9

LEARNING AND GENERALIZATION

As we have emphasized in several other chapters, the goal of network training is not to learn an exact representation of the training data itself, but rather to build a statistical model of the process which generates the data. This is important if the network is to exhibit good generalization, that is, to make good predictions for new inputs. In Section 1.5, we introduced the simple analogy of curve fitting using polynomials, and showed that a polynomial with too few coefficients gives poor predictions for new data, i.e. poor generalization, since the polynomial function has too little flexibility. Conversely, a polynomial with too many coefficients also gives poor generalization since it fits too much of the noise on the training data. The number of coefficients in the polynomial controls the effective flexibility, or complexity, of the model.

This highlights the need to optimize the complexity of the model in order to achieve the best generalization. Considerable insight into this phenomenon can be obtained by introducing the concept of the bias–variance trade-off, in which the generalization error is decomposed into the sum of the *bias* squared plus the *variance*. A model which is too simple, or too inflexible, will have a large bias, while one which has too much flexibility in relation to the particular data set will have a large variance. Bias and variance are complementary quantities, and the best generalization is obtained when we have the best compromise between the conflicting requirements of small bias and small variance.

In order to find the optimum balance between bias and variance we need to have a way of controlling the effective complexity of the model. In the case of neural networks, the complexity can be varied by changing the number of adaptive parameters in the network. This is called *structural stabilization*. One way to implement this in practice is to compare a range of models having different different numbers of hidden units. Alternatively, we can start with a relatively large network and prune out the least significant connections, either by removing individual weights or by removing complete units. Similarly, we can start with a small network, and add units during the learning process, with the goal of arriving at an optimal network structure. Yet another way to reduce variance is to combine the outputs of several networks together to form a committee.

The second principal approach to controlling the complexity of a model is through the use of *regularization* which involves the addition of a penalty term to the error function. We can control the degree of regularization, and hence the effective complexity of the model, by scaling the regularization term by an adjustable multiplicative parameter.

In a practical application, we have to optimize the model complexity for the given training data set. One of the most important techniques for doing this is called *cross-validation.*

In Chapter 10 we discuss the Bayesian framework which provides a complimentary viewpoint to the one presented in this chapter. The bias–variance trade-off is then no longer relevant, and we can in principle consider networks of arbitrarily high complexity without encountering over-fitting.

9.1 Bias and variance

In Section 1.5 we discussed the problem of curve fitting using polynomial functions, and we showed that there is an optimal number of coefficients for the polynomial, for a given training set, in order to obtain the best represertation of the underlying systematic properties of the data, and hence to obtain the best generalization on new data. This represents a trade-off between achieving a good fit to the training data, and obtaining a reasonably smooth function which is not over-fitted to the data. Similar considerations apply to the problem of density estimation, discussed in Chapter 2, where various smoothing parameters arise which control the trade-off between smoothing the model density function and fitting the data set. The same issues also arise in the supervised training of neural networks.

A key insight into this trade-off comes from the decomposition of error into *bias* and *variance* components (Geman *et al.*, 1992). We begin with a mathematical treatment of the bias–variance decomposition, and then discuss its implications.

It is convenient to consider the particular case of a model trained using a sum-of-squares error function, although our conclusions will be much more general. Also, for notational simplicity, we shall consider a network having a single output y, although again this is not a significant limitation. We showed in Section 6.1.3 that the sum-of-squares error, in the limit of an infinite data set, can be written in the form

$$E = \frac{1}{2}\int \{y(\mathbf{x}) - \langle t|\mathbf{x}\rangle\}^2 p(\mathbf{x})\,d\mathbf{x} + \frac{1}{2}\int \{\langle t^2|\mathbf{x}\rangle - \langle t|\mathbf{x}\rangle^2\} p(\mathbf{x})\,d\mathbf{x} \tag{9.1}$$

in which $p(\mathbf{x})$ is the unconditional density of the input data, and $\langle t|\mathbf{x}\rangle$ denotes the conditional average, or regression, of the target data given by

$$\langle t|\mathbf{x}\rangle \equiv \int t p(t|\mathbf{x})\,dt \tag{9.2}$$

where $p(t|\mathbf{x})$ is the conditional density of the target variable t conditioned on the input vector $\mathbf{x}$. Similarly

$$\langle t^2|\mathbf{x}\rangle \equiv \int t^2 p(t|\mathbf{x})\,dt. \tag{9.3}$$

Note that the second term in (9.1) is independent of the network function $y(\mathbf{x})$ and hence is independent of the network weights. The optimal network function $y(\mathbf{x})$, in the sense of minimizing the sum-of-squares error, is the one which makes the first term in (9.1) vanish, and is given by $y(\mathbf{x}) = \langle t|\mathbf{x}\rangle$. The second term represents the intrinsic noise in the data and sets a lower limit on the error which can be achieved.

In a practical situation we must deal with the problems arising from a finite-size data set. Suppose we consider a training set D consisting of N patterns which we use to determine our network model $y(\mathbf{x})$. Now consider a whole ensemble of possible data sets, each containing N patterns, and each taken from the same fixed joint distribution $p(\mathbf{x}, t)$. We have already argued that the optimal network mapping is given by the conditional average $\langle t|\mathbf{x}\rangle$. A measure of how close the actual mapping function $y(\mathbf{x})$ is to the desired one is given by the integrand of the first term in (9.1):

$$\{y(\mathbf{x}) - \langle t|\mathbf{x}\rangle\}^2. \tag{9.4}$$

The value of this quantity will depend on the particular data set D on which it is trained. We can eliminate this dependence by considering an average over the complete ensemble of data sets, which we write as

$$\mathcal{E}_D[\{y(\mathbf{x}) - \langle t|\mathbf{x}\rangle\}^2] \tag{9.5}$$

where $\mathcal{E}_D[\cdot]$ denotes the expectation, or ensemble average, and we recall that the function $y(\mathbf{x})$ depends on the particular data set D which is used for training. Note that this expression is itself a function of $\mathbf{x}$.

If the network function were always a perfect predictor of the regression function $\langle t|\mathbf{x}\rangle$ then this error would be zero. As we shall see, a non-zero error can arise for essentially two distinct reasons. It may be that the network function is on average different from the regression function. This is called *bias*. Alternatively, it may be that the network function is very sensitive to the particular data set D, so that, at a given $\mathbf{x}$, it is larger than the required value for some data sets, and smaller for other data sets. This is called *variance*. We can make the decomposition into bias and variance explicit by writing (9.5) in somewhat different, but mathematically equivalent, form. First we expand the term in curly brackets in (9.5) to give

$$\{y(\mathbf{x}) - \langle t|\mathbf{x}\rangle\}^2 = \{y(\mathbf{x}) - \mathcal{E}_D[y(\mathbf{x})] + \mathcal{E}_D[y(\mathbf{x})] - \langle t|\mathbf{x}\rangle\}^2$$

$$= \{y(\mathbf{x}) - \mathcal{E}_D[y(\mathbf{x})]\}^2 + \{\mathcal{E}_D[y(\mathbf{x})] - \langle t|\mathbf{x}\rangle\}^2$$
$$+2\{y(\mathbf{x}) - \mathcal{E}_D[y(\mathbf{x})]\}\{\mathcal{E}_D[y(\mathbf{x})] - \langle t|\mathbf{x}\rangle\}. \tag{9.6}$$

In order to compute the expression in (9.5) we take the expectation of both sides of (9.6) over the ensemble of data sets D. We see that the third term on the right-hand side of (9.6) vanishes, and we are left with

$$\mathcal{E}_D[\{y(\mathbf{x}) - \langle t|\mathbf{x}\rangle\}^2]$$
$$= \underbrace{\{\mathcal{E}_D[y(\mathbf{x})] - \langle t|\mathbf{x}\rangle\}^2}_{(\text{bias})^2} + \underbrace{\mathcal{E}_D[\{y(\mathbf{x}) - \mathcal{E}_D[y(\mathbf{x})]\}^2]}_{\text{variance}}. \tag{9.7}$$

It is worth studying the expressions in (9.7) closely. The bias measures the extent to which the average (over all data sets) of the network function differs from the desired function $\langle t|\mathbf{x}\rangle$. Conversely the variance measures the extent to which the network function $y(\mathbf{x})$ is sensitive to the particular choice of data set. Note that the expressions for bias and variance are functions of the input vector $\mathbf{x}$. We can also introduce corresponding average values for bias and variance by integrating over all $\mathbf{x}$. By referring back to (9.1) we see that the appropriate weighting for this integration is given by the unconditional density $p(\mathbf{x})$, so that

$$(\text{bias})^2 = \frac{1}{2}\int \{\mathcal{E}_D[y(\mathbf{x})] - \langle t|\mathbf{x}\rangle\}^2 p(\mathbf{x})\,d\mathbf{x} \tag{9.8}$$

$$\text{variance} = \frac{1}{2}\int \mathcal{E}_D[\{y(\mathbf{x}) - \mathcal{E}_D[y(\mathbf{x})]\}^2]p(\mathbf{x})\,d\mathbf{x}. \tag{9.9}$$

The meaning of the bias and variance terms can be illustrated by considering two extreme limits for the choice of functional form for $y(\mathbf{x})$. We shall suppose that the target data for network training is generated from a smooth function $h(\mathbf{x})$ to which zero mean random noise ϵ is added, so that

$$t^n = h(\mathbf{x}^n) + \epsilon^n. \tag{9.10}$$

Note that the optimal mapping function in this case is given by $\langle t|\mathbf{x}\rangle = h(\mathbf{x})$. One choice of model for $y(\mathbf{x})$ would be some fixed function $g(\mathbf{x})$ which is completely independent of the data set D, as indicated in Figure 9.1. It is clear that the variance term in (9.7) will vanish, since $\mathcal{E}_D[y(\mathbf{x})] = g(\mathbf{x}) = y(\mathbf{x})$. However, the bias term will typically be high since no attention at all was paid to the data, and so unless we have some prior knowledge which helps us to choose the function $g(\mathbf{x})$ we are making a wild guess.

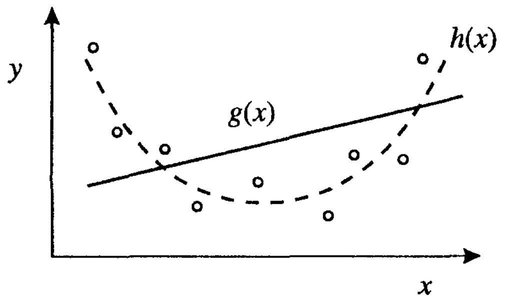

Figure 9.1. A schematic illustration of the meaning of bias and variance. Circles denote a set of data points which have been generated from an underlying function $h(x)$ (dashed curve) with the addition of noise. The goal is to try to approximate $h(x)$ as closely as possible. If we try to model the data by a fixed function $g(x)$, then the bias will generally be high while the variance will be zero.

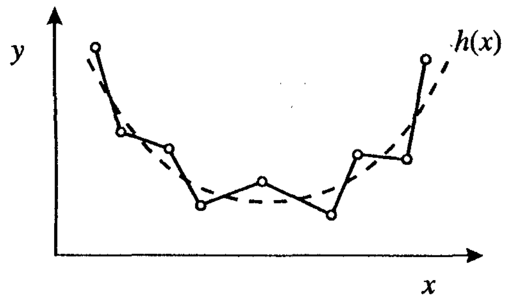

Figure 9.2. As in Figure 9.1, but in which a model is used which is a simple exact interpolant of the data points. In this case the bias is low but the variance is high.

The opposite extreme is to take a function which fits the training data perfectly, such as the simple exact interpolant indicated in Figure 9.2. In this case the bias term vanishes at the data points themselves since

$$\mathcal{E}_D[y(\mathbf{x})] = \mathcal{E}_D[h(\mathbf{x}) + \epsilon] = h(\mathbf{x}) = \langle t|\mathbf{x}\rangle \tag{9.11}$$

and the bias will typically be small in the neighbourhood of the data points. The variance, however, will be significant since

$$\mathcal{E}_D[\{y(\mathbf{x}) - \mathcal{E}_D[y(\mathbf{x})]\}^2] = \mathcal{E}_D[\{y(\mathbf{x}) - h(\mathbf{x})\}^2] = \mathcal{E}_D[\epsilon^2] \qquad (9.12)$$

which is just the variance of the noise on the data, which could be substantial.

We see that there is a natural trade-off between bias and variance. A function which is closely fitted to the data set will tend to have a large variance and hence give a large expected error. We can decrease the variance by smoothing the function, but if this is taken too far then the bias becomes large and the expected error is again large. This trade-off between bias and variance plays a crucial role in the application of neural network techniques to practical problems. We shall give a simple example of the dependence of bias and variance on the effective model complexity in Section 9.8.1.

9.1.1 *Minimizing bias and variance*

We have seen that, for any given size of data set, there is some optimal balance between bias and variance which gives the smallest average generalization error. In order to improve the performance of the network further we need to be able to reduce the bias while at the same time also reducing the variance. One way to achieve this is to use more data points. As we increase the number of data points we can afford to use more complex models, and therefore reduce bias, while at the same time ensuring that each model is more heavily constrained by the data, thereby also reducing variance. If we increase the number of data points sufficiently rapidly in relation to the model complexity we can find a sequence of models such that both bias and variance decrease. Models such as feed-forward neural networks can in principle provide *consistent* estimators of the regression function, meaning that they can approximate the regression to arbitrary accuracy in the limit as the number of data points goes to infinity. This limit requires a subtle balance of network complexity against number of data points to ensure that at each step both bias and variance are decreased. Consistency has been widely studied in the context of conventional techniques for statistical pattern recognition. For feed-forward networks, White (1990) has shown how the complexity of a two-layer network must grow in relation to the size of the data set in order to be consistent. This does not, however, tell us the complexity required for any given number of data points. It also requires that the parameter optimization algorithms are capable of finding the global minimum of the error function. Note that, even if both bias and variance can be reduced to zero, the error on new data will still be non-zero as a result of the intrinsic noise on the data given by the second term in (9.1).

In practice we are often limited in the number of training patterns available, and in many applications this may indeed be a severe limitation. An alternative approach to reducing both bias and variance becomes possible if we have some prior knowledge concerning the unknown function $h(\mathbf{x})$. Such knowledge can be used to constrain the model function $y(\mathbf{x})$ in a way which is consistent with $h(\mathbf{x})$ and which therefore does not give rise to increased bias. Note that the bias-variance problem implies that, for example, a simple linear model (single-layer network) might, in some applications involving relatively small data sets, give superior performance to a more general non-linear model (such as a multi-layer network) even though the latter contains the linear model as a special case.

9.2 Regularization

In Section 1.5 we saw that a polynomial with an excess of free coefficients tends to generate mappings which have a lot of curvature and structure, as a result of over-fitting to the noise on the training data. Similar behaviour also arises with more complex non-linear neural network models. The technique of regularization encourages smoother network mappings by adding a penalty Ω to the error function to give

$$\widetilde{E} = E + \nu\Omega. \tag{9.13}$$

Here E is one of the standard error functions as discussed in Chapter 6, and the parameter ν controls the extent to which the penalty term Ω influences the form of the solution. Training is performed by minimizing the total error function $\widetilde{E}$, which requires that the derivatives of Ω with respect to the network weights can be computed efficiently. A function $\mathbf{y}(\mathbf{x})$ which provides a good fit to the training data will give a small value for E, while one which is very smooth will give a small value for Ω. The resulting network mapping is a compromise between fitting the data and minimizing Ω. Regularization is discussed in the context of radial basis function networks in Section 5.4, and is given a Bayesian interpretation in Section 10.1.

In this section we shall consider various forms for the regularization term Ω. Regularization techniques have been extensively studied in the context of linear models for $\mathbf{y}(\mathbf{x})$. For the case of one input variable x and one output variable y, the class of *Tikhonov* regularizers takes the form

$$\Omega = \frac{1}{2}\sum_{r=0}^{R}\int_a^b h_r(x)\left(\frac{d^r y}{dx^r}\right)^2 dx \tag{9.14}$$

where $h_r \geq 0$ for $r = 0, \ldots, R-1$, and $h_R > 0$ (Tikhonov and Arsenin, 1977). Regularization has also been widely studied in the context of vision systems (Poggio *et al.*, 1985).

9.2.1 *Weight decay*

One of the simplest forms of regularizer is called *weight decay* and consists of the sum of the squares of the adaptive parameters in the network

$$\Omega = \frac{1}{2}\sum_i w_i^2 \tag{9.15}$$

where the sum runs over all weights and biases. In conventional curve fitting, the use of this form of regularizer is called *ridge regression*. It has been found