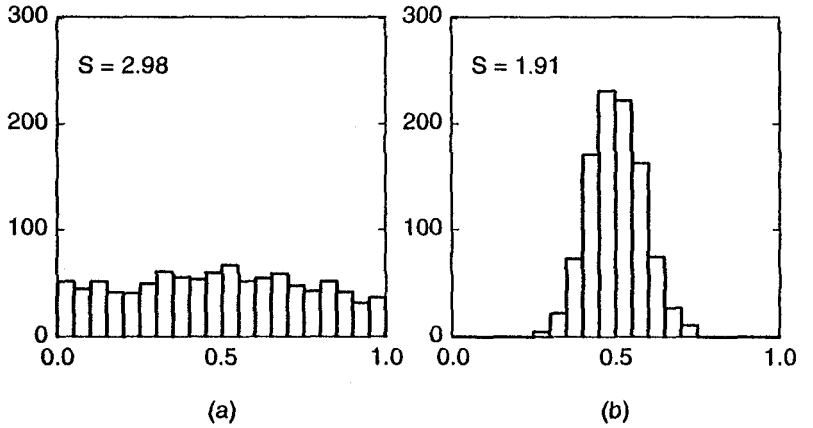

Figure 6.13. Examples of two histograms, together with their entropy values defined by (6.166). The histograms were generated by sampling two Gaussian functions with variance parameters $\sigma = 0.4$ and $\sigma = 0.08$, and each contain 1000 points. Note that the more compact distribution has a lower entropy.

where $p_i = N_i/N$ (as $N \to \infty$) represents the probability corresponding to the ith bin. The entropy therefore gives a measure of the number of different *microstates* (arrangements of objects in the bins) which can give rise to a given *macrostate* (i.e. a given set of probabilities p_i). A very sharply peaked distribution has a very low entropy, whereas if the objects are spread out over many bins the entropy is much higher. The smallest value for the entropy is 0 and occurs when all of the probability mass is concentrated in one bin (so that one of the p_i is 1 and all the rest are 0). Conversely the largest entropy arises when all of the bins contain equal probability mass, so that $p_i = 1/M$ where M is the total number of bins. This is easily seen by maximizing (6.166) subject to the constraint $\sum_i p_i = 1$ using a Lagrange multiplier (Appendix C). An example of two histograms, with their respective entropies, is shown in Figure 6.13.

For continuous distributions (rather than histograms) we can take the limit in which the number M of bins goes to infinity. If Δ is the width of each bin, then the probability mass in the ith bin is $p_i = p(x_i)\Delta$, and so the entropy can be written in the form

$$S = \lim_{M\to\infty} \sum_{i=1}^{M} p(x_i)\Delta \ln \{p(x_i)\Delta\} \tag{6.167}$$

$$= \int p(x) \ln p(x)\, dx + \lim_{M\to\infty} \ln \Delta \tag{6.168}$$

where we have used $\int p(x)\,dx = 1$. The second term on the right-hand side diverges in the limit $M \to \infty$. In order to define a meaningful entropy measure for continuous distributions we discard this term, since it is independent of $p(x)$, and simply use the first term on the right-hand side of (6.168), which is called the *differential entropy*. This is reasonable, since if we measure the difference in entropy between two distributions, the second term in (6.168) would cancel. For distributions which are functions of several variables, we define the entropy to be

$$S = -\int p(\mathbf{x}) \ln p(\mathbf{x})\,d\mathbf{x} \tag{6.169}$$

where $\mathbf{x} \equiv (x_1, \ldots, x_d)^{\mathrm{T}}$.

It is interesting to consider the form of distribution which gives rise to the maximum of the entropy function. In order to find a meaningful maximum it is necessary to constrain the variance of the distribution. For the case of a single variable x on the infinite axis $(-\infty, \infty)$, we maximize

$$S = -\int_{-\infty}^{\infty} p(x) \ln p(x)\,dx \tag{6.170}$$

subject to the constraints that the distribution be normalized and that the mean and variance of the distribution have specified values

$$\int_{-\infty}^{\infty} p(x)\,dx = 1 \tag{6.171}$$

$$\int_{-\infty}^{\infty} xp(x)\,dx = \mu \tag{6.172}$$

$$\int_{-\infty}^{\infty} (x-\mu)^2 p(x)\,dx = \sigma^2. \tag{6.173}$$

Introducing Lagrange multipliers λ_1, λ_2 and λ_3 (Appendix C) for each of the constraints, we can use calculus of variations (Appendix D) to maximize the functional

$$\int_{-\infty}^{\infty} p(x) \left\{ \ln p(x) + \lambda_1 + \lambda_2 x + \lambda_3 (x-\mu)^2 \right\} dx - \lambda_1 - \lambda_2 \mu - \lambda_3 \sigma^2 \tag{6.174}$$

which leads to

$$p(x) = \exp\left\{ -1 - \lambda_1 - \lambda_2 x - \lambda_3 (x-\mu)^2 \right\}. \tag{6.175}$$

We can solve for the Lagrange multipliers by back-substituting this expression into the constraint equations. This finally gives the expression for the maximizing distribution in the form

$$p(x) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}. \tag{6.176}$$

Thus we see that the distribution having maximum entropy, for given mean and variance, is the Gaussian.

As a second viewpoint on the interpretation of entropy, let us consider the amount of information, or equivalently the 'degree of surprise', which is obtained when we learn that a particular event has occurred. We expect that the information will depend on the probability p of the event, since if $p = 1$ then the event is certain to occur, and there is no surprise when the event is found to occur (and so no information is received). Conversely, if the probability is low, then there is a large degree of surprise in learning that it has occurred. We are therefore looking for a measure of information $s(p)$ which is a continuous, monotonically increasing function of p and which is such that $s(1) = 0$. An appropriate expression can be obtained as follows. Consider two independent events A and B, with probabilities p_A and p_B. If we know that both events have occurred then the total information is $s(p_A p_B)$. If, however, we are first told that A has occurred, then the residual information on learning that B has occurred must be $s(p_A p_B) - s(p_A)$, which must equal $s(p_B)$ since knowledge that A has occurred should not affect the information resulting from learning that B occurred (since the events are independent). This leads to the following condition

$$s(p_A p_B) = s(p_A) + s(p_B). \tag{6.177}$$

From this we can deduce that $s(p^2) = 2s(p)$ and by induction that $s(p^N) = Ns(p)$ for integer N. Similarly, $s(p) = s([p^{1/N}]^N) = Ns(p^{1/N})$ and by extension $s(p^{M/N}) = (M/N)s(p)$. This implies that

$$s(p^x) = xs(p) \tag{6.178}$$

for rational x and hence, by continuity, for real x. If we define $z = -\log_2 p$, so that $p = (1/2)^z$, then

$$s(p) = s((1/2)^z) = zs(1/2) = -s(1/2)\log_2(p). \tag{6.179}$$

It is conventional to choose $s(1/2) = 1$. The information is then expressed in *bits* (binary digits). From now on we shall consider logarithms to base e (natural logarithms) in which case the information is expressed in *nats*. We see that the amount of information is proportional to the logarithm of the probability. This arises essentially because, for independent events, probabilities are multiplicative,

while information is additive.

Consider a random variable α which can take values α_k with probabilities $p(\alpha_k)$. If a sender wishes to transmit the value of α to a receiver, then the amount of information (in bits) which this requires is $-\ln p(\alpha_k)$ if the variable takes the value α_k. Thus, the expected (average) information needed to transmit the value of α is given by

$$S(\alpha) = -\sum_k p(\alpha_k) \ln p(\alpha_k) \tag{6.180}$$

which is the entropy of the random variable α. Thus $S(\alpha)$ as the average amount of information received when the value of α is observed. The average length of a binary message (in nats) needed to transmit the value of α is at least equal to the entropy of α. This is known as the *noiseless coding theorem* (Shannon, 1948; Viterbi and Omura, 1979).

Returning to the case of continuous variables, denoted by the vector $\mathbf{x}$, we note that in practice we do not know the true distribution $p(\mathbf{x})$. If we encode the value of $\mathbf{x}$ for transmission to a receiver, then we must (implicitly or explicitly) choose a distribution $q(\mathbf{x})$ from which to construct the coding. The information needed to encode a value of $\mathbf{x}$ under this distribution is just $-\ln q(\mathbf{x})$. If the variable $\mathbf{x}$ is drawn from a true distribution $p(\mathbf{x})$ then the average information needed to encode $\mathbf{x}$ is given by

$$-\int p(\mathbf{x}) \ln q(\mathbf{x}) \, d\mathbf{x} \tag{6.181}$$

which is the *cross-entropy* between the distributions $q(\mathbf{x})$ and $p(\mathbf{x})$. Comparison with (2.68) shows that this equals the negative log likelihood under the model distribution $q(\mathbf{x})$ when the true distribution is $p(\mathbf{x})$. It is also equal to the sum of the Kullback–Leibler distance between $p(\mathbf{x})$ and $q(\mathbf{x})$, given by (2.70), and the entropy of $p(\mathbf{x})$ since

$$-\int p(\mathbf{x}) \ln q(\mathbf{x}) \, d\mathbf{x} = -\int p(\mathbf{x}) \ln \frac{q(\mathbf{x})}{p(\mathbf{x})} \, d\mathbf{x} - \int p(\mathbf{x}) \ln p(\mathbf{x}) \, d\mathbf{x}. \tag{6.182}$$

We can easily show that, of all possible distributions $q(\mathbf{x})$, the choice which gives the smallest average information, i.e. the smallest value for the cross-entropy, is the true distribution $p(\mathbf{x})$ (Exercise 6.21). Since the entropy of $p(\mathbf{x})$ is independent of the distribution $q(\mathbf{x})$, we see from (6.182) that minimization of the cross-entropy is equivalent to minimization of the Kullback–Leibler distance.

We can apply the concept of cross-entropy to the training of neural networks. For a variable α which takes a discrete set of values α_k we can write (6.181) in the form

$$-\sum_k P(\alpha_k) \ln Q(\alpha_k). \tag{6.183}$$

Consider first a network with c outputs $y_k(\mathbf{x})$ representing the model probabilities for $\mathbf{x}$ to belong to the corresponding classes $\mathcal{C}_k$. We shall suppose that we also have a set of target variables t_k representing the corresponding true probabilities. Then the cross-entropy becomes

$$-\sum_{k=1}^{c} t_k \ln y_k(\mathbf{x}). \tag{6.184}$$

For a set of N data points which are assumed to be drawn independently from a common distribution, the information is additive and hence the total cross-entropy is given by

$$-\sum_{n=1}^{N}\sum_{k=1}^{c} t_k^n \ln y_k(\mathbf{x}^n) \tag{6.185}$$

which can be used as an error function for network training. We see that this form of error function is valid not only when the targets t_k^n have a one-of-c coding (representing precise knowledge of the true classes of the data) but also when they lie anywhere in the range $0 \leq t_k^n \leq 1$, subject to the constraint $\sum_k t_k^n = 1$, corresponding to probabilities of class membership.

For two classes, we can consider a network with a single output y representing the model probability for membership of class $\mathcal{C}_1$, with corresponding true probability t. The model probability for membership of class $\mathcal{C}_2$ is then $1-y$, and the corresponding true probability is $1-t$. Following the same line of argument as above we then arrive at the cross-entropy error function for two classes and N data points in the form

$$-\sum_{n=1}^{N} \left\{ t^n \ln y(\mathbf{x}^n) + (1-t^n) \ln(1-y(\mathbf{x}^n)) \right\}. \tag{6.186}$$

6.11 General conditions for outputs to be probabilities

So far, we have considered three different error measures (sum-of-squares, cross-entropy for a single output, and cross-entropy for softmax networks) all of which allow the network outputs to be interpreted as probabilities. We may therefore wonder what conditions an error measure should satisfy in order that the network outputs have this property. The discussion given here is based on that of Hampshire and Pearlmutter (1990).

All of the error measures we are considering take the form of a sum over patterns of an error term for each pattern $E = \sum_n E^n$. We shall also take the error to be a sum over terms for each output unit separately. This corresponds to the assumption that the distributions of different target variables are statistically independent (which is not satisfied by the Gaussian mixture based error considered earlier, or by the softmax error, for instance). Thus we write

$$E^n = \sum_{k=1}^{c} f(y_k^n, t_k^n) \tag{6.187}$$

where $f(\cdot, \cdot)$ is some function to be determined. We shall also assume that f depends only on the magnitude of the difference between y_k and t_k, so that $f(y_k^n, t_k^n) = f(|y_k^n - t_k^n|)$. In the limit of an infinite data set, we can write the average (or expected) per-pattern error in the form

$$\langle E \rangle = \sum_{k=1}^{c} \int\int f(|y_k - t_k|) p(\mathbf{t}|\mathbf{x}) p(\mathbf{x}) \, d\mathbf{t} \, d\mathbf{x}. \tag{6.188}$$

If we use a 1-of-c target coding scheme, then from (6.99) we can write the conditional distribution of the target variables in the form

$$p(\mathbf{t}|\mathbf{x}) = \prod_{k=1}^{c} \left\{ \sum_{l=1}^{c} \delta(t_k - \delta_{kl}) P(\mathcal{C}_l|\mathbf{x}) \right\}. \tag{6.189}$$

We now substitute (6.189) into (6.188) and evaluate the integrals over the t_k variables (which simply involves integrals of δ-functions) to give

$$\langle E \rangle = \sum_{k=1}^{c} \int \{ f(1 - y_k) P(\mathcal{C}_k|\mathbf{x}) + f(y_k) [1 - P(\mathcal{C}_k|\mathbf{x})] \} \, p(\mathbf{x}) \, d\mathbf{x} \tag{6.190}$$

where we have used $\sum_k P(\mathcal{C}_k|\mathbf{x}) = 1$, and assumed that $0 \leq y_k \leq 1$ so that the modulus signs can be omitted. The condition that the average per-pattern error in (6.190) be minimized with respect to the $y_k(\mathbf{x})$ is given by setting the functional derivative of $\langle E \rangle$ (Appendix D) to zero

$$\frac{\delta \langle E \rangle}{\delta y_k(\mathbf{x})} = -f'(1 - y_k) P(\mathcal{C}_k|\mathbf{x}) + f'(y_k) [1 - P(\mathcal{C}_k|\mathbf{x})] = 0 \tag{6.191}$$

which gives

$$\frac{f'(1 - y_k)}{f'(y_k)} = \frac{1 - P(\mathcal{C}_k|\mathbf{x})}{P(\mathcal{C}_k|\mathbf{x})}. \tag{6.192}$$

If the outputs of the network are to represent probabilities, so that $y_k(\mathbf{x}) = P(C_k|\mathbf{x})$, then the function f must satisfy the condition

$$\frac{f'(1-y)}{f'(y)} = \frac{1-y}{y}. \tag{6.193}$$

A class of functions f which satisfies this condition is given by

$$f(y) = \int y^r(1-y)^{r-1}\,dy. \tag{6.194}$$

This includes two important error functions which we have encountered already. For $r = 1$ we obtain $f(y) = y^2/2$ which gives the sum-of-squares error function. Similarly, for $r = 0$ we obtain $f(y) = -\ln(1-y) = -\ln(1-|y|)$ which gives rise to the cross-entropy error function. To see this, consider a single output and note that $f(y,t) = -\ln(1-|y-t|) = -\ln(y)$ if $t = 1$ and $f(y,t) = -\ln(1-|y-t|) = -\ln(1-y)$ if $t = 0$. These can be combined into a single expression of the form

$$-\{t\ln y + (1-t)\ln(1-y)\}. \tag{6.195}$$

Summing over all outputs, as in (6.187), and then over all patterns gives the cross-entropy error for multiple independent attributes in the form (6.145).

As an example of an error function which does not satisfy (6.193), consider the Minkowski-R error measure which is given by $f(y) = y^R$. Substituting this into (6.193) gives

$$y^{R-2} = (1-y)^{R-2} \tag{6.196}$$

which is only satisfied if $R = 2$, corresponding to the sum-of-squares error. For $R \neq 2$, the outputs of the network do not correspond to posterior probabilities. They do, however, represent non-linear discriminant functions, so that the minimum probability of mis-classification is obtained by assigning patterns to the class for which the corresponding network output is largest. To see this, substitute $f(y) = y^R$ into the condition (6.192) satisfied by the network outputs at the minimum of the error function, to give

$$y_k(\mathbf{x}) = \frac{P(C_k|\mathbf{x})^{1/(R-1)}}{P(C_k|\mathbf{x})^{1/(R-1)} + [1-P(C_k|\mathbf{x})]^{1/(R-1)}}. \tag{6.197}$$

We see that the y_k only represent the posterior probabilities when $R = 2$, corresponding to the sum-of-squares error. However, the decision boundaries correspond to the minimum mis-classification rate discriminant for all values of R since the y_k are monotonic functions of the posterior probabilities $P(C_k|\mathbf{x})$.

Exercises

6.1 ($\star$) Throughout this chapter we have considered data in which the input vectors $\mathbf{x}$ are known exactly, but the target vectors $\mathbf{t}$ are noisy. Consider instead the situation in which the target data is generated from a smooth function $\mathbf{h}(\mathbf{x})$ but where the input data is corrupted by additive noise (Webb, 1994). Show that the sum-of-squares error, in the infinite data limit, can be written as

$$E = \frac{1}{2} \int\int \|\mathbf{y}(\mathbf{x} + \boldsymbol{\xi}) - \mathbf{h}(\mathbf{x})\|^2 p(\mathbf{x}, \boldsymbol{\xi})\, d\mathbf{x}\, d\boldsymbol{\xi}. \tag{6.198}$$

By changing variables to $\mathbf{z} = \mathbf{x} + \boldsymbol{\xi}$, and using functional differentiation (Appendix D), show that the least squares solution is given by

$$\mathbf{y}(\mathbf{z}) = \int \mathbf{h}(\mathbf{z} - \boldsymbol{\xi}) p(\boldsymbol{\xi}|\mathbf{z})\, d\boldsymbol{\xi} \tag{6.199}$$

so that the optimum solution is again given by the conditional expectation of the target data.

6.2 ($\star$) Consider a model in which the target data is taken to have the form

$$\mathbf{t}^n = \mathbf{y}(\mathbf{x}^n; \mathbf{w}) + \boldsymbol{\epsilon}^n \tag{6.200}$$

where $\boldsymbol{\epsilon}^n$ is drawn from a zero mean Gaussian distribution having a fixed covariance matrix $\boldsymbol{\Sigma}$. Derive the likelihood function for a data set drawn from this distribution, and hence write down the error function. The use of such an error function is called *generalized least squares*, and the usual sum-of-squares error function corresponds to the special case $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$ where $\mathbf{I}$ is the identity matrix.

6.3 ($\star$) Consider a network with linear output units whose final-layer weights are obtained by minimization of a sum-of-squares error function using the pseudo-inverse matrix. Show that, if the target values for each training pattern satisfy several linear constraints of the form (6.31) simultaneously, then the outputs of the trained network will satisfy the same constraints exactly for an arbitrary input vector.

6.4 ($\star$) Verify the normalization of the probability density function in (6.58). Use the result $\Gamma(1/2) = \sqrt{\pi}$ to show that the Gaussian distribution is a special case corresponding to $R = 2$.

6.5 ($\star$) Write down an expression for the Minkowski-R error function (6.59) with $R = 1$ in infinite data limit, and hence show that the network mapping which minimizes the error is given by the conditional *median* of the target data.

6.6 ($\star\star$) Write down an expression for the conditional mixture density error function (6.77) in the limit of an infinite data set. Hence, by using functional differentiation (Appendix D), find expressions satisfied by the quantities

$\alpha_j(\mathbf{x})$, $\boldsymbol{\mu}_j(\mathbf{x})$ and $\sigma_j^2(\mathbf{x})$, in terms of conditional averages, at the minimum of this error. Note that the constraint $\sum_j \alpha_j = 1$ should be enforced by using a Lagrange multiplier (Appendix C). Discuss the interpretation of these expressions.

6.7 ($\star$) Consider the circular normal distribution given by (6.95) and show that, for $\theta - \theta_0 \ll 1$, the shape of the distribution is approximately Gaussian.

6.8 ($\star\star$) In Section 6.4.1 we discussed a technique for modelling the conditional density $p(\theta|\mathbf{x})$ of a periodic variable θ based on a mixture of circular normal distributions. Here we investigate an alternative approach which involves finding a transformation from the periodic variable $\theta \in (0, 2\pi)$ to a Euclidean variable $\chi \in (-\infty, \infty)$, and then applying the Gaussian mixture technique of Section 6.4 to the estimation of the conditional density $\widetilde{p}(\theta|\mathbf{x})$ in χ-space (Bishop and Legleye, 1995). Consider the density function defined by the transformation

$$p(\theta|\mathbf{x}) = \sum_{L=-\infty}^{\infty} \widetilde{p}(\theta + L2\pi|\mathbf{x}) \tag{6.201}$$

where $\widetilde{p}(\chi|\mathbf{x})$ is a density function in χ-space. Show that (6.201) satisfies the periodicity requirement $p(\theta + 2\pi|\mathbf{x}) = p(\theta|\mathbf{x})$. Also, show that, if the density function $\widetilde{p}(\chi|\mathbf{x})$ is normalized on the interval $(-\infty, \infty)$, then the density $p(\theta|\mathbf{x})$ will be normalized on $(0, 2\pi)$. The density function $\widetilde{p}(\chi|\mathbf{x})$ can now be modelled using a mixture of Gaussians $\phi_j(\chi|\mathbf{x})$ of the form

$$\widetilde{p}(\chi|\mathbf{x}) = \sum_{j=1}^{M} \alpha_j(\mathbf{x})\phi_j(\chi|\mathbf{x}). \tag{6.202}$$

Write down the error function given by the negative logarithm of the likelihood of a set of data points $\{\mathbf{x}^n, \theta^n\}$, and find expressions for the derivatives of the error function with respect to the means and variances of the Gaussian components. Assuming that the mixing coefficients α_j are determined by a softmax function of the form (6.74), find the derivatives of the error function with respect to the corresponding network output variables z_j^α. Note that, in a practical implementation, it is necessary to restrict the summation over L to a limited range. Since the Gaussian functions $\phi_j(\chi|\mathbf{x})$ have exponentially decaying tails, this can represent an extremely good approximation in almost all cases.

6.9 ($\star$) Using the definition of the pseudo-inverse matrix given by (6.30), verify that the result (6.105) follows from the pseudo-inverse formula (6.104).

6.10 ($\star$) Verify that, for a 1-of-c target coding scheme, the between-class covariance matrix given by (6.107) reduces to the form (6.109).

6.11 ($\star$) The result (6.108) shows that minimizing a sum-of-squares error function for a network with linear output units, maximizes a particular nonlinear discriminant function defined over the space of activations of the

hidden units. Show that if, instead of using 0 and 1 as the network targets, the values 0 and $1/\sqrt{N_k}$ are used, where N_k is the number of patterns in class C_k, then the between-class covariance matrix, given by (6.107) becomes

$$\mathbf{S}_B = \sum_k N_k(\overline{\mathbf{z}}^k - \overline{\mathbf{z}})(\overline{\mathbf{z}}^k - \overline{\mathbf{z}})^{\mathrm{T}} \tag{6.203}$$

where $\overline{\mathbf{z}}^k$ is defined by (6.110). This is now the standard between-class covariance matrix as introduced in Section 3.6.

6.12 ($\star\star$) Consider a weighted sum-of-squares error function of the form (6.112) in which the network outputs y_k are given by (6.21). Show that the solution for the biases which minimizes the error function is given by

$$w_{k0} = \overline{t}_k - \sum_{j=1}^{M} w_{kj}\overline{z}_j \tag{6.204}$$

where we have introduced the following weighted averages

$$\overline{t}_k = \frac{\sum_{n=1}^{N} \kappa_n t_k^n}{\sum_{n=1}^{N} \kappa_n}, \qquad \overline{z}_j = \frac{\sum_{n=1}^{N} \kappa_n z_j^n}{\sum_{n=1}^{N} \kappa_n}. \tag{6.205}$$

Use this result to show that the error function, with the biases set to their optimal values, can be written in the form

$$E = \frac{1}{2}\mathrm{Tr}\{(\mathbf{Z}\mathbf{W}^{\mathrm{T}} - \mathbf{T})^{\mathrm{T}}\mathbf{K}^{\mathrm{T}}\mathbf{K}(\mathbf{Z}\mathbf{W}^{\mathrm{T}} - \mathbf{T})\} \tag{6.206}$$

where $\mathbf{K} = \mathrm{diag}(\kappa_n^{1/2})$, $(\mathbf{T})_{nk} = \widetilde{t}_k^n$, $(\mathbf{W})_{kj} = w_{kj}$ and $(\mathbf{Z})_{nj} = \widetilde{z}_j^n$, and we have defined

$$\widetilde{t}_k^n = t_k^n - \overline{t}_k, \qquad \widetilde{z}_j^n = z_j^n - \overline{z}_j. \tag{6.207}$$

Show that (6.206) has the same form as the error function in (6.103) but with $\mathbf{Z}$ and $\mathbf{T}$ pre-multiplied by $\mathbf{K}$. Hence show that the value of $\mathbf{W}$ which minimizes this error function is given by

$$\mathbf{W}^{\mathrm{T}} = (\mathbf{K}\mathbf{Z})^{\dagger}\mathbf{K}\mathbf{T} \tag{6.208}$$

Hence show that minimization of the error (6.206) is equivalent to maximization of a criterion of the form

$$J = \frac{1}{2}\mathrm{Tr}\{\mathbf{S}_B\mathbf{S}_T^{-1}\} \tag{6.209}$$

in which

$$\mathbf{S}_B = \mathbf{Z}^{\mathrm{T}}\mathbf{K}\mathbf{T}\mathbf{T}^{\mathrm{T}}\mathbf{K}\mathbf{Z} \tag{6.210}$$

$$\mathbf{S}_T = \mathbf{Z}^{\mathrm{T}}\mathbf{K}\mathbf{Z}. \tag{6.211}$$

Show that, for a 1-of-c target coding scheme, and for weighting factors κ_n given by (6.113), the total covariance matrix $\mathbf{S}_T$ is given by (6.114) and the between-class covariance matrix $\mathbf{S}_B$ is given by (6.116).

6.13 ($\star$) Suppose that, in Exercise 6.11, the target values had been set to $t_k^n = 1 - L_{lk}$ for a pattern n belonging class $\mathcal{C}_l$, where L_{lk} represents the loss associated with assigning such a pattern to class $\mathcal{C}_k$ (loss matrices are introduced in Section 1.10). Show that the between-class covariance matrix given by (6.107) takes the form (6.117). Verify that this reduces to the form (6.109) when $L_{lk} = 1 - \delta_{lk}$.

6.14 ($\star$) Consider the Hessian matrix for the cross-entropy error function (6.120) for two classes and a single network output. Show that, in the limit of an infinite data set, the terms involving second derivatives of the network outputs, as well as some of the terms involving first derivatives, vanish at the minimum of the error function as a consequence of the fact that the network outputs equal the conditional averages of the target data. Extend this result to the cross-entropy error (6.145) corresponding to several independent attributes.

6.15 ($\star$) Show that the entropy measure in (6.145), which was derived for targets $t_k = 0, 1$, applies also in the case where the targets are probabilities with values in the range $(0, 1)$. Do this by considering an extended data set in which each pattern t_k^n is replaced by a set of M patterns of which a fraction Mt_k^n are set to 1 and the remainder are set to 0, and then applying (6.145) to this extended data set.

6.16 ($\star$) Consider the error function (6.148), together with a network whose outputs are given by a softmax activation function (6.151), in the limit of an infinite data set. Show that the network output functions $y_k(\mathbf{x})$ which minimize the error are given by the conditional averages of the target data $\langle t_k|\mathbf{x}\rangle$. Hint: since the $\{y_k\}$ are not independent, as a result of the constraint $\sum_k y_k = 1$, consider the functional derivative (Appendix D) with respect to $a_k(\mathbf{x})$ instead.

6.17 ($\star$) Consider the Hessian matrix for the error function (6.148) and a network with a softmax output activation function (6.151) so that $\sum_k y_k(\mathbf{x}) = 1$. Show that the terms involving second derivatives of the network outputs vanish in the limit of infinite data, provided the network has been trained to a minimum of the error function. Hint: make use of the result of Exercise 6.16.

6.18 ($\star$) Consider a classification network in which the targets for training are given by $t_k^n = 1 - L_{lk}$ for an input vector $\mathbf{x}^n$ from class $\mathcal{C}_l$, where L_{lk} are the elements of a loss matrix, as discussed in Section 1.10. Use the general result $y_k(\mathbf{x}) = \langle t_k|\mathbf{x}\rangle$ for the network outputs at the minimum of the error function to show that the outputs are given by weighted posterior

probabilities such that selection of the largest output corresponds to the minimum-risk classification.

6.19 (⋆⋆) Generate histograms of the kind shown in Figure 6.13 for a discrete variable by sampling from a distribution consisting of a mixture of two Gaussians. Evaluate numerically the entropy of the histograms using (6.166) and explore the dependence of the entropy on the parameters of the mixture model.

6.20 (⋆) Using the technique of functional differentiation (Appendix D), together with Lagrange multipliers (Appendix C), show that the probability density function $p(x)$ which maximizes the entropy

$$S = \int p(x) \ln p(x)\, dx \tag{6.212}$$

subject to the constraints

$$\int p(x)\, dx = 1 \tag{6.213}$$

$$\int x p(x)\, dx = \mu \tag{6.214}$$

$$\int |x - \mu|^R p(x)\, dx = \sigma^R \tag{6.215}$$

is given by

$$p(x) = \frac{R^{1-1/R}}{2\sigma\Gamma(1/R)} \exp\left(-\frac{|x-\mu|^R}{R\sigma^R}\right) \tag{6.216}$$

where $\Gamma(a)$ is the gamma function defined on page 28.

6.21 (⋆) Show that the choice of distribution $q(\mathbf{x})$ which minimizes the cross-entropy (6.181) is given by $q(\mathbf{x}) = p(\mathbf{x})$. To do this, consider the functional derivative (Appendix D) of (6.181) with respect to $q(\mathbf{x})$. This derivative needs to be evaluated subject to the constraint

$$\int q(\mathbf{x})\, d\mathbf{x} = 1 \tag{6.217}$$

which can be imposed by using a Lagrange multiplier (Appendix C).

6.22 (⋆) By substituting (6.189) into (6.188) and evaluating the integral over $\mathbf{t}$, derive the result (6.190).