17 Effects of Training with Noisy Inputs

Noise is usually considered undesirable—something to be eliminated if possible, but many studies (e.g. [299, 118, 310, 387, 345, 246, 287, 339, 267]) have noted that adding small amounts of noise to input patterns during training often results in better generalization and fault tolerance.

A short explanation for these results is that the noise blurs the data. When random noise is added every time a pattern is presented, the network never sees exactly that same input twice, even when the same training pattern is selected, so it cannot simply "memorize" the training data. Averaging over the noise effectively smooths the target function and prevents the network from overfitting a limited set of training data. This turns out to be helpful for generalization because many of the functions that interest people tend to be smooth.

The following sections examine these ideas in more detail. The term jitter is used to refer to noise intentionally added to the inputs in contrast to undesired, uncontrolled noise from other sources.

17.1 Convolution Property of Training with Jitter

Consider a network trained with noisy input data, $\{\mathbf{x} + \mathbf{n}, t(\mathbf{x})\}$, where $\mathbf{n}$ is noise that varies with each presentation. During training, the network sees the clean target $t(\mathbf{x})$ in conjunction with the noisy input $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{n}$. The input $\tilde{\mathbf{x}}$ seen by the network may be produced by various combinations of inputs $\mathbf{x}$ and noises $\mathbf{n}$, while the target depends only on $\mathbf{x}$. Various targets may therefore be associated with the same noisy input $\tilde{\mathbf{x}}$. The network, however, can produce only a single output for any given input. For arbitrary noise and input sampling distributions, the effective target for a given input $\tilde{\mathbf{x}}$ is the expected value of the target given the noisy input

$$\left\langle t(\mathbf{x}) \mid \tilde{\mathbf{x}} \right\rangle = \frac{\int_n t(\tilde{\mathbf{x}} - \mathbf{n}) p_{\mathbf{x}}(\tilde{\mathbf{x}} - \mathbf{n}) p_{\mathbf{n}}(\mathbf{n})\, d\mathbf{n}}{\int_\xi p_{\mathbf{x}}(\tilde{\mathbf{x}} - \xi) p_{\mathbf{n}}(\xi)\, d\xi}. \tag{17.1}$$

In the special case where the distribution $p_{\mathbf{x}}$ of the training inputs is uniform and the standard deviation of the noise is small relative to the extent of the input domain, the interaction between $p_{\mathbf{x}}$ and $p_{\mathbf{n}}$ will have little effect in the interior of the domain. In regions where these boundary effects can be ignored, the $p_{\mathbf{x}}$ terms cancel, the denominator integrates to one, and this simplifies to the approximation

$$\begin{aligned} \left\langle t(\tilde{\mathbf{x}} - \mathbf{n}) \mid \tilde{\mathbf{x}} \right\rangle &\approx \int_{\mathbf{n}} t(\tilde{\mathbf{x}} - \mathbf{n}) p_{\mathbf{n}}(\mathbf{n})\, d\mathbf{n} \\ &\approx t(\tilde{\mathbf{x}}) * p_{\mathbf{n}}(\tilde{\mathbf{x}}). \end{aligned} \tag{17.2}$$

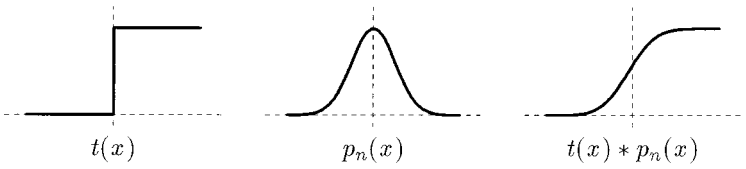

Figure 17.1
Convolution tends to be a smoothing operation. A step function, $t(\mathbf{x})$, convolved with a Gaussian, $p_{\mathbf{n}}(\mathbf{x})$, produces the Gaussian cumulative distribution. This resembles the original step function, but it is a smooth function similar to the sigmoid.

Thus, in this special case, the effective target when training with jittered input data is approximately equal to the convolution of the original target $t(\mathbf{x})$ and the noise density $p_{\mathbf{n}}(\mathbf{x})$.

Convolution tends to be a smoothing operation in general. If, for example, a step function, $t(\mathbf{x})$, is convolved with a Gaussian, $p_{\mathbf{n}}(\mathbf{x})$, the result is the Gaussian cumulative distribution which is a smoothed step function similar to the sigmoid (figure 17.1). This convolutional property resulting from jittered sampling is described by Marks [257].

Holmström and Koistinen [174, 214, 215] showed that training with jitter is consistent in that, under appropriate conditions, the resulting error function approaches the true error function as the number of training samples increases and the amount of added input noise decreases.

17.1.1 Effective Target for Sampled Data

The convolution property holds when training data are continuously and uniformly distributed over the entire input space and the magnitude of the noise is small. In practice, however, we usually have only a finite number of discrete samples $\{(\mathbf{x}_i, t_i)\}$ of the underlying function and the samples are not uniformly distributed in general. In this case, the distribution of $\tilde{\mathbf{x}} = \mathbf{x}_i + \mathbf{n}$ is not uniform and the optimum output function is modified.

Let the training set be $\{(\mathbf{x}_i, t_i) \mid i = 1 \ldots M\}$. During training, we randomly select one of the training pairs with equal probability, add noise to the input vector, and apply it to the network. Given that the training point is $\mathbf{x}_k$, a randomly selected point from the training set, the probability density of the noisy input $\tilde{\mathbf{x}}$ is

$$P\left[\mathbf{x} + \mathbf{n} = \tilde{\mathbf{x}} \mid \mathbf{x} = \mathbf{x}_k\right] = p_{\mathbf{n}}(\tilde{\mathbf{x}} - \mathbf{x}_k).$$

Training points are selected from the training set with equal probabilities $P[\mathbf{x} = \mathbf{x}_k] = 1/M$ so the probability density of the input seen by the network, $\tilde{\mathbf{x}}$, is

$$
\begin{aligned}
P\left[\tilde{\mathbf{x}}\right] &\equiv P\left[\mathbf{x}+\mathbf{n}=\tilde{\mathbf{x}}\right] \\
&= \sum_{k=1}^{M} P\left[\mathbf{x}+\mathbf{n}=\tilde{\mathbf{x}} \mid \mathbf{x}=\mathbf{x}_k\right]\ P\left[\mathbf{x}=\mathbf{x}_k\right] \\
&= \frac{1}{M}\sum_{k=1}^{M} p_{\mathbf{n}}(\tilde{\mathbf{x}}-\mathbf{x}_k).
\end{aligned}
\tag{17.3}
$$

Given that a particular noisy input $\tilde{\mathbf{x}}$ is observed, the probability that it is generated by training data $\mathbf{x}_k$ plus noise is found by Bayes' rule

$$
\begin{aligned}
P\left[\mathbf{x}=\mathbf{x}_k \mid \mathbf{x}+\mathbf{n}=\tilde{\mathbf{x}}\right] &= \frac{P\left[\mathbf{x}+\mathbf{n}=\tilde{\mathbf{x}} \mid \mathbf{x}=\mathbf{x}_k\right] P\left[\mathbf{x}=\mathbf{x}_k\right]}{P\left[\mathbf{x}+\mathbf{n}=\tilde{\mathbf{x}}\right]} \\
&= \frac{p_{\mathbf{n}}(\tilde{\mathbf{x}}-\mathbf{x}_k)(1/M)}{(1/M)\sum_{j=1}^{M} p_{\mathbf{n}}(\tilde{\mathbf{x}}-\mathbf{x}_j)} \\
&= \frac{p_{\mathbf{n}}(\tilde{\mathbf{x}}-\mathbf{x}_k)}{\sum_{j=1}^{M} p_{\mathbf{n}}(\tilde{\mathbf{x}}-\mathbf{x}_j)}.
\end{aligned}
\tag{17.4}
$$

Let P_k denote this probability.

The expected value of the training target, given the noisy training input $\tilde{\mathbf{x}}$, is then

$$
\begin{aligned}
\langle t_{\text{train}}(\tilde{\mathbf{x}}-\mathbf{n}) \mid \tilde{\mathbf{x}}\rangle &= \sum_i t_i P_i \\
&= \sum_i \frac{t_i\, p_{\mathbf{n}}(\tilde{\mathbf{x}}-\mathbf{x}_i)}{\sum_k p_{\mathbf{n}}(\tilde{\mathbf{x}}-\mathbf{x}_k)}.
\end{aligned}
\tag{17.5}
$$

This is the expected value of the training target given that the input is a noisy version of one of the training samples. As the number of samples approaches ∞, the distribution of the samples approaches $p_{\mathbf{x}}$ and equation 17.5 becomes a good approximation to equation 17.1.

Let $y(\tilde{\mathbf{x}})$ be the network output for the input $\tilde{\mathbf{x}}$. The expected value of the error while training, given this input, is

$$
\tilde{\mathcal{E}} = \sum_{i=1}^{M} (t_i - y(\tilde{\mathbf{x}}))^2 P_i. \tag{17.6}
$$

Abbreviate $y(\tilde{\mathbf{x}})$ with y. After expanding the square,

$$\begin{aligned}\tilde{\mathcal{E}} &= (\Sigma t_i^2 P_i) - 2y(\Sigma t_i P_i) + y^2 \Sigma P_i \\ &= (\Sigma t_i^2 P_i) - (\Sigma t_i P_i)^2 + \left[(\Sigma t_i P_i)^2 - 2y(\Sigma t_i P_i) + y^2 \Sigma P_i\right] \\ &= (\Sigma t_i^2 P_i) - (\Sigma t_i P_i)^2 + \left[(\Sigma_i t_i P_i) - y\right]^2,\end{aligned}$$

and

$$\frac{\partial \tilde{\mathcal{E}}}{\partial w} = -2\left[(\Sigma_i t_i P_i) - y\right]\frac{\partial y}{\partial w}. \tag{17.7}$$

In other words, under gradient descent, the system acts as if the target function is $\Sigma_i t_i P_i$, the expected value of the target in equation 17.5, given the conditions stated for P_i. This is a well-known result in optimal least squares estimation: the function that minimizes the sum-of-squares error is the expected value of the target given the input. From equation 17.7 the effective error function is

$$E_{eff} = \left[(\Sigma_i t_i P_i) - y\right]^2 \tag{17.8}$$

in the sense that $\frac{\partial E_{eff}}{\partial w} = \frac{\partial \tilde{\mathcal{E}}}{\partial w}$. In contrast to conventional training where the target is defined only at the training points, the effective target when training with jittered inputs is a function defined for all inputs $\mathbf{x}$.

Figure 17.2 illustrates the point. Figure 17.2(a) shows the Voronoi map of a set of points in two dimensions, the basis for a nearest neighbor classifier. Figure 17.2(b) is expression

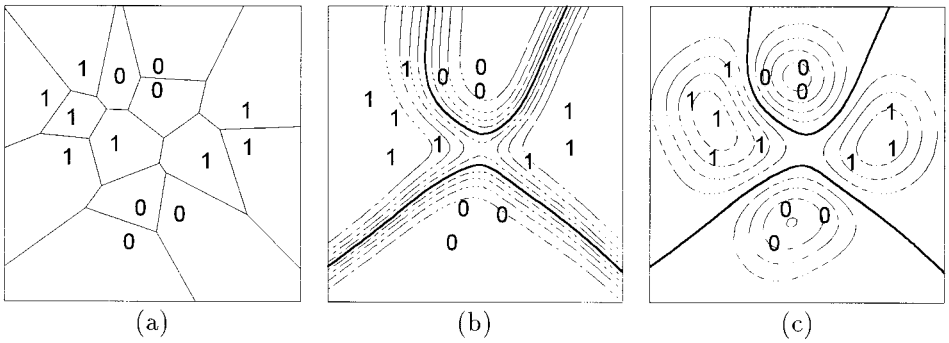

Figure 17.2
A nearest neighbor classification problem: (a) the Voronoi map for 14 points; and (b) the expected value of the classification given the noisy input as calculated in (17.5) for a Gaussian noise distribution with $\sigma = 0.1$. (c) The convolution of the training set with the same Gaussian noise. The zero contours of (b) and (c) coincide. (1s and 0s indicate classes; values 1 and -1 were actually used.)

(17.5) for the expected target given the noisy input. Figure 17.2(c) shows the convolution of the sampled target function with a Gaussian function; the convolution smooths the nearest neighbor decision surface and removes small features. Note that the zero contours in figures 17.2(b) and 17.2(c) coincide.

17.2 Error Regularization and Training with Jitter

Regularization is another method often used to improve generalization. In regularization, one often assumes that the target function is smooth and that small changes in the input do not cause large changes in the output. Poggio and Girosi [300], for example, suggest the cost function

$$\sum_i (y_i - t_i)^2 + \lambda \| Py \|^2 \tag{17.9}$$

where 'P is usually a differential operator' and λ balances the trade-off between smoothing and minimizing the error.

Jittering the inputs while keeping the target fixed embodies this smoothness assumption and results in a similar cost function. That is, we add small amounts of noise to the input data, assume that the target function does not change much, and minimize

$$\mathcal{E} = \{\langle \left[t(\mathbf{x}) - y(\mathbf{x} + \mathbf{n}) \right]^2 \rangle\} \tag{17.10}$$

$$= \{t(\mathbf{x})^2 - 2t(\mathbf{x})\langle y(\mathbf{x} + \mathbf{n})\rangle + \langle y(\mathbf{x} + \mathbf{n})^2 \rangle\} \tag{17.11}$$

where $\{u\}$ indicates the expected value of u over the training patterns and $\langle u \rangle$ indicates the expected value of u over the noise $\mathbf{n}$. For small magnitude noise, $\|\mathbf{n}\| \approx 0$, the network output can be approximated by the linear terms of a truncated Taylor series expansion

$$y(\mathbf{x} + \mathbf{n}) \approx y(\mathbf{x}) + \mathbf{g}^T \mathbf{n} \tag{17.12}$$

where $\mathbf{g} = \frac{\partial y}{\partial \mathbf{x}}$ is the gradient of the output with respect to the input. (A second-order approximation is given in appendix C.1.

Substitution into equation 17.11 and dropping the independent variable for brevity gives

$$\mathcal{E} \approx \left\{ \begin{array}{c} t^2 - 2ty - 2t\mathbf{g}^T \langle \mathbf{n} \rangle \\ + y^2 + 2y\mathbf{g}^T \langle \mathbf{n} \rangle + \mathbf{g}^T \langle \mathbf{n}\, \mathbf{n}^T \rangle \mathbf{g} \end{array} \right\}. \tag{17.13}$$

Assume zero-mean uncorrelated noise with equal variances, $\langle \mathbf{n} \rangle = 0$ and $\langle \mathbf{n}\, \mathbf{n}^T \rangle = \sigma^2 \mathrm{I}$. Then

$$\mathcal{E} \approx \left\{ t^2 - 2ty + y^2 + \sigma^2 \mathbf{g}^T \mathbf{g} \right\} \tag{17.14}$$

$$\approx \{(t-y)^2\} + \sigma^2 \left\{ \|\mathbf{g}\|^2 \right\}. \tag{17.15}$$

The term $\{(t-y)^2\} = E$ is the conventional unregularized error function and the term $\{\|\mathbf{g}\|^2\}$ is the squared magnitude of the gradient of $y(\mathbf{x})$ averaged over the training points.

$\mathcal{E}$ is an approximation to the regularized error function in equation 17.9. Like equation 17.9, it introduces a term which encourages smooth solutions [384, 42]. Comparison of equations 17.15 and 17.9 shows that σ^2 plays a role similar to λ in the regularization equation, balancing smoothness and error minimization. They differ in that training with jitter minimizes the gradient term at the training points whereas regularization usually seeks to minimize it for all $\mathbf{x}$.

Equation 17.15 shows that, when it can do so without increasing the conventional error, the system minimizes sensitivity to input noise by reducing the magnitude of the gradient of the transfer function at the training points. A similar result is derived in [260] and, by analogy with the ridge estimate method of linear regression, in [259]. A system that explicitly calculates and back-propagates similar terms in a multilayer perceptron is described by Ducker and Le Cun [112]. A more general approach using the Hessian information is described by Bishop [40, 42, 43]. A stronger result equating training with jitter and Tikhonov regularization is reported in [45].

Figure 17.3 illustrates the smoothing effect of training with input jitter. Figure 17.3(a) shows the decision boundary formed by an intentionally overtrained 2/50/10/1 feedforward

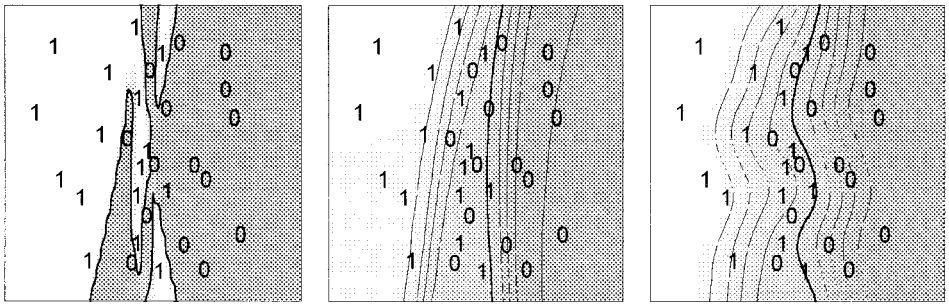

Figure 17.3
Smoothing effects of training with jitter: (a) an intentionally overtrained 2/50/10/1 feedforward network chooses an overly complex boundary and can be expected to generalize poorly; (b) the same network trained with Gaussian ($\sigma = 0.1$) input noise forms a much smoother boundary and better generalization can be expected; and (c) the expression in equation 17.5 for the expected value of the target function at an arbitrary point x.

network. With 671 weights, but only 31 training points, the network is very underconstrained and chooses a very nonlinear boundary. Training with jittered data discourages sharp changes in the response near the training points and so discourages the network from forming overly complex boundaries. Figure 17.3(b) shows the same network trained for the same amount of time from the same initial conditions with additive Gaussian input noise ($\sigma = 0.1$). Despite very long training times, the response shows no effects of overtraining. For reference, figure 17.3(c) shows the expected value of the target given the noisy input as calculated in equation 17.5.

17.3 Training with Jitter and Sigmoid Scaling

A drawback of training with jitter is that it requires the use of a small learning rate and many sample presentations in order to average over the noise. In certain special cases, the expected response of a network driven by a jittered input can be approximated by simply adjusting the sigmoid slopes. This is, of course, much faster than averaging over the noise. This result provides justification for gain scaling as a heuristic for improving generalization.

17.3.1 Linear Output Networks

Consider the function

$$y(\mathbf{x}) = \sum_k v_k \, h_k(\mathbf{x}) \tag{17.16}$$

where

$$h_k(\mathbf{x}) = g\left(w_k^T \mathbf{x} - \theta_k\right) \tag{17.17}$$

and $g(\cdot)$ is the node nonlinearity. This describes a single-hidden-layer network with a *linear* output.

With jitter (and the approximations stated for equation 17.2), the expected output for a fixed input $\mathbf{x}$ is

$$\begin{aligned} \langle y(\mathbf{x}+\mathbf{n})\rangle &\approx y(\mathbf{x}) \star p_{\mathbf{n}}(\mathbf{x}) \\ &\approx \sum_k v_k \, h_k(\mathbf{x}) \star p_{\mathbf{n}}(\mathbf{x}) \\ &\approx \sum_k v_k \left[h_k(\mathbf{x}) * p_{\mathbf{n}}(\mathbf{x})\right], \end{aligned} \tag{17.18}$$

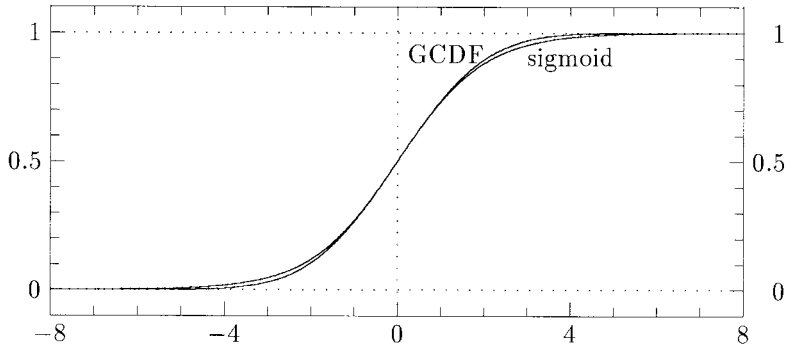

Figure 17.4
The conventional sigmoid $1/(1+e^{-x})$ and the Gaussian cumulative distribution function (GCDF) (with $\sigma = 4/\sqrt{2\pi}$) have very similar shapes and give similar results when used as the node nonlinearities. The GCDF is useful in this analysis because it is shape invariant when convolved with a spherical Gaussian noise density.

that is, a linear sum of convolutions of the hidden unit responses with the noise density. The symbol $\star$ denotes correlation, $a(x) \star b(x) = \int_{-\infty}^{+\infty} a(\tau)b(\tau - x)\, d\tau$, which is different from convolution but the operations can be interchanged here if $p_{\mathbf{n}}(\mathbf{x})$ is symmetric.

In most neural network applications, the nonlinearity is the sigmoid $g(z) = 1/(1+e^{-z})$. If, instead, we use the Gaussian cumulative distribution function (GCDF), which has a very similar shape (see figure 17.4), then the shape of the nonlinearity will be invariant to convolution with a Gaussian input noise density. That is, if we assume that the noise is zero-mean Gaussian and spherically distributed in N dimensions

$$p_{\mathbf{n}}(\mathbf{x}) = \frac{1}{\sigma_1^N (2\pi)^{N/2}} \exp\left(\frac{-\|\mathbf{x}\|^2}{2\sigma_1^2}\right) \tag{17.19}$$

(where $\|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x}$) and the g nonlinearity is the Gaussian cumulative distribution function

$$g(z) = \int_{-\infty}^{z} \frac{1}{\sigma_2\sqrt{2\pi}} \exp\left(\frac{-\tau^2}{2\sigma_2^2}\right) d\tau. \tag{17.20}$$

then the convolution in equation 17.18 can be replaced by a simple scaling operation

$$h_k(\mathbf{x}) * p_{\mathbf{n}}(\mathbf{x}) = g\left(a_k(\mathbf{w}_k^T\mathbf{x} - \theta_k)\right) \tag{17.21}$$

where a_k is a scaling constant defined below. A derivation is given in appendix C.2.

The significance of this is that when the equivalence (17.21) holds, the expected response of the network to input noise approximated by (17.18) can be computed exactly by simply scaling the hidden unit nonlinearities appropriately; we do not have to go through the time-consuming process of estimating the response by averaging over many noisy samples. That is,

$$\langle y(\mathbf{x}+\mathbf{n})\rangle \approx \sum_k v_k g\left(a_k(\mathbf{w}_k^T\mathbf{x}-\theta_k)\right) \tag{17.22}$$

where the scaling constant a_k depends on the magnitude of the weight vector $\mathbf{w}_k$ and the noise variance

$$a_k = \frac{1}{\sqrt{\|\mathbf{w}_k\|^2\sigma_1^2+1}}. \tag{17.23}$$

Note that the bias θ_k is not included in the weight vector and has no role in the computation of a_k. It is, however, scaled by a_k.

This does not say that we can train an arbitrary network without jitter and then simply scale the sigmoids to compute exactly the network that would result from training with jitter because it does not account for dynamics of training with random noise, but it does suggest similarities.

Example Figures 17.5(a) through (f) verify this scaling property. Figures 17.5(a) and (b) show the response of a network with two inputs, three GCDF hidden units, and a linear output unit. Figures 17.5(c) and (d) show the average response using spherically distributed Gaussian noise with $\sigma = 0.1$ and averaged over 2000 noisy samples per grid point. Figures 17.5(e) and (f) show the expected response computed by scaling the hidden units. The RMS error (on a 64×64 grid) between the averaged noisy response and the scaled expected response is 0.0145. The scaled expected response was computed in a few seconds; the average noisy response required hours on the same computer.

17.3.2 Relation to Weight Decay

The scaling operation is equivalent to

$$\mathbf{w} \rightarrow \frac{\mathbf{w}}{\sqrt{\|\mathbf{w}\|^2\sigma_1^2+1}}.$$

Because the denominator is not less than 1, this always reduces the magnitude of $\mathbf{w}$ or leaves it unchanged. When $\sigma_1 = 0$ (no input noise), the weights are unchanged. When

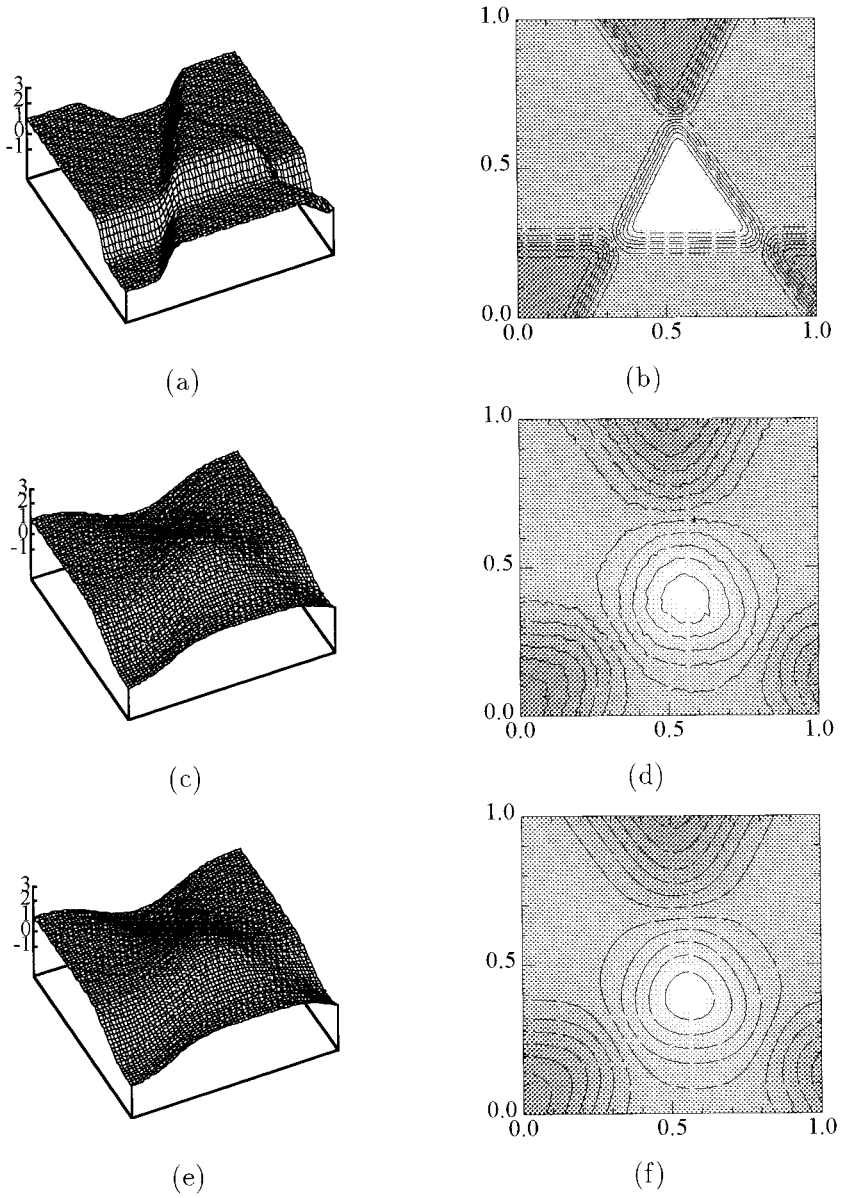

Figure 17.5
Equivalence of weight scaling and jitter averaging: (a) the transfer function of the original network and (b) its contour plot; (c) the average response with additive Gaussian input noise, $\sigma = 0.1$, averaged over 2000 noisy samples per grid point and (d) its contour plot; and (e) the expected response computed by scaling and (f) its contour plot.

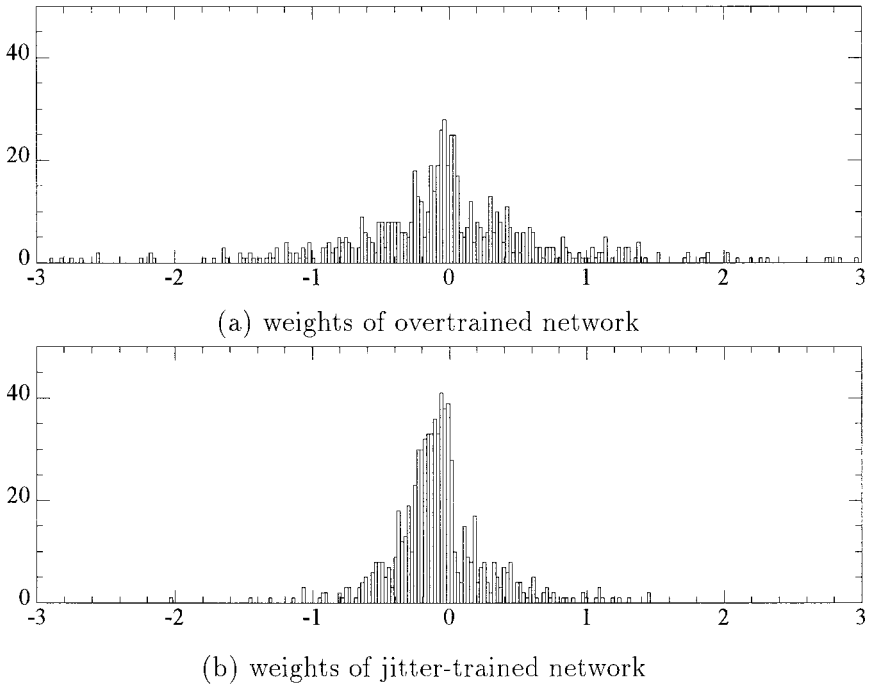

Figure 17.6
Weight-decay effects of training with jitter: (a) weights for the overtrained network of figure 17.3(a), $\sigma = 0.7262$; and (b) weights for the jitter-trained network of figure 17.3(b), $\sigma = 0.3841$.

$\sigma_1 \to \infty$, the weights approach zero. When $\|\mathbf{w}\|\sigma_1$ is small, the scaling has little effect. When $\|\mathbf{w}\|\sigma_1$ is large, the scaling is approximately

$$\mathbf{w} \to \frac{\mathbf{w}}{\|\mathbf{w}\|\sigma_1}$$

and the magnitude of $\mathbf{w}$ is reduced to approximately $1/\sigma_1$. This has some properties similar to weight decay [299, 388, 387, 308], another commonly used heuristic for improving generalization. The development of weight decay terms as a result of training single-layer linear perceptrons with input noise is shown in [167].

This is supported by figure 17.6, which shows histograms of the weights for the overtrained and jitter-trained networks of figure 17.3. Table 17.1 lists the standard deviations of the weights by layers. The jitter-trained network has smaller weight variance on all levels.

Table 17.1
Weight-decay Effects of Training with Jitter. Training with Jitter Tends to Produce Smaller Weights.

	Standard Deviations of Weights		
	overtrained network	jittered network	number of weights
In to H1 weights	1.1153	.5904	150
H1 to H2 weights	.5197	.2204	510
H2 to Out weights	1.6828	.4008	11
All weights	.7262	.3481	671

17.4 Extension to General Layered Neural Networks

The results previously discussed relating training with jittered data and regularization hold for any network. The analysis for gain scaling, however, is valid only for networks with a single hidden layer and a linear output node. More general feedforward networks have multiple layers and nonlinear output nodes. Even though the invariance property does not hold for these networks, these results lend justification to the idea of gain scaling [228, 171] and weight decay as heuristics for improving generalization.

The gain scaling analysis uses a GCDF nonlinearity in place of the usual sigmoid nonlinearity. Because these functions have similar shapes, this is not an important difference in terms of representation capability. (Differences might be observed in training dynamics, however, because the GCDF has flatter tails.) The precise form of the sigmoid is usually not important as long as it is monotonic nondecreasing; the usual sigmoid is widely used because its derivative is easily calculated.

The GCDF nonlinearity is used here because it has a convenient shape invariance property under convolution with a Gaussian input noise density. There may be other nonlinearities that, although not having this shape invariance property, are such that their expected response can still be calculated efficiently using a similar approach. If, for example, $g(x) * p_{\mathbf{n}}(x) = h(x)$, the function $h(x)$ may be different in form from $g(x)$, but still reasonably easy to calculate. As a specific example, if $g(x)$ is a step function and $p_{\mathbf{n}}(x)$ is uniform (both in one dimension), then $h(x)$ is a semilinear ramp function: 0 for $x < \alpha$, equal to x for $-\alpha \leq x \leq \alpha$, and 1 for $x > \alpha$. The expected network response can then be computed as a linear sum of $h(x)$ nonlinearities rather than a linear sum of $g(x)$ nonlinearities. Different nonlinearities must be used to calculate the normal and expected responses, but this is still much faster than averaging over many presentations of noisy samples.

The scaling results can also be applied to radial basis functions [271, 272, 300], which generally use Gaussian PDF hidden units and a linear output summation. The convolution

of two spherical Gaussian PDFs with variances σ_1^2 and σ_2^2 produces a third Gaussian PDF with variance $\sigma_3^2 = \sigma_1^2 + \sigma_2^2$, so the expected response of these networks to noise is easily calculated using similar shape invariant scaling.

17.5 Remarks

Training with jitter, error regularization, gain scaling, and weight decay are all methods that have been proposed to improve generalization. Training with small amounts of jitter approaches the generalization problem directly by assuming that slightly different inputs give approximately the same output. If the noise distribution is smooth, the network will interpolate among training points in proportion to a smooth function of the distance to each training point.

With jitter, the effective target function is a smoothed version of the discrete training set. If the training set describes the target function well, the effective target approximates a smoothed version of the actual target function. The result is similar to training with a regularized objective function favoring smooth functions and the noise variance playing the role of the regularization parameter. Where regularization works by modifying the objective function, training with jitter achieves the same result by modifying the training data. In hindsight, the fact that training with noisy data approximates regularization is not surprising because this is the sort of thing regularization was developed to address.

Although large networks generally learn rapidly, they tend to generalize poorly because of insufficient constraints. Training with jitter helps to prevent overfitting by providing additional constraints. The effective target function is a continuous function defined over the entire input space whereas the original target function may be defined only at the specific training points. This constrains the network and forces it to use any excess degrees of freedom to approximate the smoothed target function rather than forming an arbitrarily complex boundary that just happens to fit the original training data (memorization). Even though the network may be large, it models a simpler system.

The expected effect of jitter can be calculated efficiently in some cases by a simple scaling of the node gains. This suggests the possibility of a post-training step to choose optimum gains based on cross-validation with a test set. This might make it possible to improve the generalization of large networks while retaining the advantage of fast learning.

The problem of choosing an appropriate noise variance has not been addressed here. Holmström and Koistinen suggest several methods based on cross-validation. Considerable research has been done on the problem of selecting an appropriate λ for regularization, especially for linear models. Because of the relationship between training with jitter and regularization, the regularization research may be helpful in selecting an appropriate noise level.

17.6 Further Examples

17.6.1 Static Noise

The use of dynamic jitter may interfere with some training algorithms because the measured error changes from moment to moment due to the jitter. Algorithms that adapt the learning rate depending on the change in error from one iteration to the next or algorithms that use information from previous iterations to choose the next search point could become unstable. It may also be inconvenient to add dynamic jitter to the data in closed simulation systems. In these cases it may be useful to use static noise, that is, to create a larger fixed training set by adding noisy versions of the original patterns.

Figure 17.7 illustrates the effect of training with a static noisy data set. Figure 17.7(a) shows the surface learned by a 2/50/10/1 network trained on the original 31 data points (724 epochs with RProp). The 31 points are almost linearly separable, but with 671 weights the network is very underconstrained and chooses a complex decision surface with sharp transitions. A static noisy data set of 930 points was generated by perturbing each of the original points with Gaussian noise ($\sigma = 0.1$) 30 times. (Thirty was chosen to give more training patterns than weights. Simulations using 5 and 10 noisy patterns per original point yielded complex boundaries.) The original points were not included in the new training set. Figure 17.7(b) shows the surface learned by a network initialized with the same weights

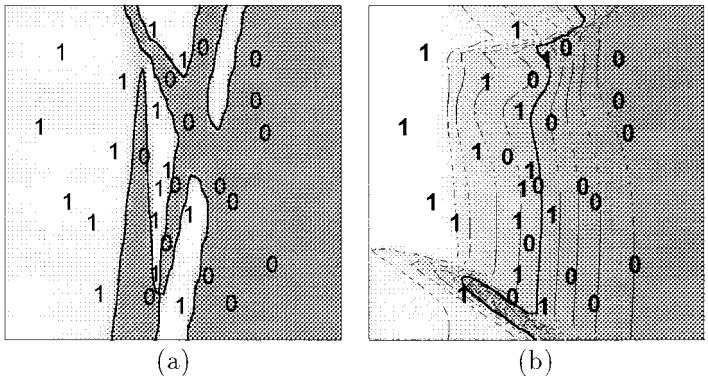

Figure 17.7
Training with static noise: (a) response of an underconstrained 2/50/10/1 net. The boundary is complex and transitions are steep, but the data is almost linearly separable. (b) Response of the same net trained on an enlarged data set obtained by replacing each original training point by 30 noisy points ($\sigma = 0.1$). The boundary is simpler and transitions are more gradual, but a few kinks remain. (1s and 0s denote the training points, the training values were 0.9 and -0.9.)

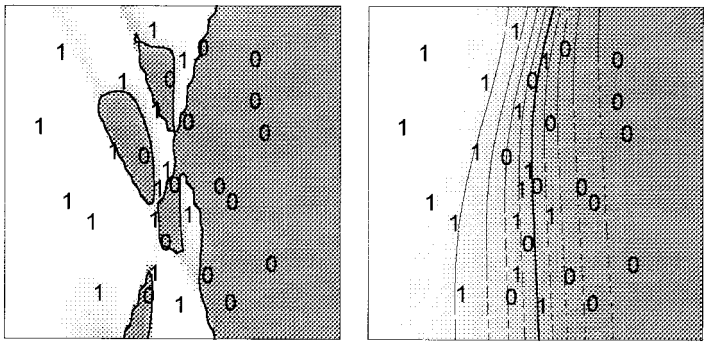

Figure 17.8
Cross-validation with jittered data. An artificial validation data set was created by generating 30 jittered points from each of the original 31 training points: (a) response of an underconstrained 2/50/10/1 net trained to convergence, and (b) response of the net with the best RMS error on the validation set (1s and 0s denote the training points, the training values were 0.9 and -0.9).

after 2500 epochs of RProp training. In most places, the decision surface is less complex and the transitions are more gradual, but a few kinks remain. Evidently the network was still able to exploit idiosyncrasies in the data so perhaps 930 points was not enough to constrain the network enough to prevent overtraining. (The second network was trained for a much longer time, however: 2,325,000 pattern presentations versus 22,444. The kinks might not have developed if the net were trained for an equivalent number of pattern presentations or an equivalent number of epochs, but this indicates that the augmented data by itself was not enough to prevent overtraining.)

17.6.2 Cross-Validation with Jittered Data

An artificial validation data set was created by generating 30 jittered points from each of the original 31 training points. Figure 17.8(a) shows the response of the network trained to convergence. Overfitting is obvious. Figure 17.8(b) shows the response of the network with the best validation error. Final convergence of the overtrained net occurred at 1365 epochs. The best validation was observed at 165 epochs.

More sophisticated versions of this approach are described by Musavi et al. [281] and Pados and Papantoni-Kazakos [293]. In both, the joint density $f(X, Y)$ is estimated by fitting Gaussians (not necessarily spherical) around each point. This is done by a radial basis function network in [293]. The resulting density estimate can then be used to estimate the generalization error of another network or, as in section 17.6.1, to generate a larger set of artificial training data.

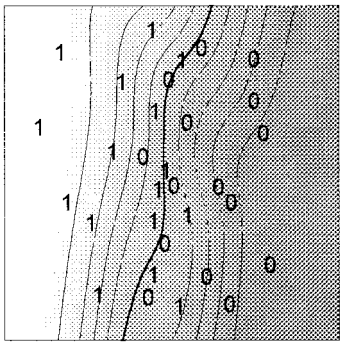

Figure 17.9
Smoothing an overtrained response. Given an overtrained net, a better estimate of the true function at a point **x** might be obtained by averaging a number of probes around **x** using a noisy input. The figure shows the expected response of the network in figure 17.7(a) to a noisy input ($\sigma = 0.1$) (1s and 0s denote the training points, the training values were 0.9 and -0.9).

17.6.3 Jitter Used to Discount an Overtrained Response

When system response time is not a critical consideration, averaging with jitter might be used to smooth the output of an overtrained network to obtain a more reliable response. That is, given an overtrained net, a better estimate of the true function at a point **x** might be obtained by averaging a number of probes around **x** using a noisy input

$$y' = \frac{1}{N} \sum_{k=1}^{N} y(\mathbf{x} + \mathbf{n}).$$

Figure 17.9 shows the expected response of the overtrained network in figure 17.7(a) to a noisy input ($\sigma = 0.1$). Unlike training with dynamic jitter, which slows training by requiring a small learning rate, or training with static jitter, which slows training by increasing the size of the training set, this allows fast training but mitigates the worst effects of overtraining at the expense of a slightly slower response during recall.