the conditional average of the target data. The quantity $(y_n - t_n)$ is then a random variable with zero mean. If we assume that its value is uncorrelated with the value of the second derivative term on the right-hand side of (8.39), then the whole term will average to zero in the summation over n.

Exercise 8.9

By neglecting the second term in (8.39), we arrive at the *Levenberg–Marquardt* approximation, also known as the *outer product* approximation because the Hessian matrix is built up from a sum of outer products of vectors, given by

$$\mathbf{H} \simeq \sum_{n=1}^{N} \nabla a_n \nabla a_n^{\mathrm{T}}. \tag{8.40}$$

Evaluating the outer product approximation for the Hessian is straightforward as it involves only first derivatives of the error function, which can be evaluated efficiently in $\mathcal{O}(W)$ steps using standard backpropagation. The elements of the matrix can then be found in $\mathcal{O}(W^2)$ steps by simple multiplication. It is important to emphasize that this approximation is likely to be valid only for a network that has been trained appropriately, and that for a general network mapping, the second derivative terms on the right-hand side of (8.39) will typically not be negligible.

For a cross-entropy error function for a network with logistic-sigmoid output-unit activation functions, the corresponding approximation is given by

Exercise 8.10

$$\mathbf{H} \simeq \sum_{n=1}^{N} y_n(1 - y_n) \nabla a_n \nabla a_n^{\mathrm{T}}. \tag{8.41}$$

An analogous result can be obtained for multi-class networks having softmax output-unit activation functions. The outer product approximation can also be used to develop an efficient sequential procedure for approximating the inverse of a Hessian (Hassibi and Stork, 1993).

Exercise 8.11
Exercise 8.12

8.2. Automatic Differentiation

We have seen the importance of using gradient information to train neural networks efficiently. There are essentially four ways in which the gradient of a neural network error function can be evaluated.

Section 7.2.1

The first approach, which formed the mainstay of neural networks for many years, is to derive the backpropagation equations by hand and then to implement them explicitly in software. If this is done carefully it results in efficient code that gives precise results that are accurate to numerical precision. However, the process of deriving the equations as well as the process of coding them both take time and are prone to errors. It also results in some redundancy in the code because the forward propagation equations are coded separately from the backpropagation equations. As these often involve duplicated calculations, then if the model is altered, both the forward and backward implementations need to be changed in unison. This effort