3.1.1Weight Decay, Weight Elimination, and Unit Elimination

It can be shown that in order to guarantee good generalization, the number of degrees of freedom or number of weights (which determines a network's complexity) must be considerably smaller than the amount of information available for training. Some insight into this matter can be gained from considering an analogous problem in curve fitting (Duda and Hart, 1973). For example, consider the rational function $g(x)=\left[(x-2)(2x+1)\right]/(1+x^2)$, which is plotted in Figure Error! No text of specified style in document.-**1** (solid line). And assume that we are given a set of 15 samples (shown as small circles) from which we are to find a "good" approximation to *g(x).* Two polynomial approximations are shown in Figure Error! No text of specified style in document.-**1**: an eleventh-order polynomial (dashed line) and an eighth-order polynomial (dotted line). These approximations are computed by minimizing the SSE criterion over the sample points. The higher-order polynomial has about the same number of parameters as the number of training samples and thus is shown to give a very close fit to the data; this is referred to as *memorization.* However, it is clear from the figure that this polynomial does not provide good "generalization" (i.e., it does not provide reliable interpolation and/or extrapolation) over the full range of data. On the other hand, fitting the data by an eighth-order polynomial leads to

relatively better overall interpolations over a wider range of *x* values (refer to the dotted line in Figure Error! No text of specified style in document.**-1**).

In this case, the number of free parameters is equal to 9, which is smaller than the number of training samples. This "underdetermined" nature leads to an approximation function that better matches the "smooth" function *g(x)* being approximated. Trying to use a yet lower-order polynomial (e.g., fifth order or less) leads to a poor approximation because this polynomial would not have sufficient "flexibility" to capture the nonlinear structure in *g(x)*.

The reader is advised to consider the nature and complexity of this simple approximation problem by carefully studying Figure Error! No text of specified style in document.**-1**. Here, the total number of possible training samples of the form (*x*, *g*(*x*)) is uncountably infinite. From this huge set of potential data, though, we chose only 15 samples to try to approximate the function. In this case, the approximation involved minimizing an SSE criterion function over these few sample points.

Clearly, however, a solution that is globally (or near globally) optimal in terms of sum-squared error over the training set (e.g., the eleventh-order polynomial) may be hardly appropriate in terms of interpolation (generalization) between data points.

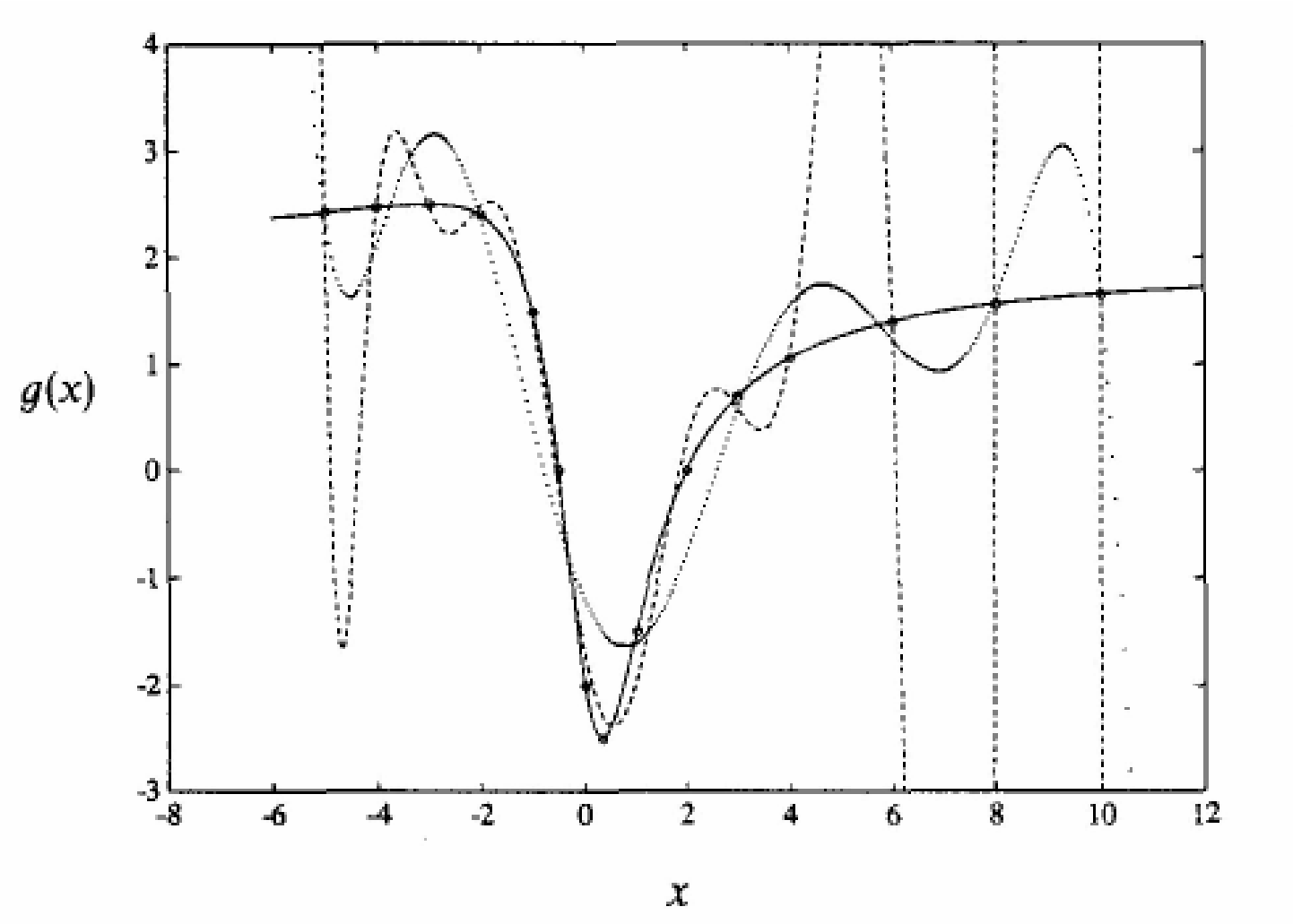

Figure Error! No text of specified style in document.**-1** Polynomial approximation for the function $g(x)=\left[(x-2)(2x+1)\right]/(1+x^2)$ *(solid line)* based on the 15 samples shown *(small circles).* The objective of the approximation is to minimize the sum of squared error criterion. The dashed line represents an eleventh-order polynomial. A better overall approximation for *g(x)* is given by an eighth-order polynomial *(dotted line).*

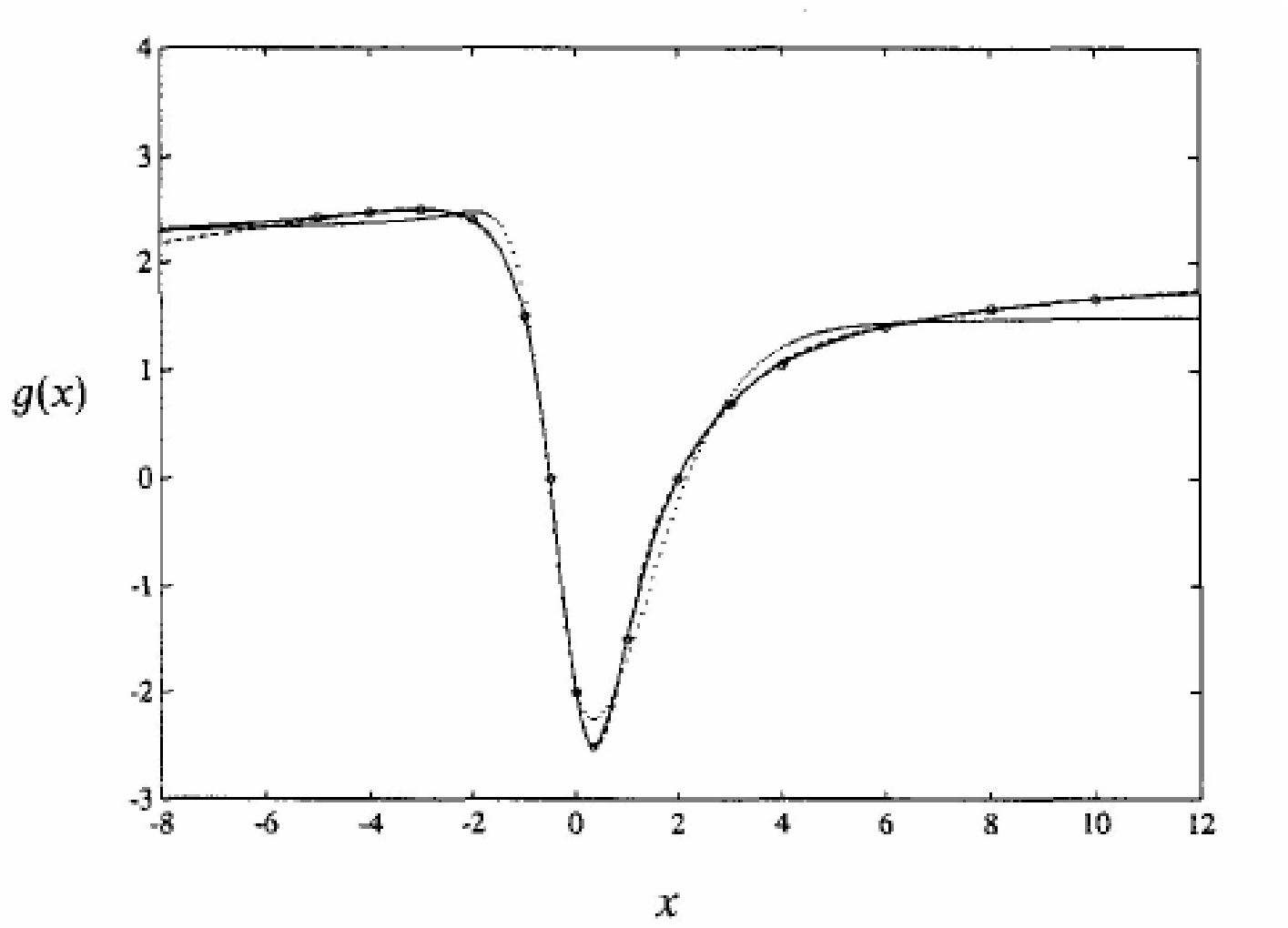

Figure Error! No text of specified style in document.**-2** Neural network approximation for the function $g(x) = \left[(x-2)(2x+1)\right]/(1+x^2)$ (*solid line*). The dotted line was generated by a 3-hidden-unit feedforward net. The dashed line, which is shown to have substantial overlap with *g(x),* was generated by a 12-hidden unit feedforward net. In both cases, standard incremental backprop training was used.

Thus one should choose a class of approximation functions that penalizes unnecessary fluctuations between training sample points. Neural networks satisfy this approximation property and are thus superior to polynomials in approximating arbitrary nonlinear functions from sample points (see further discussion given below). Figure Error! **No text of specified style in document.**-2 shows the results of simulations involving the approximation of the function *g(x)*, with the same set of samples used in the preceding simulations, using single-hidden-layer feedforward neural nets. Here, all hidden units employ the hyperbolic tangent activation function (with a slope of 1), and the output unit is linear. These nets are trained using the incremental backprop algorithm [given by Equations (3.2) and (3.9)] with $\rho_o = 0.05$ and $\rho_h = 0.01$. Weights are initialized randomly and uniformly over the range$[-0.2, +0.2]$. The training was stopped when the rate of change of the SSE became insignificantly small. The dotted line in Figure Error! **No text of specified style in document.**-2 is for a net with three hidden units (which amounts to 10 degrees of freedom). Surprisingly, increasing the number of hidden units to 12 units (37 degrees of freedom) improved the quality of the fit, as shown by the dashed line in the figure. By comparing Figure Error! **No text of specified style in document.**-2 and Figure Error!

No text of specified style in document.-2**,** it is clear that the neural net approximation for *g(x)* is superior to that of polynomials in terms of accurate interpolation and extrapolation.

The generalization superiority of the neural net can be attributed to the bounded and smooth nature of the hidden-unit responses as compared with the potentially divergent nature of polynomials. The bounded-unit response localizes the nonlinear effects of individual hidden units in a neural network and allows for the approximations in different regions of the input space to be independently tuned. This approximation process is similar in its philosophy to the traditional spline technique for curve fitting. Hornik et al. (1990) gave related theoretical justification for the usefulness of feedforward neural nets with sigmoidal hidden units in function approximation. They showed that in addition to approximating the training set, the derivative of the output of the network evaluated at the training data points is also a good approximation of the derivative of the unknown function being approximated[1]. This result explains the good extrapolation capability of neural nets observed in simulations. For example, the behavior of the neural net output shown in Figure Error! **No text of specified**

[1] In addition, Hornik et al. (1990) showed that a multilayer feedforward network can approximate functions that are not differentiable in the classical sense but possess a generalized derivative, as in the case of piece-wise differentiable functions and functions with discontinuities. For example, a neural net with one hidden unit that employs the hyperbolic tangent function and a linear output unit can approximate very accurately the discontinuous function $g(x) = a\,\mathrm{sgn}(x-b) + c$

style in document.-2 for $x > 10$ and $x < -5$ is a case in point. It should be noted, though, that in most practical situations the training data are noisy. Hence an exact fit of these data must be avoided, which means that the degrees of freedom of a neural net approximator must be constrained. Otherwise, the net will have a tendency for overfitting. This issue is explored next.

Once a particular approximation function or network architecture is decided on, generalization can be improved if the number of free parameters in the net is optimized. Since it is difficult to estimate the optimal number of weights (or units) a priori, there has been much interest in techniques that automatically remove excess weights and/or units from a network. These techniques are sometimes referred to as *network pruning algorithms* and are surveyed in Reed (1993).

One of the earliest and simplest approaches to remove excess degrees of freedom from a neural network is through the use of simple weight decay (Plaut et al., 1986), in which each weight decays toward zero at a rate proportional to its magnitude so that connections disappear unless reinforced. Hinton (1987) gave empirical justification by showing that such weight decay improves generalization in feedforward networks. Krogh and Hertz (1992) gave some theoretical justification for this generalization phenomenon.

Weight decay in the weight update equations of backprop can be accounted for by adding a complexity (regularization) term to the criterion function E that penalizes large weights:

$$J(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\sum_i w_i^2 \qquad (3.29)$$

Here, λ represents the relative importance of the complexity term with respect to the error term $E(\mathbf{w})$. Now, gradient search for minima of $J(\mathbf{w})$ leads to the following weight update rule:

$$\Delta w_i = -\rho\frac{\partial E}{\partial w_i} - \rho\lambda w_i \qquad (3.30)$$

which shows an exponential decay in w_i if no learning occurs. Because it penalizes more weights than necessary, the criterion function in Equation (3.29) overly discourages the use of large weights where a single large weight costs much more than many small ones. Weigend et al. (1991) proposed a procedure of weight elimination given by minimizing

$$J(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\sum_i \frac{w_i^2}{w_0^2}\left(1+\frac{w_i^2}{w_0^2}\right)^{-1} \qquad (3.31)$$

where the penalty term on the right-hand side helps regulate weight magnitudes and w_0 is a positive free parameter that must be determined. For large w_0 this procedure reduces to the weight decay procedure described above and hence favors many small weights, whereas if w_0 is small, fewer large weights are favored. Also note that when $|w_i| \gg w_0$, the cost of the weight approaches 1 (times λ), which justifies interpretation of the penalty term as a counter of large weights. In practice, a w_0 close to unity is used. It should be noted that this weight elimination procedure is very sensitive to the choice of λ. A heuristic for adjusting λ dynamically during learning is described in Weigend et al. (1991).

The preceding ideas have been extended to unit elimination. Here, one would start with an excess of hidden units and dynamically discard redundant ones. As an example, one could penalize redundant units by replacing the weight decay term in Equation (3.30) by $-(\rho\lambda)/\left(1+\sum_i w_{ji}^2\right)$ for all weights of hidden units, which leads to the hidden-unit update rule

$$\Delta w_{ji} = -\rho_h \frac{\partial E}{\partial w_{ji}} - \rho\lambda \frac{1}{\left(1+\sum_i w_{ji}^2\right)^2} \qquad (3.32)$$

Generalization in feedforward networks also can be improved by using network construction procedures as opposed to weight or unit pruning. Here, one starts with a small network and allows it to grow gradually (add more units) in response to incoming data. Further details on network construction procedures can be found in Marchand et al. (1990).

3.1.2Cross-Validation

An alternative or complementary strategy to the preceding methods for improving generalization in feedforward neural networks is suggested by findings based on empirical results (e.g. Weigend et al., 1991). In simulations involving backprop training of feedforward nets on noisy data, it is found that the validation (generalization) error decreases monotonically to a minimum but then starts to increase, even as the training error continues to decrease. This phenomenon is depicted in the conceptual plot in Figure Error! No text of specified style in document.-**3** and is illustrated through the computer simulation given next.

Consider the problem of approximating the rational function *g(x)* plotted in Figure Error! **No text of specified style in document.**-2 from a set of noisy sample points. This set of points is generated from the 15 perfect samples, shown in Figure Error! **No text of specified style in document.**-2, by adding zero-mean normally distributed random noise with variance of 0.25. A single-hidden-layer feedforward neural net is used with 12 sigmoidal hidden units and a single linear output unit. It employs incremental backprop training with $\rho_o = 0.05$, $\rho_h = 0.01$, and initial random weights in $[-0.2, +0.2]$. After 80 training cycles on the 15 noisy samples, the net is tested for

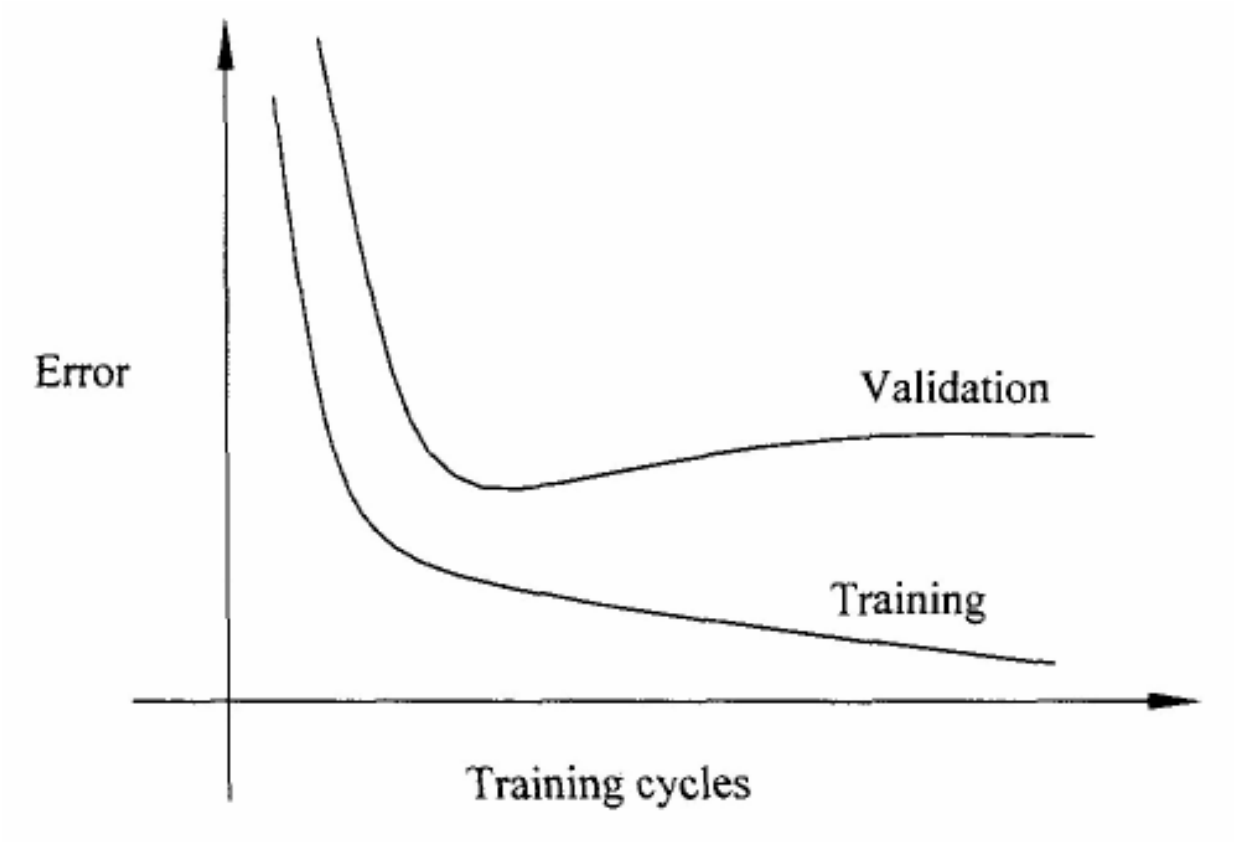

Figure Error! No text of specified style in document.**-3** Training error and validation error encountered in training multilayer feedforward neural nets using backprop.

uniformly sampled inputs x in the range $[-8,12]$. The output of this 80-cycle net is shown as a dashed line in Figure Error! **No text of specified style in document.**-4**.** Next, the training continued and then stopped after 10,000 cycles. The output of the resulting net is shown as a dotted line in the figure.

Comparing the two approximations in Figure Error! **No text of specified style in document.**-4 leads to the conclusion that the partially trained net is superior to the excessively trained net in terms of overall interpolation and extrapolation capabilities. Further insight into the dynamics of the generalization process for this problem can be gained from Figure Error! **No text of specified style in document.**-4**.** Here, the validation RMS error is monitored by testing the net on a validation set of 294 perfect samples, uniformly spaced in the interval $[-8,12]$, after every 10 training cycles. This validation error is shown as the dashed line in Figure Error! **No text of specified style in document.**-5**.** The training error (RMS error on the training set of 15 points) is also shown in the figure as a solid line. Note that the optimal net in terms of overall generalization capability is the one obtained after about 80 to 90 training cycles[2]. Beyond this training point, the training error keeps decreasing, while the validation error increases. It is interesting to note the nonmonotonic behavior of the validation error between training cycles 2000 and 7000. This suggests that, in general, multiple local minima may exist in the validation error curves of backprop-trained feedforward neural networks. The location of these

[2] In general, the global minimum of the validation RMS error curve, which is being used to determine the optimally trained net, is also a function of the number of hidden units/weights in the net. Therefore, the possibility exists for a better approximation than the one obtained here, though, in order to find such an approximation, one would need to repeat the preceding simulations for various numbers of hidden units. The reader should be warned that this example is for illustrative purposes only. For example, if one has 294 (or fewer) "perfect" samples, one would not have to worry about validation; one would just train on the available perfect data! In practice, the validation set is noisy; usually having the same noise statistics as for the training set. Also, in practice, the size of the validation set is smaller than that of the training set Thus the expected generalization capability of what is referred to as the "optimally trained" net may not be as good as those reported here.

minima is a complex function of the network size, weight initialization, and learning parameters. To summarize, when training with noisy data, excessive training usually leads to overfilling.

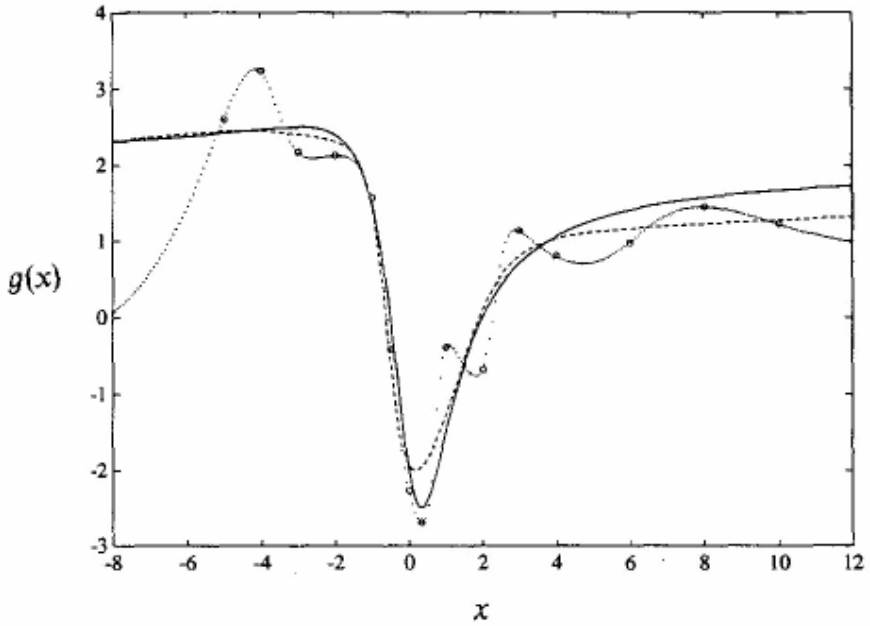

Figure Error! No text of specified style in document.**-4** Two different neural network approximations (dashed lines and dotted lines) of the rational function *g(x) (solid line)* from noisy samples. The training samples shown are generated from the 15 perfect samples in Figure Error! No text of specified style in document.**-2** by adding zero-mean normally distributed random noise with 0.25 variance. Both approximations resulted from the same net with 12 hidden units and incremental backprop learning. The dashed line represents the output of the net after 80 learning cycles. After completing 10,000 learning cycles, the same net generates the dotted line output.

On the other hand, partial training may lead to a better approximation of the unknown function in the sense of improved interpolation and, possibly, improved extrapolation.

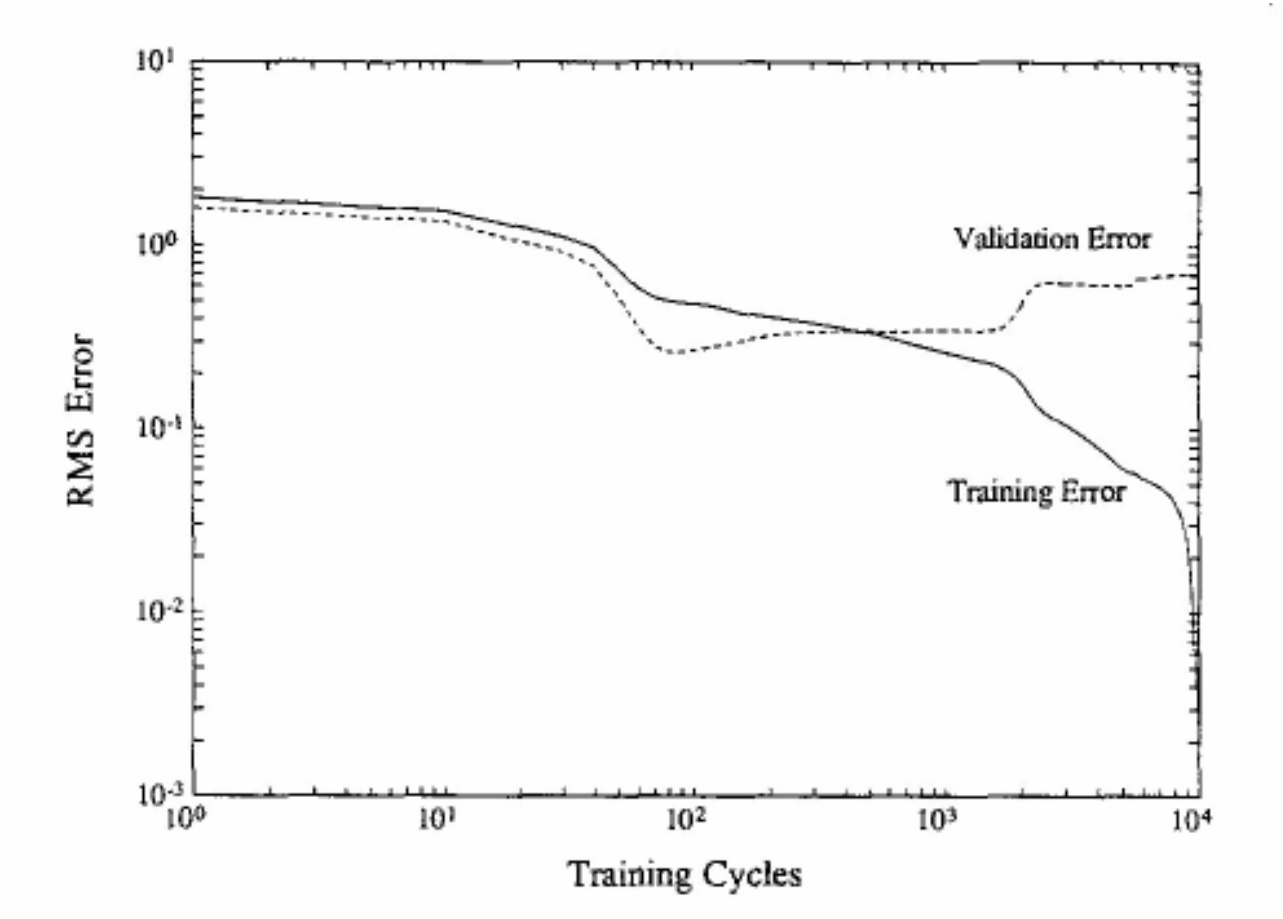

Figure Error! No text of specified style in document.**-5** Training and validation RMS errors for the neural net approximation of the function *g(x)*. The training set consists of the 15 noisy samples in Figure Error! No text of specified style in document.**-4**. The validation set consists of 294 perfect samples uniformly spaced in the interval $[-8,12]$. The validation error starts lower than the training error mainly because perfect samples are used for validation.

A qualitative explanation for the generalization phenomenon depicted in Figure Error! No text of specified style in document.**-3** (and illustrated by the simulation in Figure Error! **No text of specified style in document.**-4 and

Figure Error! **No text of specified style in document.**-5) was advanced by Weigend et al. (1991). They explain that, to a first approximation, backprop initially adapts the hidden units in the network such that they all attempt to fit the major features of the data. Later, as training proceeds, some of the units then start to fit the noise in the data. This later process continues as long as there is error and as long as training continues (this is exactly what happens in the simulation of Figure Error! **No text of specified style in document.**-4). The overall process suggests that the effective number of free parameters (weights) starts small (even if the network is oversized) and gets larger, approaching the true number of adjustable parameters in the network as training proceeds. Baldi and Chauvin (1991) derived analytical results on the behavior of the validation error in LMS-trained single-layer feedforward networks learning the identity map from noisy au-toassociation pattern pairs. Their results agree with the preceding generalization phenomenon in nonlinear multilayer feedforward nets. More recently, Wang et al. (1994) gave a formal justification for the phenomenon of improved generalization by stopping learning before the global minimum of the training error is reached. They showed that there exists a critical region in the training process where the trained network generalizes best, and after that the generalization error will increase. In this critical region, as long as the network is large enough to

learn the examples, the size of the network has only a small effect on the best generalization performance of the network. All this means that stopping learning before the global minimum of the training error has the effect of network size (that is, network complexity) selection.

Therefore, a suitable strategy for improving generalization in networks of non-optimal size is to avoid "overtraining" by carefully monitoring the evolution of the validation error during training and stopping just before it starts to increase. This strategy is based on one of the early criteria in model evaluation known as *cross-validation*. Here, the whole available data set is split into three parts: training set, validation set, and prediction set. The training set is used to determine the values of the weights of the network. The validation set is used for deciding when to terminate training. Training continues as long as the performance on the validation set keeps improving. When it ceases to improve, training is stopped. The third part of the data, the prediction set, is used to estimate the expected performance (generalization) of the trained network on new data. In particular, the prediction set should not be used for validation during the training phase. Note that this heuristic requires the application to be data-rich. Some applications, though, suffer from scarcity of training data, which makes this method inappropriate. The reader is referred to Finnoff et al. (1993) for an

empirical study of cross-validation-based generalization and its comparison to weight decay and other generalization-inducing methods.

3.1.3Criterion Functions

As seen earlier in Section 2.2.5, other criterion/error functions can be used from which new versions of the backprop weight update rules can be derived. Here we consider two such criterion functions: (1) relative entropy and (2) Minkowski-*r*. Starting from the instantaneous entropy criterion (Baum and Wilczek, 1988)

$$E(\mathbf{w}) = \frac{1}{2}\sum_{l=1}^{L}\left[(1+d_l)\ln\left(\frac{1+d_l}{1+y_l}\right)+(1-d_l)\ln\left(\frac{1-d_l}{1-y_l}\right)\right] \qquad (3.33)$$

and employing gradient-descent search, the following learning equations are obtained:

$$\Delta w_{lj} = \rho_o \beta (d_l - y_l) z_j \qquad (3.34)$$

and

$$\Delta w_{ji} = \rho_h \beta^2 \left[\sum_{l=1}^{L}(d_l - y_l) w_{lj}\right](1 - z_j^2) x_i \qquad (3.35)$$

for the output- and hidden-layer units, respectively. Equations (3.34) and (3.35) assume hyperbolic tangent activations at both layers.

From Equation (3.34) we see that the f_o' term present in the corresponding equation of standard backprop [Equation (3.2)] has now been eliminated. Thus the output units do not have a flat-spot problem; on the other hand, f_h' still appears in Equation (3.35) for the hidden units [this derivative appears implicitly as the $\beta\left(1-z_j^2\right)$ term in the standard backprop equation]. Therefore, the flat-spot problem is only partially solved by employing the entropy criterion.

The entropy-based backprop is well suited to probabilistic training data. It has a natural interpretation in terms of learning the correct probabilities of a set of hypotheses represented by the outputs of units in a multilayer neural network. Here, the probability that the *l*th hypothesis is true given an input pattern $\mathbf{x}^k$ is determined by the output of the *l*th unit as $\frac{1}{2}\left(1+y_l\right)$. The entropy criterion is a "well-formed" error function; the reader is referred to Section 2.2.5 for a definition and discussion of "well-formed" error functions. Such functions have been shown in simulations to converge faster than standard backprop.

Another choice is the Minkowski-*r* criterion function (Hanson and Burr, 1988):

$$E(\mathbf{w}) = \frac{1}{r}\sum_{l=1}^{L}\left|d_l - y_l\right|^r \qquad (3.36)$$

which leads to the following weight update equations:

$$\Delta w_{lj} = \rho_o \,\mathrm{sgn}\left(d_l - y_l\right)\left|d_l - y_l\right|^{r-1} f_o'\left(net_l\right) z_j \qquad (3.37)$$

and

$$\Delta w_{ji} = \rho_h \left[\sum_{l=1}^{L} \mathrm{sgn}\left(d_l - y_l\right)\left|d_l - y_l\right|^{r-1} w_{lj}\, f_o'\left(net_l\right)\right] f_h'\left(net_j\right) x_i \qquad (3.38)$$

where sgn is the sign function. These equations reduce to those of standard backprop for the case $r = 2$. The motivation behind the use of this criterion is that it can lead to maximum-likelihood estimation of weights for Gaussian and non-Gaussian input data distributions by appropriately choosing r (e.g., $r = 1$ for data with Laplace distributions). A small r ($1 \le r < 2$) gives less weight for large deviations and tends to reduce the influence of outlier points in the input space during learning. On the other hand, when noise is negligible, the sensitivity of the separating surfaces implemented

by the hidden units to the geometry of the problem may be increased by employing $r > 2$. Here, fewer hidden units are recruited when learning complex nonlinearly separable mappings for larger *r* values (Hanson and Burr, 1988).

If no a priori knowledge is available about the distribution of the training data, it would be difficult to estimate a value for *r* without extensive experimentation with various *r* values (e.g., *r* = 1.5, 2, 3). Alternatively, an automatic method for estimating *r* is possible by adaptively updating *r* in the direction of decreasing *E*. Here, steepest gradient descent on *E*(*r*) results in the update rule

$$\Delta r = \rho \frac{1}{r^2} \sum_l |d_l - y_l|^r - \rho \frac{1}{r} \sum_l |d_l - y_l|^r \ln |d_l - y_l| \qquad (3.38)$$

which when restricting *r* to be strictly greater than 1 (metric error measure case) may be approximated as

$$\Delta r \approx -\rho \frac{1}{r} \sum |d_l - y_l|^r \ln |d_l - y_l| \qquad (3.39)$$

Note that it is important that the *r* update rule be invoked much less frequently than the weight update rule (e.g., *r* is updated once every 10 training cycles of backprop).

The idea of increasing the learning robustness of backprop in noisy environments can be placed in a more general statistical framework where the technique of robust statistics takes effect. Here, *robustness* of learning refers to insensitivity to small perturbations in the underlying probability distribution $p(\mathbf{x})$ of the training set. These statistical techniques motivate the replacement of the linear error $\varepsilon_l = d_l - y_l$ in Equations (3.2) and (3.9) by a nonlinear error suppressor function $f_e(\varepsilon_l)$ that is compatible with the underlying probability density function $p(\mathbf{x})$. One example is to set $f_e(\varepsilon_l) = \text{sgn}(d_l - y_l)|d_l - y_l|^{r-1}$ with $1 \le r < 2$. This error suppressor leads to the exact Minkowski-*r* weight update rule of Equations (3.37) and (3.38). In fact, the case $r = 1$ is equivalent to minimizing the summed absolute error criterion that is known to suppress outlier data points. Similarly, the selection $f_e(\varepsilon_l) = 2\varepsilon_l / (1 + \varepsilon_l^2)$ leads to robust backprop if $p(\mathbf{x})$ has long tails such as a Cauchy distribution or some other infinite variance density.

Furthermore, regularization terms may be added to the preceding error functions $E(\mathbf{w})$ in order to introduce some desirable effects such as good generalization, smaller effective network size, smaller weight magnitudes, faster learning, etc. (Poggio and Girosi, 1990). The regularization terms in

Equations (3.29) and (3.31) used for enhancing generalization through weight pruning/elimination are examples. Another possible regularization term is $\lambda \left\| \nabla_x E \right\|^2$, which has been shown to improve backprop generalization by forcing the output to be insensitive to small changes in the input. It also helps speed up convergence by generating hidden-layer weight distributions that have smaller variances than those generated by standard backpropagation.

Weight sharing, a method where several weights in a network are controlled by a single parameter, is another way of enhancing generalization [Rumelhart et al. (1986b)]. It imposes equality constraints among weights, thus reducing the number of free (effective) parameters in the network, which leads to improved generalization. An automatic method for affecting weight sharing can be derived by adding the regularization term

$$J_R = -\sum_i \ln \left[\sum_j \alpha_j p_j \left(w_i \right) \right] \qquad (3.41)$$

to the error function, where each $p_j \left(w_i \right)$ is a Gaussian density with mean μ_j and variance σ_j, the α_j is the mixing proportion of Gaussian p_j with $\sum_j \alpha_j = 1$, and w_i represents an arbitrary weight in

the network. The α_j, μ_j and σ_j parameters are assumed to adapt as the network learns. The use of multiple adaptive Gaussians allows the implementation of "soft weight sharing," in which the learning algorithm decides for itself which weights should be tied together. If the Gaussians all start with high variance, the initial grouping of weights into subsets will be very soft. As the network learns and the variance shrinks, the groupings become more and more distinct and converge to subsets influenced by the task being learned. For gradient-descent-based adaptation, one may employ the partial derivatives

$$\frac{\partial J_R}{\partial w_i} = -\sum_j r_j(w_i)\left(\frac{\mu_j - w_i}{\sigma_j^2}\right) \qquad (3.42)$$

$$\frac{\partial J_R}{\partial \mu_j} = +\sum_i r_j(w_i)\left(\frac{\mu_j - w_i}{\sigma_j^2}\right) \qquad (3.43)$$

$$\frac{\partial J_R}{\partial \sigma_j} = -\sum_i r_j(w_i)\left[\frac{(\mu_j - w_i)^2 - \sigma_j^2}{\sigma_j^3}\right] \qquad (3.44)$$

and

$$\frac{\partial J_R}{\partial \alpha_j} = \sum_i \left[1 - \frac{r_j(w_i)}{\alpha_j} \right] \qquad (3.45)$$

with

$$r_j(w_j) = \frac{\alpha_j p_j(w_i)}{\sum_k \alpha_k p_k(w_i)} \qquad (3.46)$$

It should be noted that the derivation of the partial of J_R with respect to the mixing proportions is less straightforward than those in Equations (3.42) through (3.44) because the sum of the α_j values must be maintained at 1. Thus the result in Equation (3.45) has been obtained by appropriate use of a Lagrange multiplier method and a bit of algebraic manipulation. The term $r_j(w_i)$ in Equations (3.42) through (3.46) is the posterior probability of Gaussian *j* given weight w_i; i.e., it measures the responsibility of Gaussian *j* for the *i*th weight. Equation (3.42) attempts to pull the weights toward the center of the "responsible" Gaussian. It realizes a competition mechanism among the various Gaussians for taking on responsibility for weight w_i. The partial derivative for μ_j drives μ_j toward

the weighted average of the set of weights for which Gaussian *j* is responsible. Similarly, one may come up with simple interpretations for the derivatives in Equations (3.44) and (3.45). To summarize, the penalty term in Equation (3.41) leads to unsupervised clustering of weights (weight sharing) driven by the biases in the training set.