14 Factors Influencing Generalization

14.1 Definitions

How is it possible to derive general rules from specific cases? Much of science is the search for simple rules to explain observed events. Generalization is an ancient philosophical problem that has been studied from many angles.

In the context of artificial neural networks and supervised learning, generalization is often viewed as an interpolation or approximation problem. That is, the examples are seen as points in a space and the goal is to find a function that interpolates between them in a reasonable way. If the data are noisy or uncertain, constraints may be relaxed to require only that the surface be 'near' the training points.

Other definitions of generalization are also possible. Generalization can be said to occur in associative memories or clustering systems that learn ideal class prototypes after training on specific instances of each class. (In some applications, this may be considered a defect because details of the individual patterns are forgotten.) The interpolation viewpoint also ignores the perhaps more realistic case of reinforcement learning where data occurs in the form of input-response-consequence triplets and there is no teacher or single-valued target function.

The following definition of generalization is used below. The training data consist of examples of the desired input-output relationship, $\{(\mathbf{x}_k, t_k)\}$, $k = 1, \ldots M$, where M is the number of training samples, $\mathbf{x}_k$ is the k^{th} input pattern, and t_k is the corresponding desired output or target. Usually t_k is considered to be a value generated by some unknown target function $f^*(\mathbf{x}_k)$ or a sample drawn from an unknown joint distribution $p(t, \mathbf{x})$. For input $\mathbf{x}_k$ the network produces an output $y_k = y(\mathbf{x}_k)$ in response. Differences between the network output and desired target are measured by some error function E, often the mean squared error (MSE)

$$E^*_{MSE} = E\left[\|t_k - y_k\|^2\right] \tag{14.1}$$

where $E[\cdot]$ denotes expectation. Usually, the true error cannot be measured exactly. When only samples are available, an approximation based on the training set error

$$\hat{E}_{MSE} = \frac{1}{M}\sum_{k=1}^{M} \|t_k - y_k\|^2 \tag{14.2}$$

is often used. Training consists of selecting a set of parameters that (hopefully) minimize the error on all future tests.

With no restrictions on the learning system, we can always find a function that fits any finite data set exactly. If nothing else, we can simply store the training patterns in a look-up

table. The problem is that although the look-up table "learns" the training data perfectly, it cannot cope with novel patterns. What we really want is for the system to generalize from the training examples to the underlying target function so it produces correct (or at least reasonable) outputs in response to new patterns that have not been seen before. A system that learns the training data and also does well on new data is said to generalize well. It fails to generalize when it performs poorly on new data.

14.2 The Need for Additional Information

A fundamental reason for less than perfect generalization is that the problem is ill-posed because the samples alone do not uniquely determine an interpolating function (see figure 14.1). Any of an infinite number of functions passing through the sample points is equally valid according to the sample set error and other criteria are needed to choose among them. If nothing more is known about the target function, there is no basis for selecting one solution over another.

Additional information must be provided. This is often done by biasing the training procedure to favor certain types of solutions, thereby placing constraints on the sets of solutions considered. Different approximation methods can be viewed from this perspective in terms of the biases imposed and how they are implemented. Selection of a neural network rather than a polynomial approximation, decision tree, or some other fitting system already limits the set of solutions that will be considered; further selection of a particular structure, parameters, and training algorithm provides additional constraints. These choices may reflect unrecognized assumptions about the solution.

However they are implemented, the ability of the constraints to lead to good generalization depends on how well they reflect actual properties of the (unknown) target function

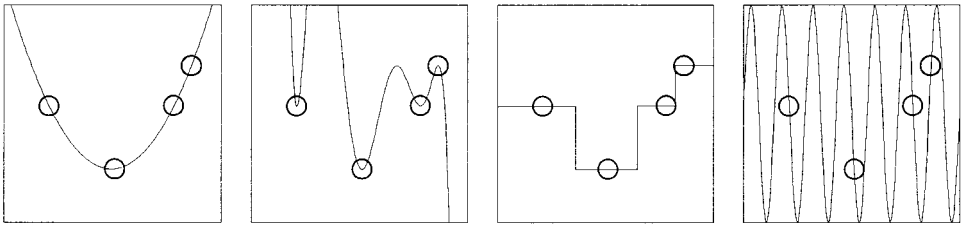

Figure 14.1
Samples alone do not provide enough information to uniquely determine an interpolating function. An infinite number of functions can be fit through the sample points; all are equally valid according to the sample set error and other criteria are needed to choose among them.

and is relatively independent of the optimization technique used to find a solution (assuming they are equally successful in satisfying the constraints).

14.3 Network Complexity versus Target Complexity

In order to generalize well, a system needs to be sufficiently powerful to approximate the target function. If it is too simple to fit even the training data then generalization to new data is also likely to be poor. (The true error may not be much worse than the training error, however, depending on how well the training data represents the target function.) If the network is powerful enough then good generalization is at least possible if not limited by other factors. In contrast to the rule of thumb that simpler is better, the larger network may generalize better since it is more powerful and better able to approximate the true target function. An overly complex system, however, may be able to approximate the data in many different ways that give similar errors and is unlikely to choose the one that will generalize best unless other constraints are imposed.

Figure 14.2 illustrates possible under- and overfitting. The fitting function is a linear combination of M evenly spaced Gaussian basis functions with width inversely proportional to M. At $M = 3$, the approximation is too simple and the error is large. At $M = 5$, the errors are smaller. At $M = 30$, the approximation may be overfitting the data.

Whether a given network overfits or underfits the data depends in part on the size of the training set. Figure 14.3 shows generalization error versus complexity curves for a slightly more complex function fitted by the same system of Gaussian basis functions. In general,

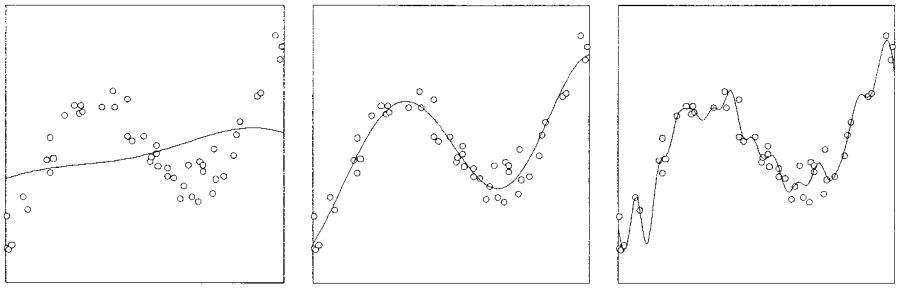

Figure 14.2
Underfitting and overfitting by a minimum-MSE approximation using M evenly spaced Gaussian basis functions with widths inversely proportional to M: (a) underfitting ($M = 3$), the approximation is too simple, (b) perhaps a reasonable fit ($M = 5$), and (c) possible overfitting ($M = 30$).

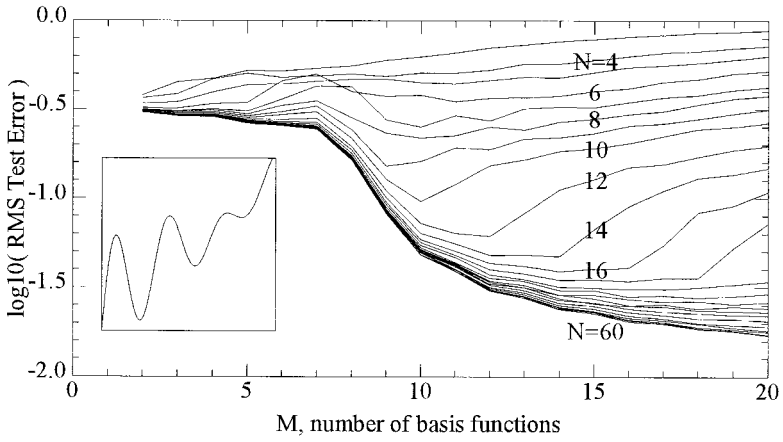

Figure 14.3
Generalization error versus network complexity for a minimum-MSE fit using M evenly spaced Gaussian basis functions for various sample sizes N. For intermediate sample sizes, the curve has a minimum at some intermediate value. (Each point is the average of 200 trials. Training samples are evenly spaced with some jitter but no additive noise. The target function is shown in the inset.)

the curve for a particular sample size N has a minimum at some intermediate complexity value M. Below a certain threshold, the approximation is too simple and all systems have large errors. At high values of M, the system begins to overfit and the error increases.

Unfortunately, if the target function is completely unknown, there is no way to determine a priori if the network is complex enough. Figure 14.2c may be overfitting if the data is noisy and the target function has a form similar to figure 14.2b, but it could be that the data are clean and the actual function is a complex deterministic function in which case figure 14.2b may be underfitting. Additional information is needed.

14.4 The Training Data

Ideally, the number and distribution of training samples should be such that minimizing the error on the samples gives results identical to minimizing the error on the true function. Poor generalization could arise due to problems in the way the training data is sampled. In general, the training set must be representative of the target function. A poor set of training data may contain misleading regularities not found in the underlying function although, when samples are randomly selected, this becomes less likely as the sample size grows.

For neural networks used as function approximators, generalization usually means interpolation. This does not imply an ability to extrapolate (figure 14.4).

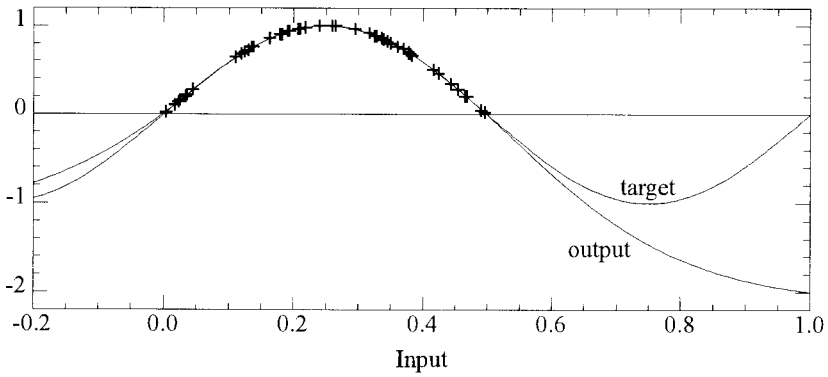

Figure 14.4
For neural networks used as function approximators, generalization usually means interpolation, not extrapolation. A 1/5/1 network with tanh hidden nodes and a linear output was trained on 50 points uniformly distributed in the first half cycle of $f(x) = \sin(2\pi x)$. The fit is good near the training data, but poor elsewhere.

14.4.1 Distribution of the Data

Systems are often developed under conditions slightly different from normal operating conditions and the resulting differences in training and test sample distributions could lead to poor generalization. Approximations using a mean squared error function are sensitive to the sampling density in the sense that the minimizer tends to focus on getting a good fit in densely sampled regions and may ignore large but rare errors in sparsely sampled regions. That is, when infinite numbers of samples were available everywhere, the mean squared error approaches the expected squared error

$$E = \int_{\mathbf{x}} (t(\mathbf{x}) - y(\mathbf{x}))^2 \, p(\mathbf{x}) \, d\mathbf{x}. \tag{14.3}$$

Errors are weighted by the sampling distribution $p(\mathbf{x})$ so the minimizer will be happy to let errors increase in regions where $p(\mathbf{x})$ is low if it can decrease errors in regions where $p(\mathbf{x})$ is high. Good generalization cannot be expected, therefore, if a network is trained on samples from one region but tested on samples from a completely different region. Most training methods assume training and test samples are drawn from the same distribution and modifications will be required if different distributions are used. Other error measures can be less sensitive to this factor than MSE.

When the target function has important features in low density regions, large sample sizes may be needed to obtain enough samples to adequately represent the function there. Large sample sizes will also be needed to obtain accurate estimates of the sampling density. Even if the training and test data share the same sampling function, a very small training set could give a misleading picture of the true density.

Differences in sampling distributions do not always have to cause bad generalization, however. A system trained on harder than average cases may generalize well even if it does only fair on the training data. Concentration on cases near classification boundaries has been useful in some problems [5, 184].

Differences in distribution may also be less important when other information is provided or the network is constrained appropriately. An image recognition system, for example, with a rotation invariant representation can be trained on images in one orientation and tested on images in another because the cases are effectively equivalent under the representation. Sampling differences may also be less important when the target function is simple enough that a fitting function that works well in the densely sampled regions also works in the sparsely sampled regions. That is, if there are so much data that the network is fully constrained everywhere, then small changes in the distribution of the data may have little effect on the solution chosen.

Finally, in some problems, there is the option of choosing the training samples, possibly while learning is in progress. The idea in [12] is to choose training samples that provide the most new information. This can be a useful strategy for any learning system, including people.

14.4.2 Number of Examples versus Target Complexity

Even when the training and test sampling densities match, generalization may suffer if the training set does not capture all significant features of the underlying target function. If the function is a bandlimited sum of sinusoids, for example, and samples are taken at less than the Nyquist rate, then essential features will be missed and the reconstruction will not generalize well.

Constraining knowledge about the generating function can reduce the number of samples needed to choose a hypothesis function. The knowledge that the function is a bandlimited sum of sinusoids, for example, allows a Nyquist rate to be set. (And further knowledge about the probabilities of likely functions may allow sampling at less than the Nyquist rate [327].)

In general, the number of examples required depends on the constraining knowledge. If the target function is known to be a polynomial of degree N or less, say, then $N + 1$ points are sufficient to determine which particular polynomial generated the data. If the target function is, in fact, an Nth order polynomial, then the chosen function should generalize perfectly. Of course, more complex function classes will generally require more samples and, in many cases, the number of samples required may be unreasonably large or unbounded, depending on the constraints. It may also be difficult to calculate the function even if it is known to exist. Finally, the constraining knowledge must be available in a form useful to the training procedure; it is not obvious how knowledge that the target is an Nth

order polynomial could be made useful to a standard neural network training procedure, for example.

14.4.3 Number of Examples versus Network Complexity

In general, larger and more complex systems can compute more functions but need more examples to determine which function to choose (one aspect of the so-called curse of dimensionality). If the number of examples is small compared to the number of degrees of freedom of the network, then the network may be able to implement many functions consistent with the examples. Some of these may generalize better than others, but a learning algorithm guided only by errors on the samples has no way to tell and is unlikely to choose the one that generalizes best. As a rule, increasing the number of samples decreases the number of hypothesis functions consistent with the samples, but it will never reduce it to just one function unless the set of hypothesis functions is finite or there are other constraints.

If a network is underconstrained, that is, if it has more degrees of freedom (the number of weights, roughly) than the number of the training samples, then it may use the extra degrees of freedom to fit noise or spurious correlations in the data. Even though it may produce exactly the right output at each of the training points, it may be very inaccurate at other points. An example is a high-order polynomial fitted through a small number of points. Figure 14.5 shows an example of overfitting by an underconstrained network.

As with polynomial approximations, a rule of thumb is to use the smallest system that fits the data. If the system has only a limited number of degrees of freedom, it will usually use them to adapt to the largest regularities in the data and ignore the smaller, possibly

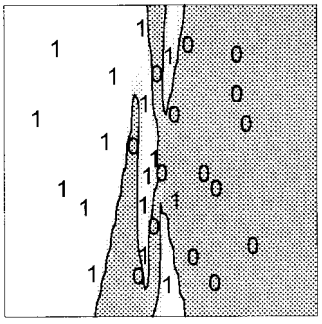

Figure 14.5
A 2/50/10/1 network with 671 weights trained on 31 points is very underconstrained. Although the points are nearly linearly separable, with some overlap, the decision surface is very nonlinear and is unlikely to generalize well. The solid line shows the network decision surface, the 0.5 contour. Other contours are omitted because the transitions are steep.

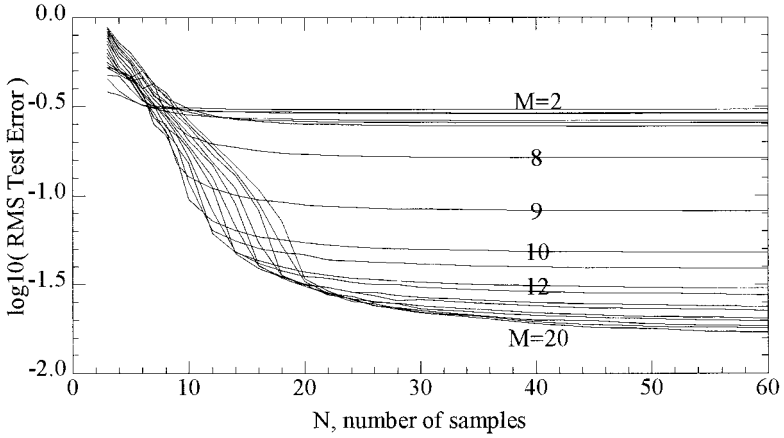

Figure 14.6
Generalization error versus sample size N for the Gaussian basis function approximation system of figure 14.3.

spurious, ones. (It should be noted that "degrees of freedom" is not a well-defined concept for nonlinear systems so it may be difficult to say how many samples are necessary to constrain a given network.)

Another general rule of thumb is that the number of samples should be several times larger than the number W of weights in the network. A simple perceptron (i.e., a linear threshold unit) with d inputs ($W = d + 1$) can always separate $N < d + 1$ arbitrarily labeled points and, if the points are randomly distributed, can separate almost all labelings of $2d$ or fewer points when d is large [87]. (This is discussed in section 3.3.) Thus, $O(3W)$ points may be needed to constrain even a simple perceptron. Of course, this simple rule of thumb may not hold for more complex functions and networks.

Figure 14.6 shows generalization error versus sample size N for the Gaussian basis function approximation system of figure 14.3. Higher complexity approximations (with more basis functions M in this case) yield smaller asymptotic errors, but need more samples before the error approaches the asymptotic value. With small sample sizes, lower complexity systems may generalize better. At $N = 15$, for example, the error of the $M = 20$ system is higher than that of the $M = 10$ system.

Figure 14.7 shows training and test set errors versus sample size N for selected M values. For small sample sizes, the training error is small but the test set error is high. As the sample size increases, the training error increases and the test set error decreases until both approach the same asymptotic value at large N. Note that the large M solutions have lower asymptotes, but the small M solutions approach their asymptotes faster. The

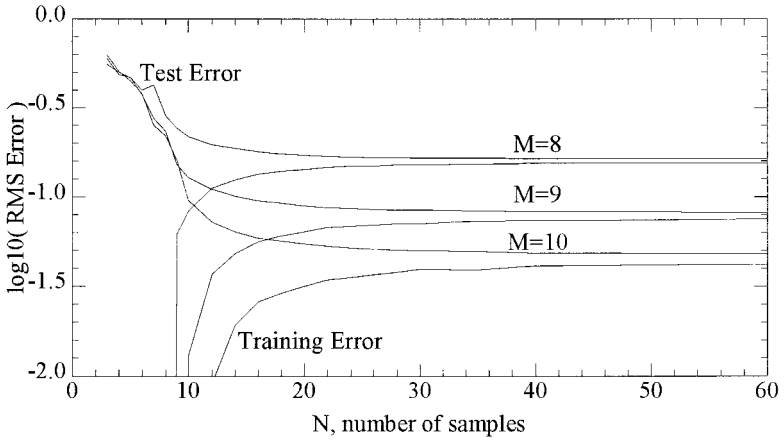

Figure 14.7
Training and test set errors versus sample size N for selected M values of the Gaussian basis function approximation system of figure 14.3. In general, the training set error is small for small sample sizes and rises as the number of samples increases while the test set error is large for small sample sizes and decreases as the number of samples increases. At large N, both approach the same asymptotic value.

figure gives some support to the heuristic that the sample size should be several times as large as the number of free parameters since the knee of the $M = 10$ curve occurs near $N = 10$.

14.4.4 Noise in the Samples

If the training data are noisy, the unknown target function can only be estimated. No finite number of examples will uniquely determine a fitting function even if the number of hypothesis functions is finite and the best that can be done is to choose among candidates based on something like the mean-squared error or the probability of misclassification. If one knows or assumes properties of the noise, then an optimal estimate can be made in some cases.

In general, if the noise is independent and zero-mean, then errors can be reduced by obtaining more samples since independent errors tend to cancel. (Systematic measurement errors will not be reduced by simply averaging over more data, however.) More samples will be needed to reduce the variance of the estimated error to an equivalent level when noise levels are high.

With a static data set whose values are unchanging, there is no obvious indication that the data are noisy. If noise is not considered and the system is allowed to fit the data exactly, noise and all, then generalization may be poor.

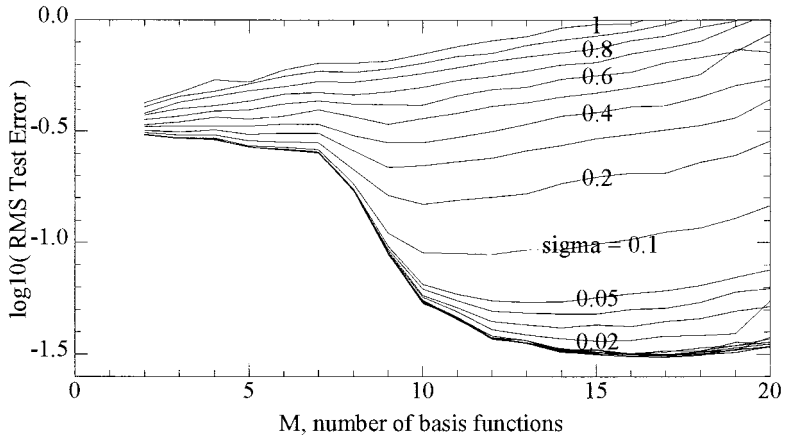

Figure 14.8
Generalization error versus M for various values of the additive noise variance σ. With a fixed sample size and increasing amounts of additive noise, minima of the generalization error curves occur at lower M values. The fitting function is the system of Gaussian basis functions used in figure 14.3. Training sample size $N = 20$. Each point is the average of 200 trials.

Increased amounts of noise may lead to overfitting at lower complexity levels. Figure 14.8 shows test set error *vs.* complexity curves for various levels σ of additive noise in the target values. The fitting function is the system of evenly spaced Gaussian functions system of figure 14.3 with training set size $N = 20$. With increasing amounts of additive noise, the minima of the error curves are higher and occur at lower M values.

14.4.5 Model Mismatch and Number of Examples

Ideally, the class of functions computable by the network would exactly equal the class of the generating function and the number of examples needed to constrain the network would be the same as the number of examples needed to determine the function by other means. This would be a parametric model. Usually however, the target function lies outside the class of functions computed by the network; no set of weights will yield a function that exactly matches the target everywhere so, at best, the network can only approximate the target. In this case, the number of examples needed to determine the best network may differ from the number of samples required to fit a parametric model. Although we need only $N + 1$ samples to fit a function we know to be an N^{th} order polynomial, this information is not available to the network and many more samples might be needed to train it. Because the network and target function classes differ, the network output will be an approximation and generalization will be imperfect even if the network fits an arbitrarily large number of data samples exactly.

14.5 The Learning Algorithm

14.5.1 Limitations of the Learning Algorithm

The size and structure of the network put an upper bound on its representational ability. Limitations of the learning algorithm, however, may prevent that potential from being realized. Some techniques predict generalization performance based on static network properties such as size and assume the learning algorithm will be powerful enough to find a solution if one exists. Of course, most learning algorithms are not perfect and may fail to find some solutions in any reasonable amount of time. Problems such as local minima and speed of convergence must be considered in practice.

14.5.2 Bias of the Learning Algorithm

All learning algorithms (except perhaps pure exhaustive search over an unlimited solution space) have biases. In many cases, bias is introduced in the form of heuristics that are not rigorously justifiable but which seem to help in practice. In any case, the bias helps generalization by favoring likely solutions over unlikely solutions. Of course, every heuristic can fail and if the bias is wrong, performance could suffer. Tailoring the bias to "agree with reality" is one of the most common ways of introducing external constraints necessary for good generalization. Many of the techniques discussed later are simply different ways of doing this.

It could be argued that part of the reason for the success of back-propagation on many problems is that it has a built-in bias for simple solutions. When initialized with small weights, the function computed by the network follows a path of increasing complexity from nearly constant functions to linear functions to more and more nonlinear functions as training continues. Because training is normally stopped as soon as some error criterion is satisfied, the algorithm is more likely to find a simple solution than to find a complex solution that gives the same result. As a result, a large network is not immediately saddled with a high complexity. Of course, back-propagation will not always stop with a simple solution if training continues.

14.5.3 Learning Dynamics: Overtraining/Overfitting

Generalization performance varies over time as the network adapts during training (see figure 14.9). A randomly selected initial configuration is likely to be completely inconsistent with the examples so both the training set and generalization errors are likely to be high before learning begins. During training, the network adapts to decrease the error on the training patterns. In the early stages of learning, the generalization error tends to decrease in step with the training error as the network captures the major features of the underlying

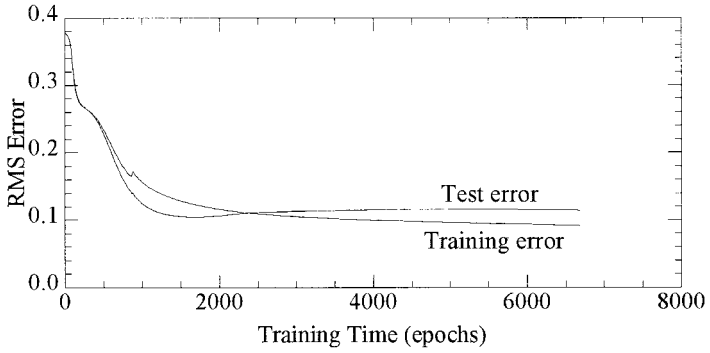

Figure 14.9
As training progresses, the generalization error may decrease to a minimum and then increase again as the network adapts to idiosyncrasies of the training data. Training and test errors are shown for an underconstrained network trained by back-propagation on 30 examples of the iris data and tested on the 120 remaining examples. (An underconstrained net and small learning rate were used to demonstrate overtraining. Smaller networks can learn the data in a much shorter time.)

function. If the training data are noisy or incomplete, however, they may contain misleading regularities. In addition to representing the general properties of the target function, it is likely to contain peculiarities unique to the particular data set and uncharacteristic of the target function. As these idiosyncrasies are exploited in later stages of learning, the improvement in generalization that comes from being right on the training examples is offset by errors (invisible to the learning algorithm) introduced elsewhere and the generalization error begins to increase again even though the training error continues to decrease. Chauvin [73] describes an example of this type of overtraining.

Thus, for a given underconstrained network, set of training data, and learning algorithm, there may be an optimal amount of training that gives the best generalization. Although further training might decrease the training-set error, it would increase the expected generalization error. The relationship between generalization and overtraining has been examined in many studies, for example [73, 74, 16].

Another way to view the situation is illustrated in figure 14.10. The training and test errors can be thought of as surfaces in the weight space. Neural networks are trained by adapting the weights to minimize the error on the training set. The training error surface is likely to be similar to the true error surface, but distorted somewhat depending on the training data. In particular, the minimum of the training error surface is likely to be displaced from the true minimum. Depending on how the network is initialized, the weight trajectory may pass by a true minimum on its way to the apparent (training error) minimum. The observed generalization error would then show an overall decreasing trend as the

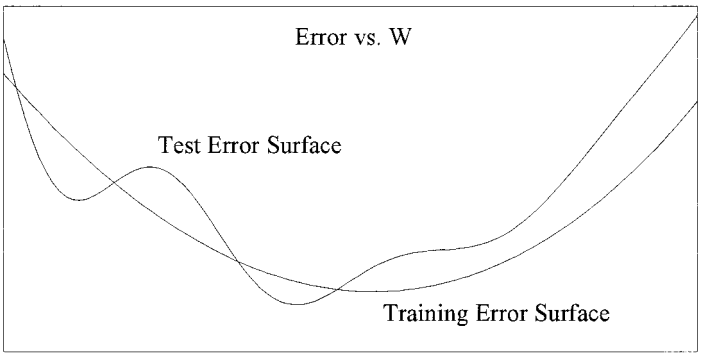

Figure 14.10
One explanation of overtraining is that the training error surface is similar to the true error surface but somewhat distorted so the apparent minimum is offset from the true minimum. The figure shows hypothetical error surfaces. Depending on how the network is initialized, the weight trajectory may pass over a true minimum on its way the apparent (training error) minimum.

network trajectory approaches both minima, followed by an increase as the network passes the true minimum and continues on to the false minimum.

Figure 14.10 also illustrates that whether or not overtraining is observed may depend on initial conditions [74] because a trajectory starting on the left side of the figure passes over the true minimum while one approaching from the right side does not. In other words, the fact that overtraining is not observed in one trial does not prove that the training data are adequate and that it will not occur in another trial with a different initialization. Also, since the training error surface may be distorted and offset in different ways depending on the training data, observation of overtraining may depend on peculiarities of different training sets even when the network is always initialized in the same state. The figure also shows that the generalization error may not decrease monotonically to a single minimum [16] and that local minima could confuse simple early-stopping schemes that halt when the validation error first bottoms out.

Example: Effect of Training Algorithms Different training algorithms may give different generalization results depending on their built-in biases and ability to "exploit loopholes" in the training criteria. Figure 14.11 shows input-output responses of a 2/2/1 network trained by various algorithms on the 2-bit XOR problem. Inputs were coded as -1 and $+1$; targets were -0.9 and 0.9. Tanh nonlinearities were used. The corners of the squares correspond to the four training points. All the networks were trained to 100% correct response with an output considered correct if the error was less than or equal to 0.1. Batch and on-line learning, although not the fastest methods, give reasonably smooth and symmetric responses here; similar inputs would give similar outputs. The networks trained by

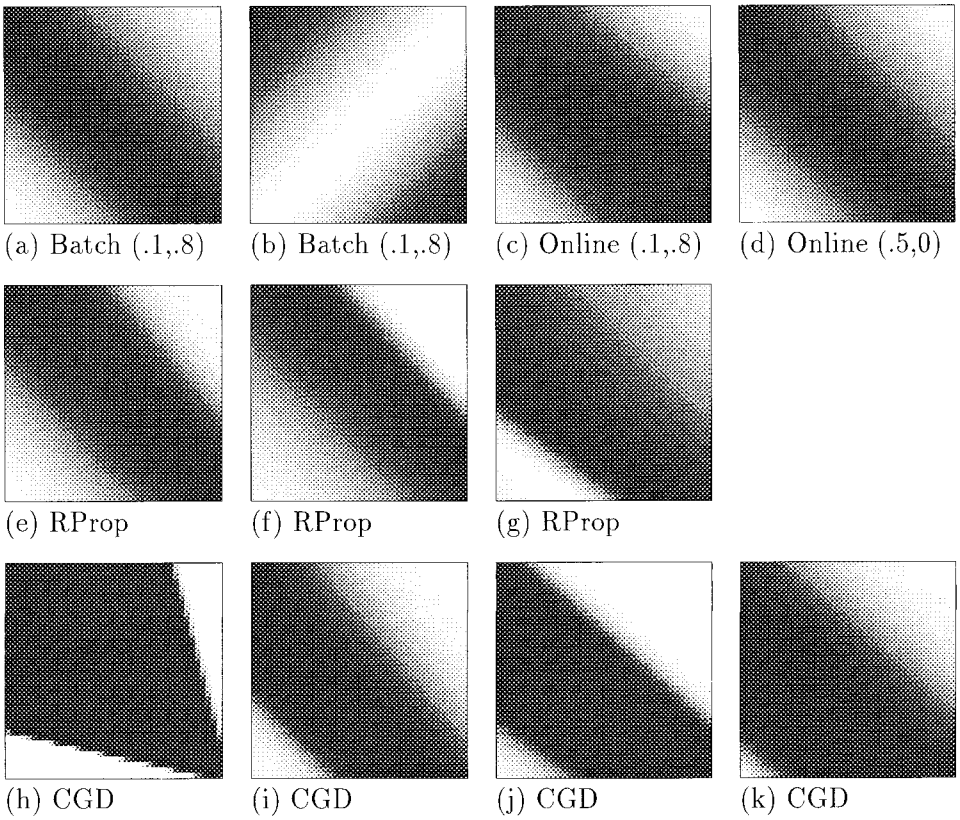

Figure 14.11
Different training algorithms may yield different generalization results. Shown are responses for a 2/2/1 net trained by various algorithms on the 2-bit XOR problem. Batch and "on-line" training appear to give reasonably smooth symmetric responses here (values in parentheses are the learning rate and momentum). RProp and conjugate gradient descent (CGD) appear to create less symmetric surfaces and allow sharp transitions near the training points at the corners. Networks (h), (j), and (k) appear sensitive to the training data and are unlikely to generalize well if the input values are changed slightly.

RProp [315] and conjugate gradient descent, faster methods, for example, show a tendency to asymmetric responses with sharp transitions close to the training points. This might be expected to lead to poor generalization on slightly different inputs; nets h, j, and k in particular would be sensitive to changes in the input data.

Although responses (a–d) might be preferred because they are symmetric and smooth, these were not among the training criteria. All the nets are equally good on the basis of training error. If smooth, symmetric responses are desired, this information has to be provided to the training procedure in some way.

The figure also illustrates that selection of a minimal network is not sufficient to guarantee good generalization. The 2/2/1 networks are nearly minimal (some of the bias connections may not be necessary) for the 2-bit XOR problem unless short-cut connections are allowed, but the instances will probably generalize differently.

14.6 Other Factors

Many other factors have strong effects on the difficulty of a learning task and thus on how well a system can be expected to generalize. The following lists a few items that involve higher-level decisions and generally fall outside the scope of designing a network to fit a given data set. Most are basic principles of good system design.

14.6.1 Choice of Error Function

It hardly needs to be said that the way in which errors are measured has a direct effect on the errors observed. It is generally assumed that identical measures are used for training and test errors, but it is common to choose an error function (e.g., mean squared error) because it is simple and convenient to use even though the real performance may be measured differently (misclassification rate, efficiency, etc.). Poor generalization due to training on one task and testing on another would not be surprising.

Poor performance might result because an inappropriate function is used to measure the error. From a Bayesian viewpoint, different error functions reflect different assumptions about the distribution of the model errors. The mean-squared-error function corresponds to selection of a maximum likelihood model under the assumption that the errors have a Gaussian distribution and is appropriate in ordinary linear regression where the errors are expected to cluster around zero with large errors less likely than small ones. For classification tasks with {0, 1} targets in which the network outputs are viewed as probabilities that the input belongs to a particular class, the cross-entropy error function is generally appropriate. Other error functions are appropriate under different assumptions about the error distribution.

14.6.2 Variable Selection

The selection of input and output variables (i.e., the choice of what information to provide, apart from how it is represented) is an extremely important factor in the difficulty of a learning problem. Certain pieces of information may make a problem very easy. The lack of crucial information may make a problem very difficult or impossible, or change it from a logical problem to a statistical problem. Of course, this is completely problem dependent and falls more in the realm of problem design than network training.

When the chosen variables do not supply needed information, identical inputs may have different targets due to differences in unsupplied variables. Noise in the input patterns can have a similar effect if it destroys necessary information and causes classes to overlap. In either case, the target is not a single-valued function and some error will always remain for any function the network chooses.

Even when the choice of variables does not introduce ambiguity, it still influences the complexity of the learning task and, when training data are limited, may determine if the data are sufficient to describe the target. If the target function is so complex when expressed in terms of the given variables that the available data are insufficient to describe it, then poor generalization could result. Another choice of variables might make the problem simple enough so that the data are adequate and good generalization is possible. The two-spirals problem [233] (illustrated in figure 12.3) is a hard benchmark problem for MLP networks in Cartesian coordinates, but easy in cylindrical coordinates.

It is also possible to confuse a network by supplying too much information in the form of redundant or irrelevant inputs. These increase the number of parameters in the system without supplying much usable information. Irrelevant inputs supply no useful information by definition, but when sample sizes are small the irrelevant inputs may have spurious correlations with the targets. More data will be required to demonstrate that they actually are irrelevant.

14.6.3 Variable Representation

The choice of how variables are represented to the network is also an important factor in learning difficulty. In cases where the network must interface with an external system, the choice may be fixed; in other cases, the representation is a free parameter. There is often a trade-off between economy of representation and decoding complexity. A one-dimensional variable could be coded by the activity on a single input unit; this is economical, but may make learning difficult if the function depends on it in a complex way. The use of a fine-grained "thermometer" code, on the other hand, might make learning easy, but cause generalization to suffer because the number of weights increases while the number of training samples stays fixed.

It is often desirable to choose representations that are invariant to certain irrelevant transformations; for example, invariance to shifts, scale, color, small rotations, and so on can be useful in character recognition. Of course, pre- or postprocessing may be needed to connect the network to the raw data and the cost has to be balanced against how much it simplifies the learning problem. Improvement in generalization due to the use of error-correcting output representations is suggested by Dietterich and Bakiri [107].

The choice of internal representation is also important and is determined in a broad way by the selection of the network structure. Local internal representations (as in radial basis

functions, Kohonen maps, etc.) often make learning easy, but often do not generalize as well as global internal representations (e.g., sigmoidal hidden units). Most of these notes apply to any network architecture, but the focus here is on layered sigmoidal networks, so these differences in architecture will not be considered.

14.6.4 Modularity

Many practical problems can be partitioned into independent subproblems. If the system designer knows this, then the information should be incorporated in the network structure rather than requiring the network to learn it from the examples. Separate networks can then be trained independently for each subproblem and combined. The result is (1) shorter training times because each subnetwork is smaller, and (2) better generalization because each subnetwork is better constrained by the available examples. Say, for example, that a problem has two input variables, x_1 and x_2, and can be separated into two independent subproblems $y_1(x_1)$ and $y_2(x_2)$. If each input can take m values, then $O(m)$ examples describe each function adequately. To train a single network to solve both problems at once, $O(m^2)$ examples would be needed to describe the system adequately. If there are few examples and spurious correlations exist between y_2 and x_1, for example, the network is likely to take advantage of them and generalize poorly as a result.

14.7 Summary

Generalization is influenced by many factors. Items considered in this chapter include the following:

- The samples alone are insufficient to choose a good generalizer; more information must be provided. This is usually done by biasing the learning procedure in some way.
- The biases of the learning procedure must fit reality.
- The network must be powerful enough to fit the target.
- The data must be representative of the target function; there must be sufficient data to illustrate the target.
- The distribution of the training and test data should be similar.
- More data will generally be needed to achieve the same accuracy when the data are noisy.
- There must be enough data to constrain the net (or the net must be chosen so that it is constrained by the data and biases of the training procedure).
- Weaknesses of the learning procedure may lead to poor solutions.

- Dynamics of the learning procedure cause generalization to vary with training time; excessive training may lead to overfitting.
- The same error measure should be used for training and testing.
- Input variables should supply necessary information.
- An appropriate data representation should be chosen.
- Independent subproblems should be assigned to independent networks, rather than combined in a single system.

Remarks The sections in this chapter list many factors that affect generalization and may give the impression that useful approximation is almost impossible because so many things could go wrong. However, the intent is to examine factors that need to be considered and might be encountered at one time or another in different problems. In most problems, many of these factors will not be critical.

Neural networks are often used to solve problems with hundreds of variables in spite of the curse of dimensionality that could make such problems very hard. Many problems turn out to be easier than expected. It is not clear why this happens. Some possible reasons are suggested in [79]:

- application studies use clever preprocessing or data encodings to simplify the learning task;
- many input variables are interdependent so the effective dimensionality is small;
- the high-dimensionality data has a nonuniform distribution or forms clusters that favor local representation methods; and
- getting the right bias (net structure, parameters, learning algorithm, etc.) may be much more important than learning from the data. An appropriate bias reduces the need for data.