13 Generalization

Objectives

One of the key issues in designing a multilayer network is determining the number of neurons to use. In effect, that is the objective of this chapter.

In Chapter 11 we showed that if the number of neurons is too large, the network will overfit the training data. This means that the error on the training data will be very small, but the network will fail to perform as well when presented with new data. A network that generalizes well will perform as well on new data as it does on the training data.

The complexity of a neural network is determined by the number of free parameters that it has (weights and biases), which in turn is determined by the number of neurons. If a network is too complex for a given data set, then it is likely to overfit and to have poor generalization.

In this chapter we will see that we can adjust the complexity of a network to fit the complexity of the data. In addition, this can be done without changing the number of neurons. We can adjust the effective number of free parameters without changing the actual number of free parameters.

Theory and Examples

Mark Twain once said "We should be careful to get out of an experience only the wisdom that is in it-and stop there; lest we be like the cat that sits down on a hot stove-lid. She will never sit down on a hot stove-lid again-and that is well; but also she will never sit down on a cold one any more." (From *Following the Equator*, 1897.)

Generalization

That is the objective of this chapter. We want to train neural networks to get out of the data only the wisdom that is in it. This concept is called *generalization*. A network trained to generalize will perform as well in new situations as it does on the data on which it was trained.

Ockham's Razor

The key strategy we will use for obtaining good generalization is to find the simplest model that explains the data. This is a variation of a principle called *Ockham's razor*, which is named after the English logician William of Ockham, who worked in the 14th Century. The idea is that the more complexity you have in your model, the greater the possibility for errors.

In terms of neural networks, the simplest model is the one that contains the smallest number of free parameters (weights and biases), or, equivalently, the smallest number of neurons. To find a network that generalizes well, we need to find the simplest network that fits the data.

There are at least five different approaches that people have used to produce simple networks: growing, pruning, global searches, regularization, and early stopping. Growing methods start with no neurons in the network and then add neurons until the performance is adequate. Pruning methods start with large networks, which likely overfit, and then remove neurons (or weights) one at a time until the performance degrades significantly. Global searches, such as genetic algorithms, search the space of all possible network architectures to locate the simplest model that explains the data.

The final two approaches, regularization and early stopping, keep the network small by constraining the *magnitude* of the network weights, rather than by constraining the *number* of network weights. In this chapter we will concentrate on these two approaches. We will begin by defining the problem of generalization and by showing examples of both good and poor generalization. We will then describe the regularization and early stopping methods for training neural networks. Finally, we will demonstrate how these two methods are, in affect, performing the same operation.

Problem Statement

Let's begin our discussion of generalization by defining the problem. We start with a training set of example network inputs and corresponding target outputs:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}. \quad (13.1)$$

For our development of the concept of generalization, we will assume that the target outputs are generated by

$$\mathbf{t}_q = \mathbf{g}(\mathbf{p}_q) + \varepsilon_q, \quad (13.2)$$

where $\mathbf{g}(.)$ is some unknown function, and ε_q is a random, independent and zero mean noise source. Our training objective will be to produce a neural network that approximates $\mathbf{g}(.)$, while ignoring the noise.

The standard performance index for neural network training is the sum squared error on the training set:

$$F(\mathbf{x}) = E_D = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q), \quad (13.3)$$

where $\mathbf{a}_q$ is the network output for input $\mathbf{p}_q$. We are using the variable E_D to represent the sum squared error on the training data, because later we will modify the performance index to include an additional term.

Overfitting

The problem of *overfitting* is illustrated in Figure 13.1. The blue curve represents the function $\mathbf{g}(.)$. The large open circles represent the noisy target points. The black curve represents the trained network response, and the smaller circles filled with crosses represent the network response at the training points. In this figure we can see that the network response exactly matches the training points. However, it does a very poor job of matching the underlying function. It overfits.

There are actually two kinds of errors that occur in Figure 13.1. The first type of error, which is caused by overfitting, occurs for input values between -3 and 0. This is the region where all of the training data points occur. The network response in this region overfits the training data and will fail to perform well for input values that are not in the training set. The network does a poor job of *interpolation*; it fails to accurately approximate the function near the training points.

Interpolation

The second type of error occurs for inputs in the region between 0 and 3. The network fails to perform well in this region, not because it is overfitting, but because there is no training data there. The network is *extrapolating* beyond the range of the input data.

Extrapolation

In this chapter we will discuss methods for preventing overfitting. *There is no way to prevent errors of extrapolation.* It is very important that the data that is used to train the network cover the regions of the input space for which the network will be used. The network has no way of knowing what the true function looks like in regions for which there is no data.

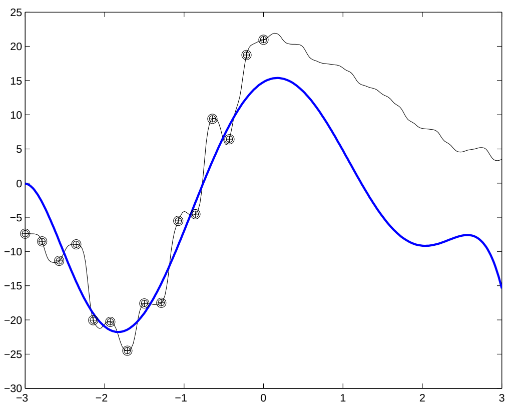

Figure 13.1 Example of Overfitting and Poor Extrapolation

In Figure 13.2 we have an example of a network that has been trained to generalize well. The network has the same number of weights as the network of Figure 13.1, and it was trained using the same data set, but it has been trained in such a way that it does not fully use all of the weights that are available. It only uses as many weights as necessary to fit the data. The network response does not fit the function perfectly, but it does the best job it can, based on limited and noisy data.

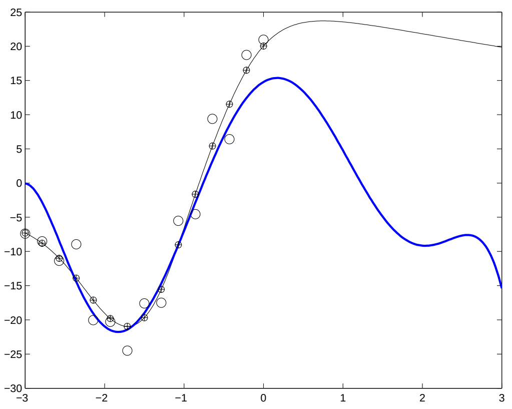

Figure 13.2 Example of Good Interpolation and Poor Extrapolation

In both Figure 13.1 and Figure 13.2 we can see that the network fails to extrapolate accurately. This is understandable, since the network has been provided with no information about the characteristics of the function outside of the range $-3 \le p \le 0$. The network response outside this range will be unpredictable. This is why it is important to have training data for all regions of the input space for which the network will be used. It is usually not difficult to determine the required input range when the network has a single input, as in this example. However, when the network has many inputs, it becomes more difficult to determine when the network is interpolating and when it is extrapolating.

This problem is illustrated in a simple way in Figure 13.3. On the left side of this figure we see the function that is to be approximated. The range for the input variables is $-3 \le p_1 \le 3$ and $-3 \le p_2 \le 3$. The neural network was trained over these ranges of the two variables, but only for $p_1 \le p_2$. Therefore, both p_1 and p_2 cover their individual ranges, but only half of the total input space is covered. When $p_1 \ge p_2$, the network is extrapolating, and we can see on the right side of Figure 13.3 that the network performs poorly in this region. (See Problem P13.4 for another example of extrapolation.) If there are many input variables, it will be quite difficult to determine when the network is interpolating and when it is extrapolating. We will discuss some practical ways of dealing with this problem in Chapter 22.

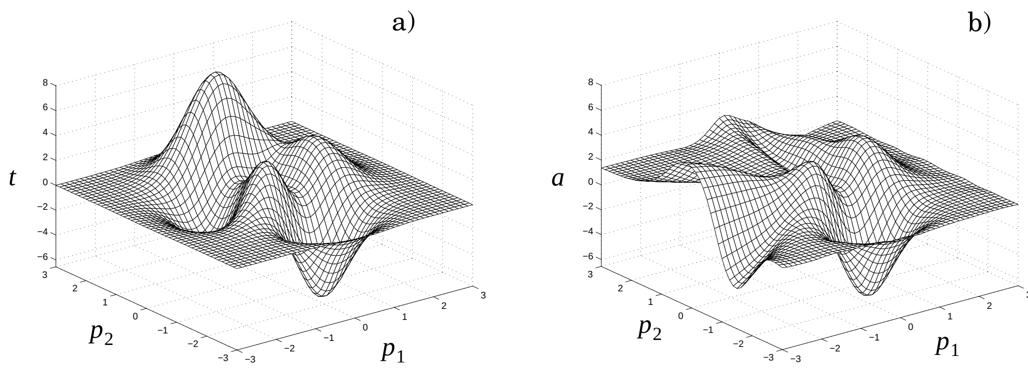

Figure 13.3 Function (a) and Neural Network Approximation (b)

Methods for Improving Generalization

The remainder of this chapter will discuss methods for improving the generalization capability of neural networks. As we discussed earlier, there are a number of approaches to this problem - all of which try to find the simplest network that will fit the data. These approaches fit into two general categories: restricting the number of weights (or, equivalently, the number of neurons) in the network, or restricting the magnitude of the weights. We will concentrate on two methods that we have found to be particularly useful: early stopping and regularization. Both of these approaches attempt to restrict the magnitude of the weights, although they do so in very different ways. At the end of this chapter, we will demonstrate the approximate equivalence of the two methods.

We should note that in this chapter we are assuming that there is a limited amount of data with which to train the network. If the amount of data is unlimited, which in practical terms means that the number of data points is significantly larger than the number of network parameters, then there will not be a problem of overfitting.

Estimating Generalization Error - The Test Set

Before we discuss methods for improving the generalization capability of neural networks, we should first discuss how we can estimate this error for a specific neural network. Given a limited amount of available data, it is important to hold aside a certain subset during the training process. After the network has been trained, we will compute the errors that the trained network makes on this *test set*. The test set errors will then give us an indication of how the network will perform in the future; they are a measure of the generalization capability of the network.

Test Set

In order for the test set to be a valid indicator of generalization capability, there are two important things to keep in mind. First, the test set must never be used in any way to train the neural network, or even to select one network from a group of candidate networks. The test set should only be used after all training and selection is complete. Second, the test set must be representative of all situations for which the network will be used. This can sometimes be difficult to guarantee, especially when the input space is high-dimensional or has a complex shape. We will discuss this problem in more detail in Chapter 22, Practical Training Issues.

In the remaining sections of this chapter, we will assume that a test set has been removed from the data set before training begins, and that this set will be used at the completion of training to measure generalization capability.

Early Stopping

The first method we will discuss for improving generalization is also the simplest method. It is called early stopping [WaVe94]. The idea behind this method is that as training progresses the network uses more and more of its weights, until all weights are fully used when training reaches a minimum of the error surface. By increasing the number of iterations of training, we are increasing the complexity of the resulting network. If training is stopped before the minimum is reached, then the network will effectively be using fewer parameters and will be less likely to overfit. In a later section of this chapter we will demonstrate how the number of parameters changes as the number of iterations increases.

Cross-Validation

In order to use early stopping effectively, we need to know when to stop the training. We will describe a method, called *cross-validation*, that uses a

Validation Set

validation set to decide when to stop [Sarl95]. The available data (after removing the test set, as described above) is divided into two parts: a training set and a validation set. The training set is used to compute gradients or Jacobians and to determine the weight update at each iteration. The validation set is an indicator of what is happening to the network function "in between" the training points, and its error is monitored during the training process. When the error on the validation set goes up for several iterations, the training is stopped, and the weights that produced the minimum error on the validation set are used as the final trained network weights.

This process is illustrated in Figure 13.4. The graph at the bottom of this figure shows the progress of the training and validation performance indices, F (the sum squared errors), during training. Although the training error continues to go down throughout the training process, a minimum of the validation error occurs at the point labeled "a," which corresponds to training iteration 14. The graph at the upper left shows the network response at this *early stopping* point. The resulting network provides a good fit to the true function. The graph at the upper right demonstrates the network response if we continue to train to point "b," where the validation error has increased and the network is overfitting.

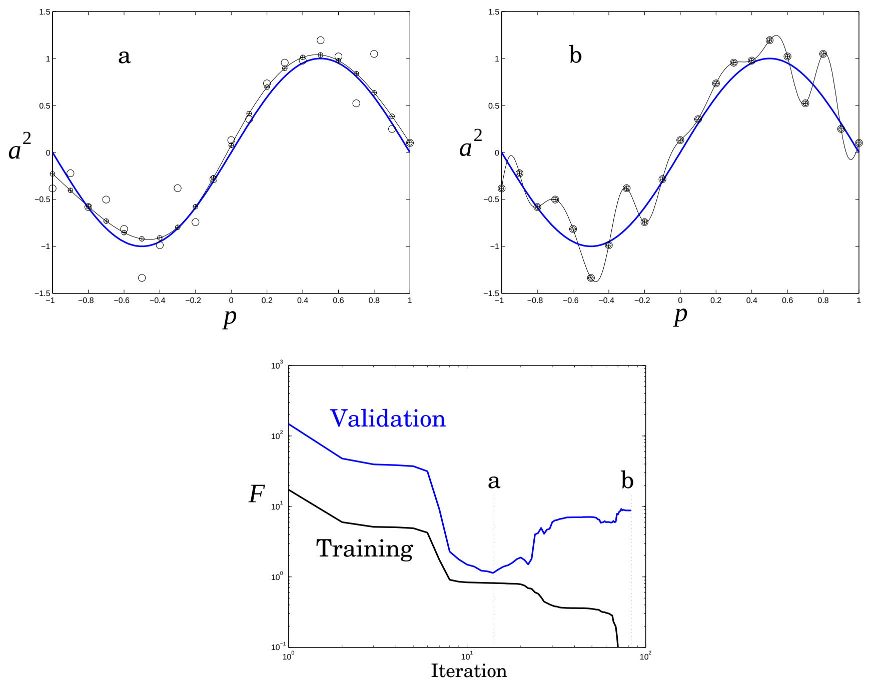

Figure 13.4 Illustration of Early Stopping

The basic concept for early stopping is simple, but there are several practical issues to be addressed. First, the validation set must be chosen so that it is representative of all situations for which the network will be used. This is also true for the test and training sets, as we mentioned earlier. Each set must be roughly equivalent in its coverage of the input space, although the size of each set may be different.

When we divide the data, approximately 70% is typically used for training, with 15% for validation and 15% for testing. These are only approximate numbers. A complete discussion of how to select the amount of data for the validation set is given in [AmMu97].

Another practical point to be made about early stopping is that we should use a relatively slow training method. During training, the network will use more and more of the available network parameters (as we will explain in the last section of this chapter). If the training method is too fast, it will likely jump past the point at which the validation error is minimized.

To experiment with the effect of early stopping, use the MATLAB® Neural Network Design Demonstration Early Stopping (**nnd13es**).

Regularization

The second method we will discuss for improving generalization is called regularization. For this method, we modify the sum squared error performance index of Eq. (13.3) to include a term that penalizes network complexity. This concept was introduced by Tikhonov [Tikh63]. He added a penalty, or regularization, term related to the derivatives of the approximating function (neural network in our case), which forced the resulting function to be smooth. Under certain conditions, this regularization term can be written as the sum of squares of the network weights, as in

$$F(\mathbf{x}) = \beta E_D + \alpha E_W = \beta \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q) + \alpha \sum_{i=1}^{n} x_i^2 , \qquad (13.4)$$

where the ratio α/β controls the effective complexity of the network solution. The larger this ratio is, the smoother the network response. (Note that we could have used a single parameter here, but developments in later sections will require two parameters.)

Why do we want to penalize the sum squared weights, and how is this similar to reducing the number of neurons? Consider again the example multilayer network shown in Figure 11.4. Recall how increasing a weight increased the slope of the network function. You can see this effect again in Figure 13.5, where we have changed the weight $w^2_{1,1}$ from 0 to 2. When the weights are large, the function created by the network can have large slopes, and is therefore more likely to overfit the training data. If we restrict the weights to be small, then the network function will create a smooth interpolation through the training data - just as if the network had a small number of neurons.

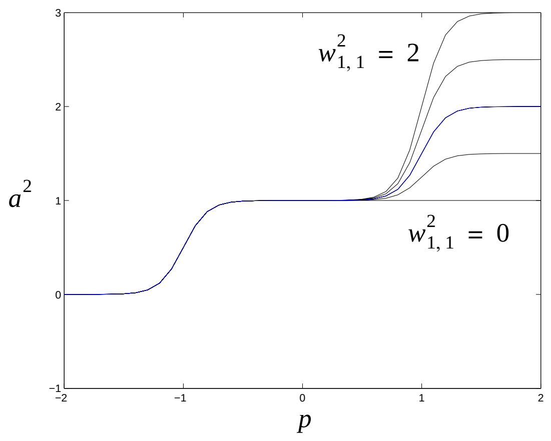

Figure 13.5 Effect of Weight on Network Response

To experiment with the effect of weight changes on the network function, use the MATLAB® Neural Network Design Demonstration Network Function (**`nnd11nf`**).

The key to the success of the regularization method in producing a network that generalizes well is the correct choice of the regularization ratio α/β. Figure 13.6 illustrates the effect of changing this ratio. Here we have trained a 1-20-1 network on 21 noisy samples of a sine wave.

In the figure, the blue line represents the true function, and the large open circles represent the noisy data. The black curve represents the trained network response, and the smaller circles filled with crosses represent the network response at the training points. From the figure, we can see that the ratio $\alpha/\beta = 0.01$ produces the best fit to the true function. For ratios larger than this, the network response is too smooth, and for ratios smaller than this, the network overfits.

There are several techniques for setting the regularization parameter. One approach is to use a validation set, such as we described in the section on early stopping; the regularization parameter is set to minimize the squared error on the validation set [GoLa98]. In the next two sections we will describe a different technique for automatically setting the regularization parameter. It is called Bayesian regularization.

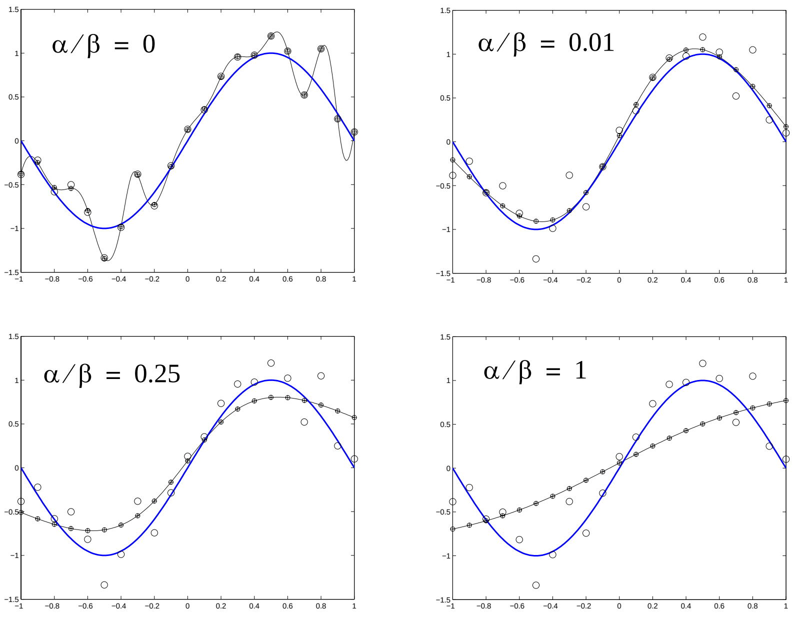

Figure 13.6 Effect of Regularization Ratio

To experiment with the effect of regularization, use the MATLAB® Neural Network Design Demonstration Regularization (**`nnd13reg`**).

Bayesian Analysis

Thomas Bayes was a Presbyterian minister who lived in England during the 1700's. He was also an amateur mathematician. His most important work was published after his death. In it, he presented what is now known as Bayes' Theorem. The theorem states that if you have two random events, A and B, then the conditional probability of the occurrence of A, given the occurrence of B can be computed as

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{13.5}$$

Eq. (13.5) is called Bayes' rule. Each of the terms in this expression has a name by which it is commonly referred. $P(A)$ is called the *prior* probability. It tells us what we know about A before we know the outcome of B. $P(A|B)$ is called the *posterior* probability. This tells us what we know about A after we learn about B. $P(B|A)$ is the conditional probability of B given A. Normally this term is given by our knowledge of the system that describes the relationship between B and A. $P(B)$ is the marginal probability of the event B, and it acts as a normalization factor in Bayes' rule.

To illustrate how Bayes' rule can be used, consider the following medical situation. Assume that 1% of the population have a certain disease. There is a test that can be performed to detect the presence of this disease. The test is 80% accurate in detecting the disease in people who have it. However, 10% of the time, someone without the disease will register a positive test. If you take the test and register positive, your question would be: What is the probability that I actually have the disease? Most of us (including most physicians, as has been shown in many studies), would guess that the probability is very high, considering that the test is 80% accurate in detecting the disease in a sick person. However, this turns out not to be the case, and Bayes' rule can help us overcome this lack of intuition, when it comes to probability.

Let A represent the event that you have the disease. Let B represent the event that you have a positive test result. We can then use Bayes' rule to find $P(A|B)$, which is the probability that you have the disease, given that you have a positive test. We know that the prior probability $P(A)$ would be 0.01, because 1% of the population have the disease. $P(B|A)$ is 0.8, because the test is 80% accurate in detecting the disease in people who have it. (Notice that this conditional probability is based on our knowledge of the test procedure and its accuracy.) In order to use Bayes' rule, we need one more term, which is $P(B)$. This is the probability of getting a positive test, whether or not you have the disease. This can be obtained by adding the probability of having a positive test when you have the disease to the probability of having a positive test when you don't have the disease:

$$P(B) = P(A \cap B) + P(\bar{A} \cap B) = P(B|A)P(A) + P(B|\bar{A})P(\bar{A}), \qquad (13.6)$$

where we have used the definition of conditional probability:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}, \text{ or } P(A \cap B) = P(B|A)P(A). \qquad (13.7)$$

If we plug in our known probabilities into Eq. (13.6), we find

$$P(B) = 0.8 \times 0.01 + 0.1 \times 0.99 = 0.107, \qquad (13.8)$$

where $P(B|\bar{A})$ is 0.1, because 10% of health people register a positive test. We can now use Bayes' rule to find the posterior probability $P(A|B)$:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{0.8 \times 0.01}{0.107} = 0.0748. \qquad (13.9)$$

This tells us that even if you get a positive test, you only have a 7.5% chance of having the disease. For most of us, this result is not intuitive.

The key to Bayes' rule is the prior probability $P(A)$. In this case, the prior odds of having the disease were only 1 in 100. If this number had been much higher, then our posterior probability $P(A|B)$ would have also increased significantly. It is important when using Bayes' rule to have the prior probability $P(A)$ accurately reflect our prior knowledge.

For another example of using Bayes' rule and the effect of the prior density, see Solved Problem P13.2 and its associated demonstration.

In the next section, we will apply Bayesian analysis to the training of multilayer networks. The advantage of Bayesian methods is that we can insert prior knowledge through the selection of the prior probability. For neural network training, we will make the prior assumption that the function we are approximating is smooth. This means that the weights cannot be too large, as was demonstrated in Figure 13.5. The trick will be to incorporate this prior knowledge into an appropriate choice for the prior probability.

Bayesian Regularization

Although there have been many approaches to the automatic selection of the regularization parameter, we will concentrate on one developed by David MacKay [MacK92]. This approach puts the training of neural networks into a Bayesian statistical framework. This framework is useful for many aspects of training, in addition to the selection of the regularization parameter, so it is an important concept to become familiar with. There are two levels to this Bayesian analysis. We will begin with Level I.

Level I Bayesian Framework

The Bayesian framework begins with the assumption that the network weights are random variables. We then choose the weights that maximize the conditional probability of the weights given the data. Bayes' rule is used to find this probability function:

$$P(\mathbf{x}|D,\alpha,\beta,M) = \frac{P(D|\mathbf{x},\beta,M)P(\mathbf{x}|\alpha,M)}{P(D|\alpha,\beta,M)}, \qquad (13.10)$$

where $\mathbf{x}$ is the vector containing all of the weights and biases in the network, D represents the training data set, α and β are parameters associated with the density functions $P(D|\mathbf{x},\beta,M)$ and $P(\mathbf{x}|\alpha,M)$, and M is the selected model - the architecture of the network we have chosen (i.e., how many layers and how may neurons in each layer).

It is worth taking some time to investigate each of the terms in Eq. (13.10). First, $P(D|\mathbf{x},\beta,M)$ is the probability density for the data, given a certain set of weights $\mathbf{x}$, the parameter β (which we will explain shortly), and the choice of model M. If we assume that the noise terms in Eq. (13.2) are independent and have a Gaussian distribution, then

$$P(D|\mathbf{x},\beta,M) = \frac{1}{Z_D(\beta)}\exp(-\beta E_D), \qquad (13.11)$$

where $\beta = 1/(2\sigma_\varepsilon^2)$, σ_ε^2 is the variance of each element of ε_q, E_D is the squared error (as defined in Eq. (13.3)), and

$$Z_D(\beta) = (2\pi\sigma_\varepsilon^2)^{N/2} = (\pi/\beta)^{N/2}, \tag{13.12}$$

where N is $Q \times S^M$, as in Eq. (12.34).

Likelihood Function

Maximum Likelihood

Eq. (13.11) is called the *likelihood function*. It is a function of the network weights $\mathbf{x}$, and it describes how likely a given data set is to occur, given a specific set of weights. The *maximum likelihood* method selects the weights so as to maximize the likelihood function, which in this Gaussian case is the same as minimizing the squared error E_D. Therefore, our standard sum squared error performance index can be derived statistically with the assumption of Gaussian noise in the training set, and our standard choice for the weights is the maximum likelihood estimate.

Prior Density

Now consider the second term on the right side of Eq. (13.10): $P(\mathbf{x}|\alpha, M)$. This is called the *prior density*. It embodies our knowledge about the network weights before we collect any data. Bayesian statistics allows us to incorporate prior knowledge through the prior density. For example, if we assume that the weights are small values centered around zero, we might select a zero-mean Gaussian prior density:

$$P(\mathbf{x}\,|\,\alpha, M) = \frac{1}{Z_W(\alpha)}\exp(-\alpha E_W) \tag{13.13}$$

where $\alpha = 1/(2\sigma_w^2)$, σ_w^2 is the variance of each of the weights, E_W is the sum squared weights (as defined in Eq. (13.4)), and

$$Z_W(\alpha) = (2\pi\sigma_w^2)^{n/2} = (\pi/\alpha)^{n/2}, \tag{13.14}$$

where n is the number of weights and biases in the network, as in Eq. (12.35).

Evidence

Posterior Density

The final term on the right side of Eq. (13.10) is $P(D|\alpha, \beta, M)$. This is called the *evidence*, and it is a normalizing term that is not a function of $\mathbf{x}$. If our objective is to find the weights $\mathbf{x}$ that maximize the *posterior density* $P(\mathbf{x}\,|\,D, \alpha, \beta, M)$, then we do not need to be concerned with $P(D|\alpha, \beta, M)$. (However, it will be important later for estimating α and β.)

With the Gaussian assumptions that we made earlier, we can rewrite the posterior density, using Eq. (13.10), in the following form:

$$P(\mathbf{x} \mid D, \alpha, \beta, M) = \frac{\frac{1}{Z_W(\alpha)} \frac{1}{Z_D(\beta)} \exp(-(\beta E_D + \alpha E_W))}{\text{Normalization Factor}} = \frac{1}{Z_F(\alpha, \beta)} \exp(-F(\mathbf{x})) \quad (13.15)$$

where $Z_F(\alpha, \beta)$ is a function of α and β (but not a function of $\mathbf{x}$), and $F(\mathbf{x})$ is our regularized performance index, which we defined in Eq. (13.4). To find the most probable value for the weights, we should maximize the posterior density $P(\mathbf{x} \mid D, \alpha, \beta, M)$. This is equivalent to minimizing the regularized performance index $F(\mathbf{x}) = \beta E_D + \alpha E_W$.

Most Probable

Therefore, our regularized performance index can be derived using Bayesian statistics, with the assumption of Gaussian noise in the training set and a Gaussian prior density for the network weights. We will identify the weights that maximize the posterior density as $\mathbf{x}^{MP}$, or *most probable*. This is to be contrasted with the weights that maximize the likelihood function: $\mathbf{x}^{ML}$.

Note how this statistical framework provides a physical meaning for the parameters α and β. The parameter β is inversely proportional to the variance in the measurement noise ε_q. Therefore, if the noise variance is large, β will be small, and the regularization ratio α/β will be large. This will force the resulting weights to be small and the network function to be smooth (as seen in Figure 13.6). The larger the measurement noise, the more we will smooth the network function, in order to average out the affects of the noise.

The parameter α is inversely proportional to the variance in the prior distribution for the network weights. If this variance is large, it means that we have very little certainty about the values of the network weights, and, therefore, they might be very large. The parameter α will then be small, and the regularization ratio α/β will also be small. This will allow the network weights to be large, and the network function will be allowed to have more variation (as seen in Figure 13.6). The larger the variance in the prior density for the network weights, the more variation the network function will be allowed to have.

Level II Bayesian Framework

So far we have an interesting statistical derivation of the regularized performance index and some new insight into the meanings of the parameters α and β, but what we really want to find is a way to estimate these parameters from the data. In order to do this, we need to take the Bayesian analysis to another level. If we want to estimate α and β using Bayesian analysis, we need the probability density $P(\alpha, \beta | D, M)$. Using Bayes' rule this can written

$$P(\alpha, \beta | D, M) = \frac{P(D | \alpha, \beta, M) P(\alpha, \beta | M)}{P(D | M)} . \tag{13.16}$$

This has the same format as Eq. (13.10), with the likelihood function and the prior density in the numerator of the right hand side. If we assume a uniform (constant) prior density $P(\alpha, \beta | M)$ for the regularization parameters α and β, then maximizing the posterior is achieved by maximizing the likelihood function $P(D | \alpha, \beta, M)$. However, note that this likelihood function is the normalization factor (evidence) from Eq. (13.10). Since we have assumed that all probabilities have a Gaussian form, we know the form for the posterior density of Eq. (13.10). It is shown in Eq. (13.15). Now we can solve Eq. (13.10) for the normalization factor (evidence).

$$\begin{aligned} P(D | \alpha, \beta, M) &= \frac{P(D | \mathbf{x}, \beta, M) P(\mathbf{x} \,|\, \alpha, M)}{P(\mathbf{x} \,|\, D, \alpha, \beta, M)} \\ &= \frac{\left[\frac{1}{Z_D(\beta)} \exp(-\beta E_D)\right]\left[\frac{1}{Z_W(\alpha)} \exp(-\alpha E_W)\right]}{\frac{1}{Z_F(\alpha, \beta)} \exp(-F(\mathbf{x}))} \\ &= \frac{Z_F(\alpha, \beta)}{Z_D(\beta) Z_W(\alpha)} \cdot \frac{\exp(-\beta E_D - \alpha E_W)}{\exp(-F(\mathbf{x}))} = \frac{Z_F(\alpha, \beta)}{Z_D(\beta) Z_W(\alpha)} \end{aligned} \tag{13.17}$$

Note that we know the constants $Z_D(\beta)$ and $Z_W(\alpha)$ from Eq. (13.12) and Eq. (13.14). The only part we do not know is $Z_F(\alpha, \beta)$. However, we can estimate it by using a Taylor series expansion.

Since the objective function has the shape of a quadratic in a small area surrounding a minimum point, we can expand $F(\mathbf{x})$ in a second order Taylor series (see Eq. (8.9)) around its minimum point, $\mathbf{x}^{MP}$, where the gradient is zero:

$$F(\mathbf{x}) \approx F(\mathbf{x}^{MP}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{MP})^T \mathbf{H}^{\mathrm{MP}} (\mathbf{x} - \mathbf{x}^{MP}) , \tag{13.18}$$

where $\mathbf{H} = \beta \nabla^2 E_D + \alpha \nabla^2 E_W$ is the Hessian matrix of $F(\mathbf{x})$, and $\mathbf{H}^{MP}$ is the Hessian evaluated at $\mathbf{x}^{MP}$. We can now substitute this approximation into the expression for the posterior density, Eq. (13.15):

$$P(\mathbf{x} | D, \alpha, \beta, M) \approx \frac{1}{Z_F} \exp\left[- F(\mathbf{x}^{MP}) - \frac{1}{2}(\mathbf{x} - \mathbf{x}^{MP})^T \mathbf{H}^{MP} (\mathbf{x} - \mathbf{x}^{MP})\right], \tag{13.19}$$

which can be rewritten as

$$P(\mathbf{x}|D, \alpha, \beta, M) \approx \left\{\frac{1}{Z_F}\exp(-F(\mathbf{x}^{MP}))\right\}\exp\left[-\frac{1}{2}(\mathbf{x}-\mathbf{x}^{MP})^T\mathbf{H}^{MP}(\mathbf{x}-\mathbf{x}^{MP})\right]. \quad (13.20)$$

The standard form of the Gaussian density is

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n\left|(\mathbf{H}^{MP})^{-1}\right|}}\exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x}^{MP})^T\mathbf{H}^{MP}(\mathbf{x}-\mathbf{x}^{MP})\right). \quad (13.21)$$

Therefore, equating Eq. (13.21) with Eq. (13.20), we can solve for $Z_F(\alpha, \beta)$:

$$Z_F(\alpha, \beta) \approx (2\pi)^{n/2}(\det((\mathbf{H}^{MP})^{-1}))^{1/2}\exp(-F(\mathbf{x}^{MP})). \quad (13.22)$$

Placing this result into Eq. (13.17), we can solve for the optimal values for α and β at the minimum point. We do this by taking the derivative with respect to each of the log of Eq. (13.17) and set them equal to zero. This yields (see Solved Problem P13.3):

$$\alpha^{MP} = \frac{\gamma}{2E_W(\mathbf{x}^{MP})} \text{ and } \beta^{MP} = \frac{N-\gamma}{2E_D(\mathbf{x}^{MP})}, \quad (13.23)$$

Effective # of Parameters

where $\gamma = n - 2\alpha^{MP}\mathrm{tr}(\mathbf{H}^{MP})^{-1}$ is called the *effective number of parameters*, and n is the total number of parameters in the network. The term γ is a measure of how many parameters (weights and biases) in the neural network are effectively used in reducing the error function. It can range from zero to n. (See the example on page 13-23 for more analysis of γ.)

Bayesian Regularization Algorithm

The Bayesian optimization of the regularization parameters requires the computation of the Hessian matrix of $F(\mathbf{x})$ at the minimum point $\mathbf{x}^{MP}$. We propose using the Gauss-Newton approximation to the Hessian matrix [FoHa97], which is readily available if the Levenberg-Marquardt optimization algorithm is used to locate the minimum point (see Eq. (12.31)). The additional computation required for optimization of the regularization is minimal.

Here are the steps required for Bayesian optimization of the regularization parameters, with the Gauss-Newton approximation to Hessian matrix:

0. Initialize α, β and the weights. The weights are initialized randomly, and then E_D and E_W are computed. Set $\gamma = n$, and compute α and β using Eq. (13.23).
1. Take one step of the Levenberg-Marquardt algorithm toward minimizing the objective function $F(\mathbf{x}) = \beta E_D + \alpha E_W$.
2. Compute the effective number of parameters $\gamma = n - 2\alpha\mathrm{tr}(\mathbf{H})^{-1}$, mak-

ing use of the Gauss-Newton approximation to the Hessian available in the Levenberg-Marquardt training algorithm: $\mathbf{H} = \nabla^2 F(\mathbf{x}) \approx 2\beta \mathbf{J}^T \mathbf{J} + 2\alpha \mathbf{I}_n$, where $\mathbf{J}$ is the Jacobian matrix of the training set errors (see Eq. (12.37)).

3. Compute new estimates for the regularization parameters $\alpha = \frac{\gamma}{2E_W(\mathbf{x})}$ and $\beta = \frac{N-\gamma}{2E_D(\mathbf{x})}$.
4. Now iterate steps 1 through 3 until convergence.

Bear in mind that with each reestimate of the regularization parameters α and β the objective function $F(\mathbf{x})$ changes; therefore, the minimum point is moving. If traversing the performance surface generally moves toward the next minimum point, then the new estimates for the regularization parameters will be more precise. Eventually, the precision will be good enough that the objective function will not significantly change in subsequent iterations. Thus, we will obtain convergence.

GNBR

When this Gauss-Newton approximation to Bayesian regularization (*GNBR*) algorithm is used, the best results are obtained if the training data is first mapped into the range [-1,1] (or some similar region). We will discuss this preprocessing of the training data in Chapter 22.

In Figure 13.7 you can see the results of training a 1-20-1 network with GNBR on the same data set represented in Figure 13.4 and Figure 13.6. The network has fit the underlying function, without overfitting to the noise. The fit looks similar to that obtained in Figure 13.6, with the regularization ratio set to $\alpha/\beta = 0.01$. In fact, at the completion of training with GNBR, the final regularization ratio for this example was $\alpha/\beta = 0.0137$.

The training process for this example is illustrated in Figure 13.8. In the upper left of this figure, you see the squared error on the training set. Notice that it does not necessarily go down at each iteration. In the upper right of the figure, you see the squared testing error. This was obtained by comparing the network function to the true function at a number of points between -1 and 1. It is a measure of the generalization capability of the network. (This would not be possible in a practical case, where the true function was unknown.) Note that the testing error is at its minimum at the completion of training.

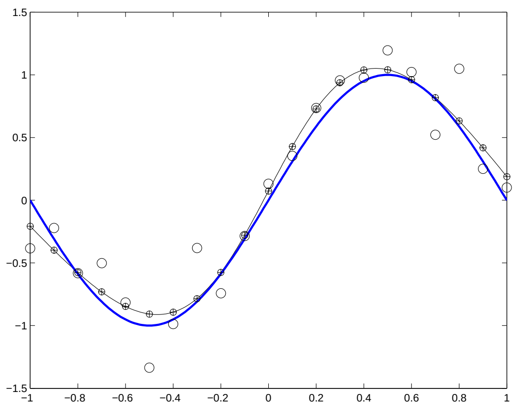

Figure 13.7 Bayesian Regularization Fit

Figure 13.8 also shows the regularization ratio α/β and the effective number of parameters γ during training. These parameters have no particular meaning during the training process, but at the completion of training they are significant. As we mentioned earlier, the final regularization ratio was $\alpha/\beta = 0.0137$, which is consistent with our earlier investigation of regularization - illustrated in Figure 13.6. The final effective number of parameters was $\gamma = 5.2$. This is out of a total of 61 total weights and biases in the network.

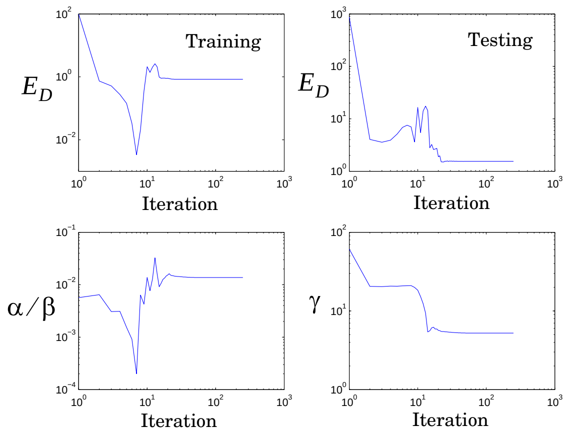

Figure 13.8 Bayesian Regularization Training Process

The fact that in this example the effective number of parameters is much less than the total number of parameters (6 versus 61) means that we might well have been able to use a smaller network to fit this data. There are two disadvantages of a large network: 1) it may overfit the data, and 2) it requires more computation to calculate the network output. We have overcome the first disadvantage by training with GNBR; although the network has 61 parameters, it is equivalent to a network with only 6 parameters. The second disadvantage is only important if the calculation time for the network response is critical to the application. This is not usually the case, since the time to calculate a network response to a particular input is measured in milliseconds. In those cases where the calculation time is significant, you can train a smaller network on the data.

On the other hand, when the effective number of parameters is close to the total number of parameters, this can mean that the network is not large enough to fit the data. In this case, you should increase the size of the network and retrain on the data set.

To experiment with Bayesian Regularization, use the MATLAB® Neural Network Design Demonstration Bayesian Regularization (**`nnd17breg`**).

Relationship Between Early Stopping and Regularization

We have discussed two techniques for improving network generalization: early stopping and regularization. These two methods were developed in very different ways, but they both improve generalization by restricting the network weights and, therefore, producing a network with fewer effective parameters. Early stopping restricts the network weights by stopping the training before the weights have converged to the minimum of the squared error. Regularization restricts the weights by adding a term to the squared error that penalizes large weights. In this section we want to demonstrate, using a linear example, an approximate equivalence of these two methods. During the process, we will also shed some light on the meaning of the effective number of parameters, γ. This development is based on the more general procedures described in [SjLj94].

Early Stopping Analysis

Consider the single layer linear network shown in Figure 10.1. We have shown in Eq. (10.12) and Eq. (10.14) that the mean square error performance function for this linear network is quadratic, of the form

$$F(\mathbf{x}) = c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} , \qquad (13.24)$$

where $\mathbf{A}$ is the Hessian matrix. In order to study the performance of early stopping, we will analyze the evolution of the steepest descent algorithm on this linear network. From Eq. (10.16), we know that the gradient of the performance index is

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d} . \qquad (13.25)$$

Therefore, the steepest descent algorithm (see Eq. (9.10)) will be

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\mathbf{g}_k = \mathbf{x}_k - \alpha(\mathbf{A}\mathbf{x}_k + \mathbf{d}) . \tag{13.26}$$

We want to know how close we come to the minimum of the squared error at each iteration. For quadratic performance indices, we know that the minimum will occur at the following point (see Eq. (8.62)):

$$\mathbf{x}^{ML} = -\mathbf{A}^{-1}\mathbf{d} , \tag{13.27}$$

where the superscript ML indicates that this result maximizes the likelihood function, in addition to minimizing the squared error, as we saw in Eq. (13.11).

We can now rewrite Eq. (13.26) as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\mathbf{A}(\mathbf{x}_k + \mathbf{A}^{-1}\mathbf{d}) = \mathbf{x}_k - \alpha\mathbf{A}(\mathbf{x}_k - \mathbf{x}^{ML}) . \tag{13.28}$$

With some additional algebra we can find

$$\mathbf{x}_{k+1} = [\mathbf{I} - \alpha\mathbf{A}]\mathbf{x}_k + \alpha\mathbf{A}\mathbf{x}^{ML} = \mathbf{M}\mathbf{x}_k + [\mathbf{I} - \mathbf{M}]\mathbf{x}^{ML} , \tag{13.29}$$

where $\mathbf{M} = [\mathbf{I} - \alpha\mathbf{A}]$. The next step is to relate $\mathbf{x}_{k+1}$ to the initial guess $\mathbf{x}_0$. Starting at the first iteration, using Eq. (13.29), we have

$$\mathbf{x}_1 = \mathbf{M}\mathbf{x}_0 + [\mathbf{I} - \mathbf{M}]\mathbf{x}^{ML} , \tag{13.30}$$

where the initial guess $\mathbf{x}_0$ usually consists of random values near zero. Continuing to the second iteration:

$$\begin{aligned}\mathbf{x}_2 &= \mathbf{M}\mathbf{x}_1 + [\mathbf{I} - \mathbf{M}]\mathbf{x}^{ML} \\ &= \mathbf{M}^2\mathbf{x}_0 + \mathbf{M}[\mathbf{I} - \mathbf{M}]\mathbf{x}^{ML} + [\mathbf{I} - \mathbf{M}]\mathbf{x}^{ML} \\ &= \mathbf{M}^2\mathbf{x}_0 + \mathbf{M}\mathbf{x}^{ML} - \mathbf{M}^2\mathbf{x}^{\mathrm{ML}} + \mathbf{x}^{\mathrm{ML}} - \mathbf{M}\mathbf{x}^{ML} \\ &= \mathbf{M}^2\mathbf{x}_0 + \mathbf{x}^{ML} - \mathbf{M}^2\mathbf{x}^{ML} = \mathbf{M}^2\mathbf{x}_0 + [\mathbf{I} - \mathbf{M}^2]\mathbf{x}^{ML} .\end{aligned} \tag{13.31}$$

Following similar steps, at the k^{th} iteration we have

$$\mathbf{x}_k = \mathbf{M}^k\mathbf{x}_0 + [\mathbf{I} - \mathbf{M}^k]\mathbf{x}^{ML} , \tag{13.32}$$

This key result shows how far we progress from the initial guess to the maximum likelihood weights in k iterations. We will use this result later to compare with regularization.

Regularization Analysis

Recall from Eq. (13.4) that the regularized performance index adds a penalty term to the sum squared error, as in

$$F(\mathbf{x}) = \beta E_D + \alpha E_W . \tag{13.33}$$

For the following analysis, it will more convenient to consider the following equivalent (because the minimum occurs at the same place) performance index

$$F^*(\mathbf{x}) = \frac{F(\mathbf{x})}{\beta} = E_D + \frac{\alpha}{\beta} E_W = E_D + \rho E_W , \tag{13.34}$$

which has only one regularization parameter.

The sum squared weight penalty term E_W can be written

$$E_W = (\mathbf{x} - \mathbf{x}_0)^T (\mathbf{x} - \mathbf{x}_0) , \tag{13.35}$$

where the nominal value $\mathbf{x}_0$ is normally taken to be the zero vector.

In order to locate the minimum of the regularized performance index, which is also the most probable value $\mathbf{x}^{MP}$, we will set the gradient equal to zero:

$$\nabla F^*(\mathbf{x}) = \nabla E_D + \rho \nabla E_W = \mathbf{0} . \tag{13.36}$$

The gradient of the penalty term, Eq. (13.35), is

$$\nabla E_W = 2(\mathbf{x} - \mathbf{x}_0) . \tag{13.37}$$

From Eq. (13.25) and Eq. (13.28), the gradient of the sum squared error is

$$\nabla E_D = \mathbf{A}\mathbf{x} + \mathbf{d} = \mathbf{A}(\mathbf{x} + \mathbf{A}^{-1}\mathbf{d}) = \mathbf{A}(\mathbf{x} - \mathbf{x}^{ML}) . \tag{13.38}$$

We can now set the total gradient to zero:

$$\nabla F^*(\mathbf{x}) = \mathbf{A}(\mathbf{x} - \mathbf{x}^{ML}) + 2\rho(\mathbf{x} - \mathbf{x}_0) = \mathbf{0} . \tag{13.39}$$

The solution of Eq. (13.39) is the most probable value for the weights, $\mathbf{x}^{MP}$. We can make that substitution and perform some algebra to obtain

$$\begin{aligned} \mathbf{A}(\mathbf{x}^{MP} - \mathbf{x}^{ML}) &= -2\rho(\mathbf{x}^{MP} - \mathbf{x}_0) = -2\rho(\mathbf{x}^{MP} - \mathbf{x}^{ML} + \mathbf{x}^{ML} - \mathbf{x}_0) \\ &= -2\rho(\mathbf{x}^{MP} - \mathbf{x}^{ML}) - 2\rho(\mathbf{x}^{ML} - \mathbf{x}_0) \end{aligned} \tag{13.40}$$

Now combine the terms multiplying $(\mathbf{x}^{MP} - \mathbf{x}^{ML})$:

$$(\mathbf{A} + 2\rho\mathbf{I})(\mathbf{x}^{MP} - \mathbf{x}^{ML}) = 2\rho(\mathbf{x}_0 - \mathbf{x}^{ML}) . \qquad (13.41)$$

Solving for $(\mathbf{x}^{MP} - \mathbf{x}^{ML})$, we find

$$(\mathbf{x}^{MP} - \mathbf{x}^{ML}) = 2\rho(\mathbf{A} + 2\rho\mathbf{I})^{-1}(\mathbf{x}_0 - \mathbf{x}^{ML}) = \mathbf{M}_\rho(\mathbf{x}_0 - \mathbf{x}^{ML}) , \qquad (13.42)$$

where $\mathbf{M}_\rho = 2\rho(\mathbf{A} + 2\rho\mathbf{I})^{-1}$.

We want to know the relationship between the regularized solution $\mathbf{x}^{MP}$ and the minimum of the squared error $\mathbf{x}^{ML}$, so we can solve Eq. (13.42) for $\mathbf{x}^{MP}$:

$$\mathbf{x}^{MP} = \mathbf{M}_\rho\mathbf{x}_0 + [\mathbf{I} - \mathbf{M}_\rho]\mathbf{x}^{ML} . \qquad (13.43)$$

This is the key result that describes the relationship between the regularized solution and the minimum of the squared error. By comparing Eq. (13.43) with Eq. (13.32), we can investigate the relationship between early stopping and regularization. We will do that in the next section.

Connection Between Early Stopping and Regularization

To compare early stopping and regularization, we need to compare Eq. (13.43) and Eq. (13.32). They are summarized in Figure 13.9. We would like to find out when these two solutions are equal. In other words, when do early stopping and regularization produce the same weights?

Early Stopping	*Regularization*
$\mathbf{x}_k = \mathbf{M}^k\mathbf{x}_0 + [\mathbf{I} - \mathbf{M}^k]\mathbf{x}^{ML}$	$\mathbf{x}^{MP} = \mathbf{M}_\rho\mathbf{x}_0 + [\mathbf{I} - \mathbf{M}_\rho]\mathbf{x}^{ML}$
$\mathbf{M} = [\mathbf{I} - \alpha\mathbf{A}]$	$\mathbf{M}_\rho = 2\rho(\mathbf{A} + 2\rho\mathbf{I})^{-1}$

Figure 13.9 Early Stopping and Regularization Solutions

The key matrix for early stopping is $\mathbf{M}^k = [\mathbf{I} - \alpha\mathbf{A}]^k$. The key matrix for regularization is $\mathbf{M}_\rho = 2\rho(\mathbf{A} + 2\rho\mathbf{I})^{-1}$. If these two matrices are equal, then the weights for early stopping will be the same as the weights for regularization. In Eq. (9.22) we showed that the eigenvectors of $\mathbf{M}$ are the same as the eigenvectors of $\mathbf{A}$ and that the eigenvalues of $\mathbf{M}$ are $(1 - \alpha\lambda_i)$, where the eigenvalues of $\mathbf{A}$ are λ_i. The eigenvalues of $\mathbf{M}^k$ are then

$$eig(\mathbf{M}^k) = (1 - \alpha\lambda_i)^k . \qquad (13.44)$$

Now let's consider the matrix $\mathbf{M}_\rho$. First, using the same procedures that led to Eq. (9.22), we can show that the eigenvectors of $(\mathbf{A} + 2\rho\mathbf{I})$ are the same as the eigenvectors of $\mathbf{A}$, and the eigenvalues of $(\mathbf{A} + 2\rho\mathbf{I})$ are

$(2\rho + \lambda_i)$. Also, the eigenvectors of the inverse of a matrix are the same as the eigenvectors of the original matrix, and the eigenvalues of the inverse are the reciprocals of the original eigenvalues. Therefore, the eigenvectors of $\mathbf{M}_\rho$ are the same as the eigenvectors of $\mathbf{A}$, and the eigenvalues of $\mathbf{M}_\rho$ are

$$eig(\mathbf{M}_\rho) = \frac{2\rho}{(\lambda_i + 2\rho)}. \tag{13.45}$$

Therefore, in order for $\mathbf{M}^k$ to equal $\mathbf{M}_\rho$, they just need to have equal eigenvalues:

$$\frac{2\rho}{(\lambda_i + 2\rho)} = (1 - \alpha\lambda_i)^k. \tag{13.46}$$

Take the logarithm of both sides:

$$-\log\left(1 + \frac{\lambda_i}{2\rho}\right) = k\log(1 - \alpha\lambda_i). \tag{13.47}$$

These expressions are equal at $\lambda_i = 0$, so they will always be equal if their derivatives are equal. Taking derivatives of both sides, we have

$$-\frac{1}{\left(1 + \frac{\lambda_i}{2\rho}\right)}\frac{1}{\rho} = \frac{k}{1 - \alpha\lambda_i}(-\alpha), \tag{13.48}$$

or

$$\alpha k = \frac{1}{2\rho}\,\frac{(1 - \alpha\lambda_i)}{(1 + \lambda_i/(2\rho))}. \tag{13.49}$$

If $\alpha\lambda_i$ is small (slow, stable learning) and $\lambda_i/(2\rho)$ is small, then we have the approximate result

$$\alpha k \cong \frac{1}{2\rho}. \tag{13.50}$$

Therefore, early stopping is approximately equivalent to regularization. Increasing the number of iterations k is approximately the same as decreasing the regularization parameter ρ.

Example, Interpretation of Effective Number of Parameters

We will illustrate this result with a simple example. Suppose that we have a single layer, linear network with no bias. The input/target pairs are given by

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\1\end{bmatrix}, t_1 = 1\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix}-1\\1\end{bmatrix}, t_2 = -1\right\},$$

where the probability of the first pair is 0.75, and the probability of the second pair is 0.25. Following Eq. (10.13) and Eq. (10.15), we can find the quadratic mean square error performance index as

$$c = E[t^2] = (1)^2(0.75) + (-1)^2(0.25) = 1,$$

$$\mathbf{h} = E[t\mathbf{z}] = (0.75)(1)\begin{bmatrix}1\\1\end{bmatrix} + (0.25)(-1)\begin{bmatrix}-1\\1\end{bmatrix} = \begin{bmatrix}1\\0.5\end{bmatrix},$$

$$\mathbf{d} = -2\mathbf{h} = (-2)\begin{bmatrix}1\\0.5\end{bmatrix} = \begin{bmatrix}-2\\-1\end{bmatrix},$$

$$\mathbf{A} = 2\mathbf{R} = 2(E[\mathbf{z}\mathbf{z}^T]) = 2\left((0.75)\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1 & 1\end{bmatrix} + 0.25\begin{bmatrix}-1\\1\end{bmatrix}\begin{bmatrix}-1 & 1\end{bmatrix}\right) = \begin{bmatrix}2 & 1\\1 & 2\end{bmatrix},$$

$$E_D = c + \mathbf{x}^T\mathbf{d} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}.$$

The minimum of the mean squared error occurs at

$$\mathbf{x}^{ML} = -\mathbf{A}^{-1}\mathbf{d} = \mathbf{R}^{-1}\mathbf{h} = \begin{bmatrix}1 & 0.5\\0.5 & 1\end{bmatrix}^{-1}\begin{bmatrix}1\\0.5\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix}.$$

Now let's investigate the eigensystem of the Hessian matrix of E_D:

$$\nabla^2 E_D(\mathbf{x}) = \mathbf{A} = 2\mathbf{R} = \begin{bmatrix}2 & 1\\1 & 2\end{bmatrix}.$$

To find the eigenvalues:

$$\left|\mathbf{A} - \lambda\mathbf{I}\right| = \begin{vmatrix}2-\lambda & 1\\1 & 2-\lambda\end{vmatrix} = \lambda^2 - 4\lambda + 3 = (\lambda-1)(\lambda-3),$$

$$\lambda_1 = 1, \qquad \lambda_2 = 3.$$

To find the eigenvectors:

$$\left[\mathbf{A} - \lambda\mathbf{I}\right]\mathbf{v} = 0 .$$

For $\lambda_1 = 1$,

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\mathbf{v}_1 = 0 \qquad \mathbf{v}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} ,$$

and for $\lambda_2 = 3$,

$$\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}\mathbf{v}_2 = 0 \qquad \mathbf{v}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} .$$

The contour plot for E_D is shown in Figure 13.10

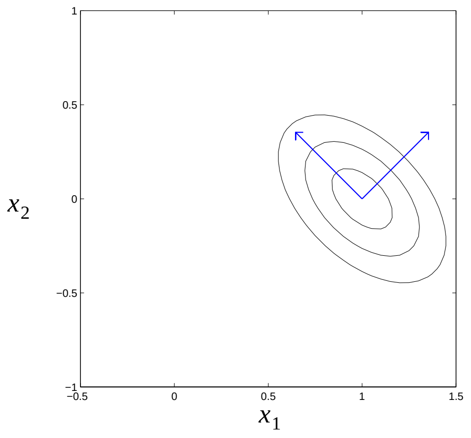

Figure 13.10 Contour Plot for E_D

Now consider the regularized performance index of Eq. (13.34). Its Hessian matrix will be

$$\nabla^2 F^*(\mathbf{x}) = \nabla^2 E_D + \rho\nabla^2 E_W = \nabla^2 E_D + 2\rho\mathbf{I} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} + \rho\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 2+2\rho & 1 \\ 1 & 2+2\rho \end{bmatrix} .$$

In Figure 13.11 we have contour plots for F as ρ is equal to 0, 1 and ∞.

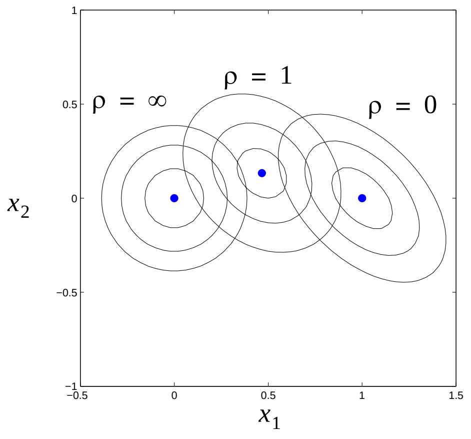

Figure 13.11 Contour Plot for F

In Figure 13.12 the blue curve represents the movement of $\mathbf{x}^{MP}$ as ρ is varied.

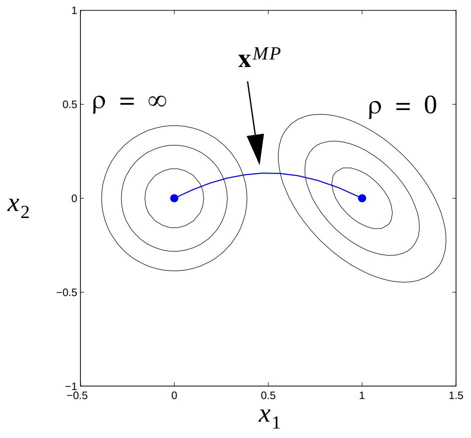

Figure 13.12 $\mathbf{x}^{MP}$ as ρ is Varied

Now let's compare this regularization result with early stopping. Figure 13.13 shows the steepest descent trajectory for minimizing E_D, starting from very small values for the weights. If we stop early, the result will fall along the blue curve. Notice that this curve is very close to the regularization curve in Figure 13.12. If the number of iterations is very small, this is equivalent to a very large value for ρ. As the number of iterations increases, it is equivalent to reducing ρ.

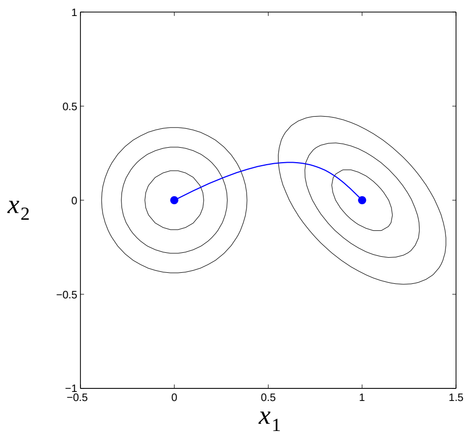

Figure 13.13 Steepest Descent Trajectory

To experiment with the relationship between Early Stopping and Regularization, use the MATLAB® Neural Network Design Demonstration Early Stopping/Regularization (**nnd17esr**).

It is useful to consider the relationship between the eigenvalues and eigenvectors of the Hessian matrix $\nabla^2 E_D(\mathbf{x})$ and the results of regularization and early stopping. In this example, λ_2 is larger than λ_1, so E_D has higher curvature in the $\mathbf{v}_2$ direction. This means that we will get a quicker reduction in the squared error if we move in that direction first. This is shown in Figure 13.13, as the initial steepest descent movement is almost in the direction of $\mathbf{v}_2$. Note also that in regularization, as shown in Figure 13.12, as ρ decreases from a large value, the weights move first in the $\mathbf{v}_2$ direction. For a given change in the weights, this direction provides the largest reduction in the squared error.

Since the eigenvalue λ_1 is smaller than λ_2, we only move in the $\mathbf{v}_1$ direction after achieving significant reduction in E_D in the $\mathbf{v}_2$ direction. This would be even more pronounced if the difference between λ_1 and λ_2 were greater. In the limiting case, where $\lambda_1 = 0$, we would not have to move in the $\mathbf{v}_1$ direction at all. We would only need to move in the $\mathbf{v}_2$ direction to get the complete reduction in the squared error. (This would be the case of the stationary valley, as in Figure 8.9.) Note that in this case we would only be effectively using one parameter, even though the network has two weights. (Of course, this one effective parameter is some combination of the two weights.) Therefore, the effective number of parameters is related to the number of eigenvalues of $\nabla^2 E_D(\mathbf{x})$ that are significantly different than zero. We will analyze this in detail in the next section.

Effective Number of Parameters

Recall the previous definition for the effective number of parameters:

$$\gamma = n - 2\alpha^{MP}\mathrm{tr}\{(\mathbf{H}^{MP})^{-1}\} \tag{13.51}$$

We can express this in terms of the eigenvalues of $\nabla^2 E_D(\mathbf{x})$. First, we can write the Hessian matrix as

$$\mathbf{H}(\mathbf{x}) = \nabla^2 F(\mathbf{x}) = \beta\nabla^2 E_D + \alpha\nabla^2 E_W = \beta\nabla^2 E_D + 2\alpha\mathbf{I}. \tag{13.52}$$

Using arguments similar to those leading to Eq. (13.44), we can show that the eigenvalues of $\mathbf{H}(\mathbf{x})$ are $(\beta\lambda_i + 2\alpha)$. We can then use two properties of eigenvalues to compute $\mathrm{tr}\{\mathbf{H}^{-1}\}$. First, the eigenvalues of $\mathbf{H}^{-1}$ are the reciprocals of the eigenvalues of $\mathbf{H}$, and, second, the trace of a matrix is equal to the sum of its eigenvalues. Using these two properties, we can write

$$\mathrm{tr}\{\mathbf{H}^{-1}\} = \sum_{i=1}^{n} \frac{1}{\beta\lambda_i + 2\alpha}. \tag{13.53}$$

We can now write the effective number of parameters as

$$\gamma = n - 2\alpha^{MP}\mathrm{tr}\{(\mathbf{H}^{MP})^{-1}\} = n - \sum_{i=1}^{n} \frac{2\alpha}{\beta\lambda_i + 2\alpha} = \sum_{i=1}^{n} \frac{\beta\lambda_i}{\beta\lambda_i + 2\alpha}, \tag{13.54}$$

or

$$\gamma = \sum_{i=1}^{n} \frac{\beta\lambda_i}{\beta\lambda_i + 2\alpha} = \sum_{i=1}^{n} \gamma_i, \tag{13.55}$$

where

$$\gamma_i = \frac{\beta\lambda_i}{\beta\lambda_i + 2\alpha}. \tag{13.56}$$

Note that $0 \le \gamma_i \le 1$, so the effective number of parameters γ must fall between zero and n. If all of the eigenvalues of $\nabla^2 E_D(\mathbf{x})$ are large, then the effective number of parameters will equal the total number of parameters. If some of the eigenvalues are very small, then the effective number of parameters will equal the number of large eigenvalues, as was also demonstrated by our example in the previous section.

Summary of Results

Problem Statement

A network trained to generalize will perform as well in new situations as it does on the data on which it was trained.

$$E_D = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q)$$

Methods for Improving Generalization

Estimating Generalization Error - The Test Set

Given a limited amount of available data, it is important to hold aside a certain subset during the training process. After the network has been trained, we will compute the errors that the trained network makes on this *test set*. The test set errors will then give us an indication of how the network will perform in the future; they are a measure of the generalization capability of the network.

Early Stopping

The available data (after removing the test set) is divided into two parts: a training set and a validation set. The training set is used to compute gradients or Jacobians and to determine the weight update at each iteration. When the error on the validation set goes up for several iterations, the training is stopped, and the weights that produced the minimum error on the validation set are used as the final trained network weights.

Regularization

$$F(\mathbf{x}) = \beta E_D + \alpha E_W = \beta \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q) + \alpha \sum_{i=1}^{n} x_i^2$$

Bayesian Regularization

Level I Bayesian Framework

$$P(\mathbf{x} | D, \alpha, \beta, M) = \frac{P(D|\mathbf{x}, \beta, M) P(\mathbf{x}|\alpha, M)}{P(D|\alpha, \beta, M)}$$

$$P(D|\mathbf{x}, \beta, M) = \frac{1}{Z_D(\beta)} \exp(-\beta E_D), \quad \beta = 1/(2\sigma_\varepsilon^2)$$

$$Z_D(\beta) = (2\pi\sigma_\varepsilon^2)^{N/2} = (\pi/\beta)^{N/2}$$

$$P(\mathbf{x} | \alpha, M) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W), \quad \alpha = 1/(2\sigma_w^2)$$

$$Z_W(\alpha) = (2\pi\sigma_w^2)^{n/2} = (\pi/\alpha)^{n/2}$$

$$P(\mathbf{x} | D, \alpha, \beta, M) = \frac{1}{Z_F(\alpha, \beta)} \exp(-F(\mathbf{x}))$$

Level II Bayesian Framework

$$P(\alpha, \beta | D, M) = \frac{P(D|\alpha, \beta, M)P(\alpha, \beta|M)}{P(D|M)}$$

$$\alpha^{MP} = \frac{\gamma}{2E_W(\mathbf{x}^{MP})} \text{ and } \beta^{MP} = \frac{N - \gamma}{2E_D(\mathbf{x}^{MP})}$$

$$\gamma = n - 2\alpha^{MP} \mathrm{tr}(\mathbf{H}^{MP})^{-1}$$

Bayesian Regularization Algorithm

0. Initialize α, β and the weights. The weights are initialized randomly, and then E_D and E_W are computed. Set $\gamma = n$, and compute α and β using Eq. (13.23).

1. Take one step of the Levenberg-Marquardt algorithm toward minimizing the objective function $F(\mathbf{x}) = \beta E_D + \alpha E_W$.

2. Compute the effective number of parameters $\gamma = N - 2\alpha \mathrm{tr}(\mathbf{H})^{-1}$, making use of the Gauss-Newton approximation to the Hessian available in the Levenberg-Marquardt training algorithm:
$\mathbf{H} = \nabla^2 F(\mathbf{x}) \approx 2\beta \mathbf{J}^T\mathbf{J} + 2\alpha \mathbf{I}_n$, where $\mathbf{J}$ is the Jacobian matrix of the training set errors (see Eq. (12.37)).

3. Compute new estimates for the regularization parameters
$\alpha = \dfrac{\gamma}{2E_W(\mathbf{x})}$ and $\beta = \dfrac{N - \gamma}{2E_D(\mathbf{x})}$.

4. Now iterate steps 1 through 3 until convergence.

Relationship Between Early Stopping and Regularization

Early Stopping	*Regularization*
$\mathbf{x}_k = \mathbf{M}^k \mathbf{x}_0 + [\mathbf{I} - \mathbf{M}^k]\mathbf{x}^{ML}$	$\mathbf{x}^{MP} = \mathbf{M}_\rho \mathbf{x}_0 + [\mathbf{I} - \mathbf{M}_\rho]\mathbf{x}^{ML}$
$\mathbf{M} = [\mathbf{I} - \alpha\mathbf{A}]$	$\mathbf{M}_\rho = 2\rho(\mathbf{A} + 2\rho\mathbf{I})^{-1}$

$$eig(\mathbf{M}^k) = (1 - \alpha\lambda_i)^k$$

$$eig(\mathbf{M}_\rho) = \frac{2\rho}{(\lambda_i + 2\rho)}$$

$$\alpha k \cong \frac{1}{2\rho}$$

Effective Number of Parameters

$$\gamma = \sum_{i=1}^{n} \frac{\beta\lambda_i}{\beta\lambda_i + 2\alpha}$$

$$0 \le \gamma \le n$$

Solved Problems

P13.1 In this problem and in the following one we want to investigate the relationship between maximum likelihood methods and Bayesian methods. Suppose that we have a random variable that is uniformly distributed between 0 and *x*. We take a series of *Q* independent samples of the random variable. Find the maximum likelihood estimate of *x*.

Before we begin this problem, let's review the Level I Bayesian formulation of Eq. (13.10). We will not need the Level II formulation for this simple problem, so we do not need the regularization parameters. Also, we only have a single parameter to estimate, so x is a scalar. Eq. (13.10) can then be simplified to

$$P(x|D) = \frac{P(D|x)P(x)}{P(D)}.$$

We are interested in the maximum likelihood estimate for this problem, so we need to find the value of x that maximizes the likelihood term $P(D|x)$. The data is the Q independent samples from the uniformly distributed random variable. A graph of the uniform density function is given in Figure P13.1.

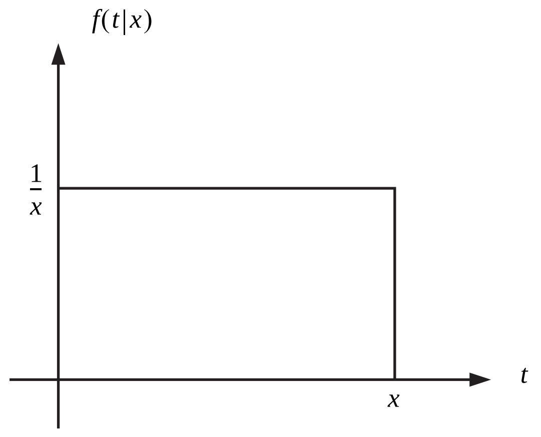

Figure P13.1 Uniform Density Function

The definition can be written

$$f(t|x) = \begin{cases} \frac{1}{x}, & 0 \le t \le x \\ 0, & \text{elsewhere} \end{cases} .$$

If we have Q independent samples of the random variable, then we can multiply each of the individual probabilities to get the joint probability of all samples:

$$P(D|x) = \prod_{i=1}^{Q} f(t_i|x) = \begin{cases} \frac{1}{x^Q}, & 0 \le t_i \le x, \text{ for all } i \\ 0, & \text{elsewhere} \end{cases} = \begin{cases} \frac{1}{x^Q}, & x \ge max(t_i) \\ 0, & x < max(t_i) \end{cases}$$

The plot of the resulting likelihood function is shown in Figure P13.1.

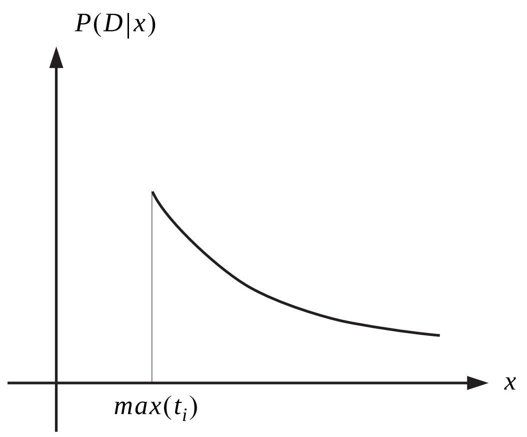

Figure P13.2 Likelihood Function for Solved Problem P13.1

From this plot, we can see that the value of x that maximizes the likelihood function is

$$x^{ML} = max(t_i) .$$

Therefore, the maximum likelihood estimate of x is the maximum value obtained from the Q independent samples of the random variable. This seems like a reasonable estimate of x, which is the upper limit of the random variable.

P13.2 In this problem we will compare the maximum likelihood and Bayesian estimators. Assume that we have a series of measurements of a random signal in noise:

$$t_i = x + \varepsilon_i .$$

Assume that the noise has a Gaussian density, with zero mean:

$$f(\varepsilon_i) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\varepsilon_i^2}{2\sigma^2}\right)$$

i. Find the maximum likelihood estimate of x.

Assume that x is a zero-mean random variable, with Gaussian prior density:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma_x}\exp\left(-\frac{x^2}{2\sigma_x^2}\right) = \frac{1}{Z_W(\alpha)}\exp(-\alpha E_W)$$

ii. Find the most probable estimate of x.

i. To find the maximum likelihood estimate, we need to find the likelihood function $P(D|x)$. This represents the density of the data, given x. The first step is to use the noise density to find the density of the measurement. Since, with x given, the density for the measurement would be the same as the density for the noise, but with a mean of x, we have

$$f(t_i|x) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(t_i - x)^2}{2\sigma^2}\right) .$$

Assuming that the measurement noises are independent, we can multiply the probability densities:

$$P(D|x) = f(t_1, t_2, \ldots, t_Q|x) = f(t_1|x)f(t_2|x)\ldots f(t_Q|x) = P(D|x)$$

$$= \frac{1}{(2\pi)^{Q/2}\sigma^Q}\exp\left(-\frac{\sum_{i=1}^{Q}(t_i - x)^2}{2\sigma^2}\right) = \frac{1}{Z(\beta)}\exp(-\beta E_D)$$

where

$$\beta = \frac{1}{2\sigma^2},\ E_D = \sum_{i=1}^{Q}(t_i - x)^2 = \sum_{i=1}^{Q} e_i^2,\ Z(\beta) = (\pi/\beta)^{Q/2} .$$

To maximize the likelihood, we should minimize E_D. Setting the derivative to zero, we find

$$\frac{dE_D}{dx} = \frac{d}{dx}\sum_{i=1}^{Q}(t_i - x)^2 = -2\sum_{i=1}^{Q}(t_i - x) = -2\left[\left(\sum_{i=1}^{Q} t_i\right) - Qx\right] = 0 .$$

Solving for x, we find the maximum likelihood estimate:

$$x^{ML} = \frac{1}{Q}\sum_{i=1}^{Q} t_i$$

ii. To find the most probable estimate, we need to use Bayes' rule (Eq. (13.10)) to find the posterior density:

$$P(x|D) = \frac{P(D|x)P(x)}{P(D)} .$$

The likelihood function $P(D|x)$ was found above to be

$$P(D|x) = \frac{1}{Z(\beta)}\exp(-\beta E_D)$$

The prior density is

$$P(x) = f(x) = \frac{1}{\sqrt{2\pi}\sigma_x}\exp\left(-\frac{x^2}{2\sigma_x^{\ 2}}\right) = \frac{1}{Z_W(\alpha)}\exp(-\alpha E_W) ,$$

where

$$\alpha = \frac{1}{2\sigma_x^{\ 2}},\ Z_W(\alpha) = (\pi/\alpha)^{1/2},\ E_W = x^2 .$$

The posterior density can then be computed as

$$\begin{aligned} P(x|D) &= f(x|t_1, t_2, \ldots, t_Q) \\ &= \frac{f(t_1, t_2, \ldots, t_Q|x)f(x)}{f(t_1, t_2, \ldots, t_Q)} \\ &= \frac{\frac{1}{Z_D(\beta)}\frac{1}{Z_W(\alpha)}\exp(-(\beta E_D + \alpha E_W))}{\text{Normalization Factor}} \end{aligned}$$

To find the most probable value for x, we maximize the posterior density. This is equivalent to minimizing

$$\beta E_D + \alpha E_W = \beta \sum_{i=1}^{Q} (t_i - x)^2 + \alpha x^2 .$$

To find the minimum, we take the derivative with respect to x and set it equal to zero:

$$\frac{d}{dx}(\beta E_D + \alpha E_W) = \frac{d}{dx}\left(\beta \sum_{i=1}^{Q} (t_i - x)^2 + \alpha x^2\right) = -2\beta \sum_{i=1}^{Q} (t_i - x) + 2\alpha x$$

$$= -2\beta\left[\left(\sum_{i=1}^{Q} t_i\right) - Qx\right] + 2\alpha x$$

$$= -2\left[\beta\left(\sum_{i=1}^{Q} t_i\right) - (\alpha + Q\beta)x\right] = 0$$

Solving for μ, we obtain

$$x^{MP} = \frac{\beta\left(\sum_{i=1}^{Q} t_i\right)}{\alpha + Q\beta}$$

Notice that as α goes to zero (variance σ_x^2 goes to infinity), x^{MP} approaches x^{ML}. Increasing the variance of the prior density represents uncertainty in our prior knowledge about x. With large prior uncertainty, we rely on the data for our estimate of x.

Figure P13.3 illustrates $P(D|x)$, $P(x)$ and $P(x|D)$ for the case where $\sigma_x^2 = 2$, $\sigma^2 = 1$, $Q = 1$ and $t_1 = 1$. Here the variance associated with the measurement is smaller than the variance associated with our prior density for x, so x^{MP} is closer to $x^{ML} = t_1 = 1$ than it is to the maximum of the prior density, which occurs at 0.

To experiment with this signal in noise example, use the MATLAB® Neural Network Design Demonstration Signal Plus Noise (**nnd17spn**).

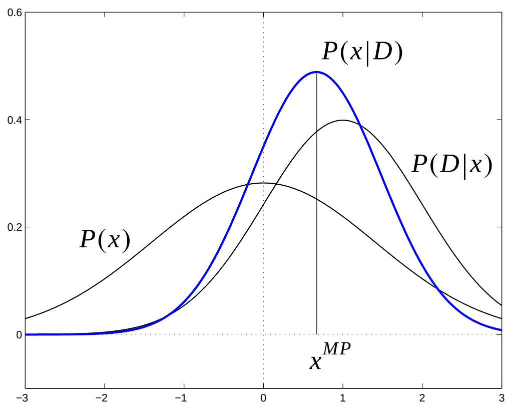

Figure P13.3 Prior and Posterior Density Functions

P13.3 Derive Eq. (13.23).

To solve for α^{MP} and β^{MP}, we will take the derivatives of the log of $P(D|\alpha, \beta, M)$, given in Eq. (13.17), with respect to α and β, and set the derivatives to zero. Taking the log of Eq. (13.17), and substituting Eq. (13.12), Eq. (13.14) and Eq. (13.22), we obtain

$$\log P(D|\alpha, \beta, M) = \log(Z_F) - \log(Z_D(\beta)) - \log(Z_W(\alpha))$$

$$= \frac{n}{2}\log(2\pi) - \frac{1}{2}\log\det(\mathbf{H}^{MP}) - F(\mathbf{x}^{MP}) - \frac{N}{2}\log\left(\frac{\pi}{\beta}\right) - \frac{n}{2}\log\left(\frac{\pi}{\alpha}\right)$$

$$= -F(\mathbf{x}^{MP}) - \frac{1}{2}\log\det(\mathbf{H}^{MP}) + \frac{N}{2}\log(\beta) + \frac{n}{2}\log(\alpha) + \frac{n}{2}\log 2 - \frac{N}{2}\log\pi$$

We will consider first the second term in this expression. Since $\mathbf{H}$ is the Hessian of F in Eq. (13.4), we can write it as $\mathbf{H} = \nabla^2 F = \nabla^2(\beta E_D) + \nabla^2(\alpha E_W) = \beta\mathbf{B} + 2\alpha\mathbf{I}$, where $\mathbf{B} = \nabla^2 E_D$. If we let λ^h be an eigenvalue of $\mathbf{H}$ and λ^b be an eigenvalue of $\beta\mathbf{B}$, then $\lambda^h = \lambda^b + 2\alpha$ for all corresponding eigenvalues. Now we take the derivative of the second term in the above equation with respect to α. Since the determinant of a matrix can be expressed as the product of its eigenvalues, we can reduce it as shown below, where $\mathrm{tr}(\mathbf{H}^{-1})$ is the trace of the inverse of the Hessian $\mathbf{H}$.

$$\begin{aligned}
\frac{\partial}{\partial\alpha}\frac{1}{2}\log\det\mathbf{H} &= \frac{1}{2\det\mathbf{H}}\frac{\partial}{\partial\alpha}\left(\prod_{k=1}^{n}\lambda^{h}\right) \\
&= \frac{1}{2\det\mathbf{H}}\frac{\partial}{\partial\alpha}\left(\prod_{i=1}^{n}(\lambda_i^b+2\alpha)\right) \\
&= \frac{1}{2\det\mathbf{H}}\left[\sum_{i=1}^{n}\left(\prod_{j\neq i}(\lambda_j^b+2\alpha)\right)\frac{\partial}{\partial\alpha}(\lambda_i^b+2\alpha)\right] \\
&= \frac{\sum_{i=1}^{n}\left(\prod_{j\neq i}(\lambda_j^b+2\alpha)\right)}{\prod_{i=1}^{n}(\lambda_i^b+2\alpha)} \\
&= \sum_{i=1}^{n}\frac{1}{\lambda_i^b+2\alpha} = \operatorname{tr}(\mathbf{H}^{-1})
\end{aligned}$$

Next, we will take the derivative of the same term with respect to β. First, define the parameter γ, as shown below, and expand it for use in our next step. The parameter γ is referred to as the effective number of parameters.

$$\begin{aligned}
&\gamma \equiv n - 2\alpha\operatorname{tr}(\mathbf{H}^{-1}) \\
&= n - 2\alpha\sum_{i=1}^{N}\frac{1}{\lambda_i^b+2\alpha} = \sum_{i=1}^{n}\left(1-\frac{2\alpha}{\lambda_i^b+2\alpha}\right) = \sum_{i=1}^{n}\left(\frac{\lambda_i^b}{\lambda_i^b+2\alpha}\right) = \sum_{i=1}^{n}\frac{\lambda_i^b}{\lambda_i^h}
\end{aligned}$$

Now take the derivative of $\frac{1}{2}\log\det(\mathbf{H}^{MP})$ with respect to β.

$$\begin{aligned}
\frac{\partial}{\partial\beta}\frac{1}{2}\log\det\mathbf{H} &= \frac{1}{2\det\mathbf{H}}\frac{\partial}{\partial\beta}\left(\prod_{k=1}^{n}\lambda_k^h\right)\\
&= \frac{1}{2\det\mathbf{H}}\frac{\partial}{\partial\beta}\left(\prod_{i=1}^{n}(\lambda_i^b+2\alpha)\right)\\
&= \frac{1}{2\det\mathbf{H}}\left[\sum_{i=1}^{n}\left(\prod_{j\neq i}(\lambda_j^b+2\alpha)\right)\frac{\partial}{\partial\beta}(\lambda_i^b+2\alpha)\right]\\
&= \frac{1}{2}\frac{\sum_{i=1}^{n}\left(\prod_{j\neq i}(\lambda_j^b+2\alpha)\left(\frac{\lambda_i^b}{\beta}\right)\right)}{\prod_{i=1}^{n}(\lambda_i^b+2\alpha)}\\
&= \frac{1}{2\beta}\sum_{i=1}^{n}\frac{\lambda_i^b}{\lambda_i^b+2\alpha} = \frac{\gamma}{2\beta}
\end{aligned}$$

where the fourth step is derived from the fact that λ_i^b is an eigenvalue of $\beta\mathbf{B}$, and therefore the derivative of λ_i^b with respect to β is just the eigenvalue of $\mathbf{B}$ which is λ_i^b/β.

Now we are finally ready to take the derivatives of all terms in $\log P(D|\alpha,\beta,M)$ and set them equal to zero. The derivative with respect to α will be

$$\begin{aligned}
\frac{\partial}{\partial\alpha}\log P(D|\alpha,\beta,M) &= -\frac{\partial}{\partial\alpha}F(\mathbf{w}^{\mathrm{MP}}) - \frac{\partial}{\partial\alpha}\frac{1}{2}\log\det(\mathbf{H}^{\mathrm{MP}}) + \frac{\partial}{\partial\alpha}\frac{n}{2}\log\alpha\\
&= -\frac{\partial}{\partial\alpha}(\alpha E_W(\mathbf{w}^{\mathrm{MP}})) - \mathrm{tr}(\mathbf{H}^{\mathrm{MP}})^{-1} + \frac{n}{2\alpha^{\mathrm{MP}}}\\
&= -E_W(\mathbf{w}^{\mathrm{MP}}) - \mathrm{tr}(\mathbf{H}^{\mathrm{MP}})^{-1} + \frac{n}{2\alpha^{\mathrm{MP}}} = 0
\end{aligned}$$

Rearranging terms, and using our definition of γ, we have

$$\begin{aligned}
E_W(\mathbf{w}^{\mathrm{MP}}) &= \frac{n}{2\alpha^{\mathrm{MP}}} - \mathrm{tr}(\mathbf{H}^{\mathrm{MP}})^{-1}\\
2\alpha^{\mathrm{MP}}E_W(\mathbf{w}^{\mathrm{MP}}) &= n - 2\alpha^{\mathrm{MP}}\mathrm{tr}(\mathbf{H}^{\mathrm{MP}})^{-1} = \gamma\\
\alpha^{\mathrm{MP}} &= \frac{\gamma}{2E_W(\mathbf{w}^{\mathrm{MP}})}
\end{aligned}$$

We now repeat the process for β.

$$\begin{aligned}\frac{\partial}{\partial\beta}\log P(D|\alpha,\beta,M) &= -\frac{\partial}{\partial\beta}F(\mathbf{w}^{\mathrm{MP}}) - \frac{\partial}{\partial\beta}\frac{1}{2}\log\det(\mathbf{H}^{\mathrm{MP}}) + \frac{\partial}{\partial\beta}\frac{N}{2}\log\beta \\ &= -\frac{\partial}{\partial\beta}(\beta E_D(\mathbf{w}^{\mathrm{MP}})) - \frac{\gamma}{2\beta^{\mathrm{MP}}} + \frac{N}{2\beta^{\mathrm{MP}}} \\ &= -E_D(\mathbf{w}^{\mathrm{MP}}) - \frac{\gamma}{2\beta^{\mathrm{MP}}} + \frac{N}{2\beta^{\mathrm{MP}}} = 0\end{aligned}$$

Rearranging terms,

$$\begin{aligned}E_D(\mathbf{w}^{\mathrm{MP}}) &= \frac{N}{2\beta^{\mathrm{MP}}} - \frac{\gamma}{2\beta^{\mathrm{MP}}} \\ \beta^{\mathrm{MP}} &= \frac{N-\gamma}{2E_D(\mathbf{w}^{\mathrm{MP}})}\end{aligned}$$

P13.4 Demonstrate that extrapolation can occur in a region that is surrounded by training data.

Consider the function displayed in Figure 13.3. In that example, extrapolation occurred in the upper left region of the input space, because all of the training data was in the lower right. Let's provide training data around the outside of the input space, but without data in the region

$$-1.5 < p_1 < 1.5 \qquad -1.5 < p_2 < 1.5 .$$

The training data is distributed as shown in Figure P13.4.

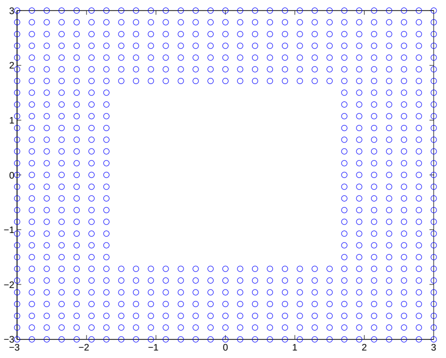

Figure P13.4 Training Data Locations

The result of the training is shown in Figure P13.5. The neural network approximation significantly overestimates the true function in the region without training data, even though surrounded by regions with training data. In addition, this result is random. With a different set of initial random weights, the network might underestimate the true function in this region. Extrapolation occurs because there is a significantly large region without training data. When the input space is of high dimension, it can be very difficult to tell when a network is extrapolating. It cannot be done by simply checking the individual ranges of each input variable.

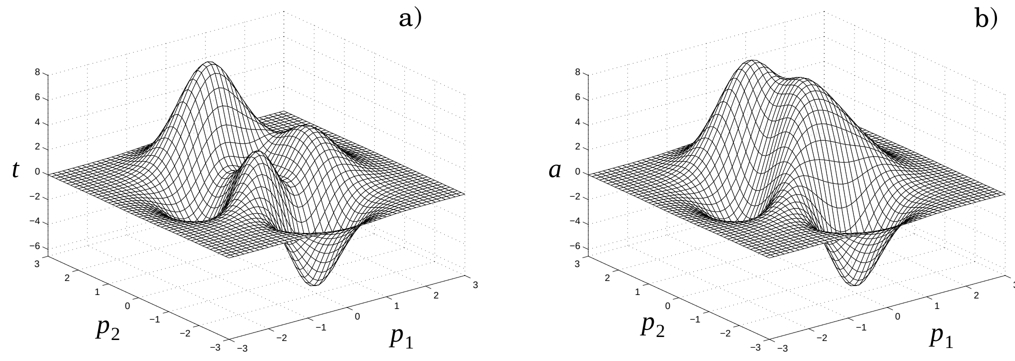

Figure P13.5 Function (a) and Neural Network Approximation (b)

P13.5 Consider the example starting on page 13-23. Find the effective number of parameters if $\rho = 1$.

To find the effective number of parameters, we can use Eq. (13.55):

$$\gamma = \sum_{i=1}^{n} \frac{\beta\lambda_i}{\beta\lambda_i + 2\alpha}.$$

We earlier found the eigenvalues to be $\lambda_1 = 1$, $\lambda_2 = 3$. The regularization parameter is

$$\rho = \frac{\alpha}{\beta} = 1.$$

We can rewrite γ in terms of ρ as follows

$$\gamma = \sum_{i=1}^{n} \frac{\beta\lambda_i}{\beta\lambda_i + 2\alpha} = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda_i + 2\frac{\alpha}{\beta}} = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda_i + 2\rho}.$$

Substituting our numbers, we find

$$\gamma = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda_i + 2\rho} = \frac{1}{1+2} + \frac{3}{3+2} = \frac{1}{3} + \frac{3}{5} = \frac{14}{15}.$$

Therefore, we are using approximately one of the two available parameters. The network has two parameters: $w_{1,1}$ and $w_{1,2}$. The parameter we are using is not one of these two, but rather a combination. As we can see from Figure 13.11, we move in the direction of the second eigenvector:

$$\mathbf{v}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

which means that we are changing $w_{1,1}$ and $w_{1,2}$ by the same amount. Although there are two parameters, we are effectively using only one. Since $\mathbf{v}_2$ is the eigenvector with the largest eigenvalue, we move in that direction to obtain the greatest reduction in the squared error.

P13.6 Demonstrate overfitting with polynomials. Consider fitting a polynomial

$$g_k(p) = x_0 + x_1 p + x_2 p^2 + \ldots + x_k p^k$$

to a set of data $\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}$ **so as to minimize the following squared error performance function.**

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (t_q - g_k(p_q))^2$$

First, we want to express the problem in matrix form. Define the following vectors.

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_Q \end{bmatrix} \quad \mathbf{G} = \begin{bmatrix} 1 & p_1 & \ldots & p_1^k \\ 1 & p_2 & \ldots & p_2^k \\ \vdots & \vdots & & \vdots \\ 1 & p_Q & \ldots & p_Q^k \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_k \end{bmatrix}$$

We can then write the performance index as follows.

$$F(\mathbf{x}) = [\mathbf{t} - \mathbf{G}\mathbf{x}]^T[\mathbf{t} - \mathbf{G}\mathbf{x}] = \mathbf{t}^T\mathbf{t} - 2\mathbf{x}^T\mathbf{G}^T t + \mathbf{x}^T\mathbf{G}^T\mathbf{G}\mathbf{x}$$

To locate the minimum, we take the gradient and set it equal to zero.

$$\nabla F(\mathbf{x}) = -2\mathbf{G}^T t + 2\mathbf{G}^T\mathbf{G}\mathbf{x} = 0$$

Solving for the weights, we obtain the least squares solution (maximum likelihood for the Gaussian noise case).

$$[\mathbf{G}^T\mathbf{G}]\mathbf{x}^{ML} = \mathbf{G}^T t \qquad \Rightarrow \qquad \mathbf{x}^{ML} = [\mathbf{G}^T\mathbf{G}]^{-1}\mathbf{G}^T t$$

To demonstrate the operation of the polynomial fitting, we will use the simple linear function $t = p$. To create the data set, we will sample the function at five different points and will add noise as follows

$$t_i = p_i + \varepsilon_i\, ,\ p = \{-1, -0.5, 0, 0.5, 1\}\, ,$$

where ε_i has a uniform density with range $[-0.25, 0.25]$. The code below shows how to generate the data and fit a 4th order polynomial. The results of fitting 2nd and 4th order polynomials are shown in Figure P13.6. The 4th order polynomial has five parameters, which allow it to exactly fit the five noisy data points, but it doesn't produce an accurate approximation of the true function.

```
» 2 + 2
ans =
      4
```

```
p = -1:.5:1;
t = p + 0.5*(rand(size(p))-0.5);
Q = length(p);
ord = 4;
G = ones(Q,1);
for i=1:ord,
    G = [G (p').^i];
end
x = (G'*G)\G'*t'; % Could also use x = G\t';
```

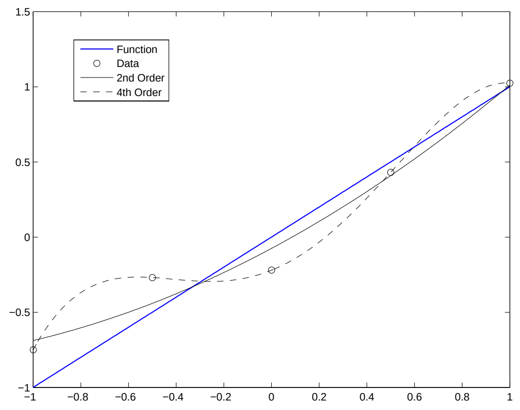

Figure P13.6 Polynomial Approximations to a Straight Line

Epilogue

The focus of this chapter has been the development of algorithms for training multilayer neural networks so that they generalize well. A network that generalizes well will perform as well in new situations as it performs on the data for which it was trained.

The basic approach to producing networks that generalize well is to find the simplest network that can represent the data. A simple network is one that has a small number of weights and biases.

The two methods that we presented in this chapter, early stopping and regularization, produce simple networks by constraining the weights, rather than by reducing the number of weights. We showed in this chapter that constraining the weights is equivalent to reducing the number of weights.

Further Reading

[AmMu97] S. Amari, N. Murata, K.-R. Muller, M. Finke, and H. H. Yang, “Asymptotic Statistical Theory of Overtraining and Cross-Validation,” *IEEE Transactions on Neural Networks*, vol. 8, no. 5, 1997.

When using early stopping, it is important to decide on the number of data points to place in the validation set. This paper provides a theoretical basis for the choice of validation set size.

[FoHa97] D. Foresee and M. Hagan, “Gauss-Newton Approximation to Bayesian Learning,” *Proceedings of the 1997 International Joint Conference on Neural Networks*, vol. 3, pp. 1930 - 1935, 1997.

This paper describes a method for implementing Bayesian regularization by using the Gauss-Newton approximation to the Hessian matrix.

[GoLa98] C. Goutte and J. Larsen, “Adaptive Regularization of Neural Networks Using Conjugate Gradient,” *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 2, pp. 1201-1204, 1998.

When using regularization, the important step is setting the regularization parameter. This paper describes a procedure for setting the regularization parameter to minimize the validation set error.

[MacK92] D. J. C. MacKay, “Bayesian Interpolation,” *Neural Computation*, vol. 4, pp. 415-447, 1992.

Bayesian approaches have been used for many years in statistics. This paper presents one of the first developments of a Bayesian framework for training neural networks. MacKay followed this paper with many others describing refinements of the approach.

[Sarle95] W. S. Sarle, “Stopped training and other remedies for overfitting,” In *Proceedings of the 27th Symposium on Interface*, 1995.

This is one of the early papers on the use of early stopping with a validation set to prevent overfitting. The paper describes simulation results comparing early stopping with other methods for improving generalization.

[SjLj94] J. Sjoberg and L. Ljung, “Overtraining, regularization and searching for minimum with application to neural networks,” Linkoping University, Sweden, Tech. Rep. LiTH-ISY-R-1567, 1994.

This report explains how early stopping and regularization are approximately equivalent processes. It demonstrates that the number of iterations of training is inversely proportional to the regularization parameter.

[Tikh63] A. N. Tikhonov, “The solution of ill-posed problems and the regularization method,” *Dokl. Acad. Nauk USSR*, vol. 151, no. 3, pp. 501-504, 1963.

Regularization is a method by which a squared error performance index is augmented by a penalty on the complexity of the approximating function. This is the original paper that introduced the concept of regularization. The penalty involved the derivatives of the approximating function.

[WaVe94] C. Wang, S. S. Venkatesh, and J. S. Judd, “Optimal Stopping and Effective Machine Complexity in Learning,” *Advances in Neural Information Processing Systems*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds., vol. 6, pp. 303-310, 1994.

This paper describes how the effective number of network parameters changes during the training process and how the generalization capability of the network can be improved by stopping the training early.

Exercises

E13.1 Consider fitting a polynomial (kth order)

$$g_k(p) = x_0 + x_1 p + x_2 p^2 + \ldots + x_k p^k$$

to a set of data $\{p_1, t_1\}$, $\{p_2, t_2\}$, … , $\{p_Q, t_Q\}$. It has been proposed that minimizing a performance index that penalizes the derivatives of the polynomial will provide improved generalization. Investigate the relationship between this technique and regularization using squared weights.

i. Derive the least squares solution for the weights x_i, which minimizes the following squared error performance index. (See Solved Problem P13.6.)

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (t_q - g_k(p_q))^2$$

ii. Derive the regularized least squares solution, with a squared weight penalty.

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (t_q - g_k(p_q))^2 + \rho \sum_{i=1}^{n} x_i^2$$

iii. Derive a solution for the weights that minimizes a sum of the squared error plus a sum of squared derivatives.

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (t_q - g_k(p_q))^2 + \rho \sum_{i=1}^{Q} \left[\frac{d}{dp} g_k(p_q)\right]^2$$

iv. Derive a solution for the weights that minimizes a sum of the squared error plus a sum of squared second derivatives.

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (t_q - g_k(p_q))^2 + \rho \sum_{i=1}^{Q} \left[\frac{d^2}{dp^2} g_k(p_q)\right]^2$$

E13.2 Write a MATLAB program to implement the solutions you found in E13.1 i. through iv. Using the following data points, adjust the ρ values to obtain the best results. Use $k = 8$ for all cases. Plot the data points, the noise-free function ($t = p$) and the polynomial approximation in each case. Compare the four approximations. Which do you think produces the best results? Which cases produce similar results?

$$t_i = p_i + \varepsilon_i \,,\ p = \{-1, -0.5, 0, 0.5, 1\} \,,$$

where ε_i has a uniform density with range $[-0.1, 0.1]$ (use the `randn` command in MATLAB).

E13.3 Investigate the extrapolation characteristics of neural networks and polynomials. Consider the problem described in E11.11, where a sine wave is fit over the range $-2 \le p \le 2$. Select 11 training points evenly spaced over this interval.

```
» 2 + 2
ans =
     4
```

i. After fitting the 1-2-1 neural network over this range, plot the actual sine function and the neural network approximation over the range $-4 \le p \le 4$.

ii. Fit a fifth-order polynomial (which has the same number of free parameters as the 1-2-1 network) to the sine wave over the range $-2 \le p \le 2$ (using your results from E13.1 i.). Plot the actual function and the polynomial approximation over the range $-4 \le p \le 4$.

iii. Discuss the extrapolation characteristics of the neural network and the polynomial.

E13.4 Suppose that we have a random variable t that is distributed according to the following density function. We take a series of Q independent samples of the random variable. Find the maximum likelihood estimate of x - x^{ML}.

$$f(t|x) = \frac{t}{x^2}\exp\left(-\frac{t}{x}\right) \qquad t \ge 0$$

E13.5 For the random variable given in E13.4, suppose that x is a random variable with the following prior density function. Find the most probable estimate of x - x^{MP}.

$$f(x) = \exp(-x) \qquad x \ge 0$$

E13.6 Repeat E13.5 for the following prior density function. Under what conditions will $x^{MP} = x^{ML}$?

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma_x}\exp\left(-\frac{(x-\mu_x)^2}{2\sigma_x^{\ 2}}\right)$$

E13.7 In the signal plus noise example given in Solved Problem P13.2, find x^{MP} for the following prior density functions.

i. $f(x) = \exp(-x) \qquad x \ge 0$

ii. $f(x) = \frac{1}{2}\exp(-|x-\mu|)$

E13.8 Suppose that the prior density in the level I Bayesian analysis (see page 13-12) has nonzero mean, μ_x. Find the new performance index.

E13.9 Repeat E11.11, but modify your program to use early stopping and to use 30 neurons. Select 10 training points and 5 validation points. Add noise to the validation and testing points that is uniformly distributed between -0.1 and 0.1 (using the MATLAB function `rand`). Measure the mean square error of the trained network on a testing set consisting of 20 equally-spaced points of the noise-free function. Try 10 different random sets of training and validation data. Compare the results with early-stopping with the results without early stopping.

E13.10 Repeat E13.9, but use regularization instead of early stopping. This will require modifying your program to compute the gradient of the regularized performance index. Add the standard gradient of the squared error, which is computed by the standard backpropagation algorithm, to the gradient of ρ times the squared weights. Try three different values of ρ. Compare these results with the early stopping results.

E13.11 Consider again the problem described in E10.4

i. Find the regularized performance index for $\rho = 0, 1, \infty$. Sketch the contour plot in each case. Indicate the location of the optimal weights in each case.

ii. Find the effective number of parameters for $\rho = 0, 1, \infty$.

iii. Starting with zero initial weights, approximately how many iterations of the steepest descent algorithm would need to be made on the mean square performance index to produce results that would be equivalent to minimizing the regularized performance index with $\rho = 1$? Assume a learning rate of $\alpha = 0.01$.

iv. Write a MATLAB M-file to implement the steepest descent algorithm to minimize the mean square error performance index that you found in part i. (This is a quadratic function.) Start the algorithm with zero initial conditions, and use a learning rate of $\alpha = 0.01$. Sketch the trajectory on a contour plot of the mean square error (the contour plot was found in E10.4). Verify that at the iteration you computed in part iii., the weights are close to the same values you found to minimize the regularized performance index with $\rho = 1$ in part i.