Communicated by Joshua Alspector

The Effects of Adding Noise During Backpropagation Training on a Generalization Performance

Guozhong An
Shell Research, P.O. Box 60, 2280 AB Rijswijk, The Netherlands

We study the effects of adding noise to the inputs, outputs, weight connections, and weight changes of multilayer feedforward neural networks during backpropagation training. We rigorously derive and analyze the objective functions that are minimized by the noise-affected training processes. We show that input noise and weight noise encourage the neural-network output to be a smooth function of the input or its weights, respectively. In the weak-noise limit, noise added to the output of the neural networks only changes the objective function by a constant. Hence, it cannot improve generalization. Input noise introduces penalty terms in the objective function that are related to, but distinct from, those found in the regularization approaches. Simulations have been performed on a regression and a classification problem to further substantiate our analysis. Input noise is found to be effective in improving the generalization performance for both problems. However, weight noise is found to be effective in improving the generalization performance only for the classification problem. Other forms of noise have practically no effect on generalization.

1 Introduction

A neural network that is determined solely on the basis of its training set often does not give satisfactory results when applied to new data. This is the problem of generalization, or of model selection in the context of data modeling.

There are a number of approaches to solve the problem. The validation-set method is one such approach. It uses an extra set of data, the validation set, to select a neural network. On the basis of its performance on the validation set, a neural network is selected from those neural networks that have equivalent training-set performance. The shortcoming of the validation-set approach is that it is effective only if both the training set and the validation set are large and representative. Otherwise, the neural network selected will be biased by the validation and the training set, since the selected network has indirectly adapted itself to the validation set. A secondary deficiency of the validation-set approach is that it does not provide any strategy for finding the desired model.

Regularization is another established method that addresses the problem of generalization (e.g., Poggio and Girosi 1990; Weigend *et al.* 1991; Guyon *et al.* 1992; Krogh and Hertz 1992). In the regularization approach, one adds a penalty term to the objective function of training. The penalty term serves as a constraint on the possible models. The success of the regularization approach depends on the specific form of the penalty term that is used and on the degree to which the penalty-term constraint is consistent with the underlying relation being sought. The weakness of this method is that the type of regularization is problem dependent; it is often not clear what form of regularization to use a priori.

Recently, it has been observed that injecting noise to various parts of the neural network during backpropagation training can improve the generalization performance remarkably (e.g., Sietsma and Dow 1988; Weigend *et al.* 1991; Hanson 1990; Clay and Sequin 1992; NeuralWare 1991; Murray and Edwards 1993; Rögnvaldsson 1994). Most forms of introducing noise that are found in the literature belong to one of the following classes: (1) data noise that is added to the inputs and outputs of the neural network (e.g., Sietsma and Dow 1988; Weigend *et al.* 1991), (2) weight noise that is added to the weights of the neural networks (Hanson 1990; NeuralWare 1991; Clay and Sequin 1992; Murray and Edwards 1993; Hinton and van Camp 1993), and (3) Langevin noise that is added to the weight changes (Rögnvaldsson 1994). In this paper, we investigate the mechanisms that have led to the observed better generalization performance. Using results from stochastic optimization and statistical mechanics, we derive and analyze the cost functions that are minimized by the posttraining weight vectors. In particular, we aim to determine the conditions under which each type of noise addition will contribute positively to the generalization performance. We have treated output noise and input noise on an equal footing. Both mean-square and cross-entropy error functions are considered. Our analysis of the weight noise is limited to the mean-square error function. The conclusions we reached for the Langevin noise do not depend on the specific form of the error function. Simulations have been performed on a regression and a classification problem to verify our theoretical predictions.

Input noise that is added to the inputs of neural networks has been previously studied with the mean-square error function. Interpreting the addition of noise to the inputs as generating additional training data, Holmström and Koistinen (1992) showed that the method is asymptotically consistent, i.e., as the number of training examples increases to infinity and the variance of the noise decreases to zero, training with input noise is equivalent to minimizing the true error function. Closer to the spirit of our work is that of Reed *et al.* (1992), Matsuoka (1992), and Bishop (1995). Without detailed analysis, Reed *et al.* and Bishop assume that backpropagation training with input noise converges to a kind of average of a stochastic error function. On the basis of a Taylor expansion of the expected error function, they concluded that training with input noise is equivalent to a form of regularization. Matsuoka proposes to minimize an objective function that is formed by adding a sensitivity measure to the standard error function. He then asserts that this objective function is minimized by the backpropagation algorithm with input noise.

Using a recently proved convergence theorem on stochastic gradient descent, we have rigorously derived the objective function that is minimized by the backpropagation training with input noise. It takes the form of the expectation of the noise-infected error function over the noise distribution. Performing a careful analysis of this objective function, we are led to contradict Reed (1992), Matsuoka (1992), and Bishop (1995). Even in the weak-noise limit, training with input noise is not equivalent to a regularization method. The difference lies in a noise-induced term in the objective function that depends on the fitting residues. This contribution to the objective function escaped the notice of Reed *et al.* and Matsuoka. Although the residue-dependent contribution was found by Bishop (1995), he concluded that such a term vanishes at the end of training, which is true only in the case of an infinite training set. In our simulation, we found that the importance of this term is comparable to that of the regularization term. We do not dispute that the main mechanism for the improved generalization of input noise is due to the noise's smoothing effect. We merely point out that input noise in fact smoothes the neural network function in a way that is different from that of the regularization approach.

Murray and Edwards (1994) attempted to analyze the improvement of generalization that they observed on two classification problems. On the basis of a heuristic analysis and the results of their case studies, they believed that adding weight noise to all training algorithms should, in general, have a positive effect on generalization. By treating weight noise as a special type of noise added to the output of the neural network, we are able to rigorously analyze the addition of zero-mean and constant variance gaussian and uniform noise to the weights during on-line backpropagation training. Our theoretical analysis and simulation results, however, do not demonstrate that weight noise improves generalization in all cases.

In the following section, we introduce the definitions and notations that we rely on in the main sections of this paper. The effects of the data noise and the weight noise are respectively analyzed in Sections 3 and 4. In Section 5, we study the Langevin noise. Two illustrative examples are presented in Section 6 to highlight the results obtained in Sections 3–5. We present our conclusions in Section 7.

2 Definitions

We consider multilayer feed-forward neural networks with input vector $\mathbf{x}$ and, without loss of generality, a scalar output $f(\mathbf{x}, \mathbf{w})$. Let a_i be the output and θ_i the bias of the ith neuron, w_{ij} the weight connection from neuron j to neuron i, and $h(t)$ the transfer function. At the input layer, we have $a_i = x_i$. For the hidden and output layers, we have

$$a_i = h\left(\sum_j w_{ij}a_j + \theta_i\right) \tag{2.1}$$

We shall refer to the w_{ij}s and θ_is collectively as weights, and denote them by the vector $\mathbf{w}$. Denote the training set that consists of N input–output pairs by $\{z^\mu \equiv (\mathbf{x}^\mu, y^\mu) : \mu = 1, 2, \ldots, N\}$. Let $e(z^\mu, \mathbf{w})$ be the error function for the μth training example. Let

$$E(\mathbf{w}) \equiv \frac{1}{N}\sum_{\mu=1}^{N} e(z^\mu, \mathbf{w}) \tag{2.2}$$

be the total error function of the training set.

The normal backpropagation training algorithm (Rumelhart *et al.* 1986) and its on-line version are both described by the following weight-update equation:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta\mathbf{w}_t \tag{2.3}$$

where $\mathbf{w}_t$ is the weight value and $\Delta\mathbf{w}_t$ the weight change at iteration t. For the normal backpropagation training, the weight change at iteration t is given by $\Delta\mathbf{w}_t = -\eta_t \nabla E(\mathbf{w})$, where η_t is the learning rate. In the case of the on-line backpropagation training algorithm,

$$\Delta\mathbf{w}_t = -\eta_t \nabla e(z, \mathbf{w}) \tag{2.4}$$

where z is randomly drawn from the training set $\{z^1, z^2, \ldots, z^\mu, \ldots, z^N\}$ at each iteration. The normal backpropagation training minimizes the error function $E(\mathbf{w})$ by steepest descent. Under appropriate conditions, it can be shown that the on-line backpropagation algorithm minimizes the error function $E(\mathbf{w})$ by the stochastic gradient descent (White 1989).

The three types of noise mentioned in the previous section are described by the following substitution rules in the computation of the weight changes $\Delta\mathbf{w}$:

1. data noise $z^\mu \longrightarrow z^\mu + \zeta$ (2.5)
2. weight noise $\mathbf{w} \longrightarrow \mathbf{w} + \xi$ (2.6)
3. Langevin noise $\Delta\mathbf{w} \longrightarrow \Delta\mathbf{w} + \xi$ (2.7)

Here, ζ denotes a noise vector that has the same dimensions as z^μ, whereas ξ has the same dimensions as $\mathbf{w}$.

3 Training with Data Noise

3.1 The Cost Function. The following major steps are involved in injecting data noise into the on-line backpropagation training algorithm:

1. Select a training example $z^\mu = (\mathbf{x}^\mu, y^\mu)$ randomly out of the N training examples.
2. Draw a sample noise vector ζ^μ from a density $\rho(\zeta^\mu)$.
3. Set $z = z^\mu + \zeta^\mu$ in equation 2.4.

With this procedure, the probability density of generating a particular data point $z = (\mathbf{x}, y)$ from the μth training example z^μ is thus $\rho(\zeta^\mu) = \rho(z - z^\mu)$. The total probability density of z being generated from the complete training set is then

$$\hat{g}(z) = \frac{1}{N}\sum_{\mu=1}^{N} \rho(z - z^\mu) \tag{3.1}$$

Let $c(\mathbf{u}, \mathbf{w})$ be a continuous and differentiable function of $\mathbf{w}$, and let $\mathbf{u}$ be a random vector sampled from the distribution $g(\mathbf{u})$. Under general conditions, Bottou (1991) has shown that stochastic gradient descent, i.e., the following iteration

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} c(\mathbf{u}, \mathbf{w}) \tag{3.2}$$

converges to a minimum of the expectation values of $c(\mathbf{u}, \mathbf{w})$, i.e., to

$$C(\mathbf{w}) = \int c(\mathbf{u}, \mathbf{w}) g(\mathbf{u}) d\mathbf{u} \tag{3.3}$$

provided that $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$. The on-line backpropagation training algorithm is a special case of the stochastic gradient descent method in which the function c is given by the error function $e(z, \mathbf{w})$, and the data density g is given by

$$g_t(z) \equiv \sum_\mu \frac{1}{N} \delta(z - z^\mu) \tag{3.4}$$

The delta function δ in equation 3.4 is the Dirac delta function. It follows from equation 3.3 that the on-line backpropagation training converges to a minimum of $E(\mathbf{w})$. Combining equation 2.4 with equations 3.1–3.3, it follows that training with data noise minimizes not $E(\mathbf{w})$ but

$$\mathcal{E}(\mathbf{w}) = \int e(z, \mathbf{w}) \hat{g}(z)\, dz \tag{3.5}$$

In passing, we remark that equation 3.1 can be viewed as a kernel estimation of the true density of the data based on the training set (Holmström and Koistinen 1992). The distribution of the noise serves as the kernel function.

The main difference between $E(\mathbf{w})$ and $\mathcal{E}(\mathbf{w})$ is that the former measures the error over a finite number of training examples, whereas the latter measures the error over an infinite data set that is described by the continuous density $\tilde{g}(z)$. Those data that are described by $\tilde{g}(z)$ but are not contained in the training set can be considered as synthetic data generated by the noise. Such synthetic data supplement the training set and dilute the importance of an individual example contained in the training set. Thus, noise injection could prevent the neural network from overfitting the training set and may result in neural networks that are insensitive to noise in the data.

We see that training with data noise effectively minimizes a cost function that differs from the standard error function $E(\mathbf{w})$; the injected noise implicitly alters the training objective function. There are a number of methods that explicitly modify the cost function to improve generalization. A widely used method to construct a new cost function $C(\mathbf{w})$ is to add a penalty function $\mathcal{P}(\mathbf{w})$ to the standard training error $E(\mathbf{w})$:

$$C(\mathbf{w}) = E(\mathbf{w}) + \lambda\mathcal{P}(\mathbf{w}) \tag{3.6}$$

where λ is a positive constant. Training is then achieved by minimizing $C(\mathbf{w})$. It is a standard result of variational calculus (see, e.g., Arfken 1985) that the vector that minimizes $C(\mathbf{w})$ also minimizes $E(\mathbf{w})$ under a constraint of the form

$$\mathcal{P}(\mathbf{w}) = \text{constant} \tag{3.7}$$

where the value of the constant depends on λ. The specific form of the penalty function depends on one's assumptions. In intuitive approaches, such as weight decay (Hinton 1986) and weight elimination (Weigend *et al.* 1991), $\mathcal{P}$ represents an intuitive measure of the size of the neural network. In the maximum a posteriori probability approach, $\mathcal{P}(\mathbf{w})$ represents the negative of the logarithm of the a priori probability of finding a set of weight $\mathbf{w}$ (MacKay 1992; Nowlan and Hinton 1992). In an information theoretical approach, such as the minimum description length approach, $\mathcal{P}(\mathbf{w})$ would represent the description length of the model (Rissanen 1989; Kendall and Hall 1993). In the regularization approach as applied to inverse problems (Poggio and Girosi 1990), $\mathcal{P}(\mathbf{w})$ represents a smoothness measure.

Minimizing $C(\mathbf{w})$ can lead to better generalization than does minimizing $E(\mathbf{w})$, but only if $C(\mathbf{w})$ contains a constraint that is consistent with the underlying data-generating process. To gain more insight into $\mathcal{E}$, and to make connections with other approaches, we compute the difference between $\mathcal{E}$ and the standard error E. This difference corresponds to the penalty function of equation 3.6.

3.2 Penalty Function Induced by the Data Noise. Let us define a penalty function P by

$$\mathcal{E}(\mathbf{w}) \equiv E(\mathbf{w}) + P(\mathbf{w}) \tag{3.8}$$

Combining equations 3.1, 3.5, and 3.8 (and for brevity suppressing the argument $\mathbf{w}$ in e), we have

$$P = \frac{1}{N}\sum_{\mu}\left[\int e(z)\rho(z - z^{\mu})dz - e(z^{\mu})\right] \tag{3.9}$$

Replacing the integration variable z by $\zeta = z - z^{\mu}$, we obtain

$$P = \frac{1}{N}\sum_{\mu}\left[\int e(z^{\mu} + \zeta)\rho(\zeta)d\zeta - e(z^{\mu})\right] \tag{3.10}$$

Denoting the average over the noise distribution $\rho(\zeta)$ by $\langle\,\rangle_{\zeta}$, we can rewrite P as

$$P = \frac{1}{N}\sum_{\mu}[\langle e(z^{\mu} + \zeta)\rangle_{\zeta} - e(z^{\mu})] \tag{3.11}$$

Expanding $e(z + \zeta)$ as a Taylor series in ζ, we have

$$e(z + \zeta) = e(z) + \frac{\partial e(z)}{\partial z_i}\zeta_i + \frac{1}{2}\frac{\partial^2 e(z)}{\partial z_i \partial z_j}\zeta_i\zeta_j + \cdots + \frac{1}{n!}(\zeta_i\partial_i)^n e(z) \tag{3.12}$$

where ∂_i denotes the partial derivative with respect to z_i, and summation over repeated Latin indices is implied. To proceed further, we assume the following: (1) different components of the noise are independent; (2) the noise distribution is symmetric about zero, which implies that all the odd-order moments, including the mean, vanish. Stated mathematically, $\langle\zeta_i^n\rangle = 0$ when n is odd. With these conditions, we have

$$\langle e(z + \zeta)\rangle - e(z) = \sum_i T_i \frac{\partial^2 e(z)}{\partial^2 z_i} + R \tag{3.13}$$

where $2T_i$ represents the variance of the ith component of the noise, i.e., $\langle\zeta_i^2\rangle = 2T_i$, and R denotes the remainder. The remainder R is given by

$$R = \sum_{n=3}\frac{1}{n!}\sum{}' \frac{n!}{n_1!n_2!\cdots n_m!}\langle(\zeta_1\partial_1)^{n_1}\rangle\langle(\zeta_2\partial_2)^{n_2}\rangle\cdots\langle(\zeta_m\partial_m)^{n_m}\rangle e(z) \tag{3.14}$$

where m is the dimension of z. The summation $\sum'$ is over all possible combinations of n_i that satisfy $\sum_i n_i = n$. Assuming that the remainder R is negligible compared to the first term, we have

$$\langle e(z + \zeta)\rangle - e(z) \approx \sum_i T_i \frac{\partial^2 e(z)}{\partial^2 z_i} \tag{3.15}$$

Replacing z by z^μ in equation 3.15 and substituting it into equation 3.11, we obtain

$$P \approx \frac{1}{N} \sum_{\mu,i} T_i \frac{\partial^2 e(z^\mu)}{\partial^2 z_i^\mu} \tag{3.16}$$

In the weak-noise limit, i.e., when $T_i \ll 1$, we can show that R is indeed negligible for two popular noise distributions: the gaussian distribution and the uniform distribution. We have, for the uniform distribution that is constant within an interval $[-a, a]$ and zero outside,

$$\langle \zeta_i^n \rangle = \begin{cases} \frac{a^n}{n+1} & \text{if } n \text{ is even} \\ 0 & \text{if } n \text{ is odd} \end{cases} \tag{3.17}$$

In the case of gaussian noise (see, e.g., Kendall and Stuart 1977), the moments $\langle \zeta_i^n \rangle$ are given by

$$\langle \zeta_i^n \rangle = \begin{cases} \frac{n!}{(n/2)!} T_i^{n/2} & \text{if } n \text{ is even} \\ 0 & \text{if } n \text{ is odd} \end{cases} \tag{3.18}$$

Using the above expressions for the moments, we can evaluate the higher-order terms contained in R. It is evident that they are of higher orders in T, and hence negligible in the limit $T \ll 1$.

Defining a Hessian matrix H by $H_{ij}(z) \equiv \partial^2 e(z)/\partial z_i \partial z_j$, and for simplicity assuming $T_i = T$, we can rewrite equation 3.16 in a more compact form as

$$P \approx \frac{T}{N} \sum_{\mu} \mathrm{Tr} H(z^\mu) \tag{3.19}$$

where Tr is the trace operator.

Next, we apply equation 3.16 to the two most popular forms of the error functions, i.e., the quadratic and cross-entropy error functions, which are given, respectively, by

$$e_q(z, \mathbf{w}) \equiv \frac{1}{2} [y - f(\mathbf{x}, \mathbf{w})]^2 \tag{3.20}$$

and

$$e_c(z, \mathbf{w}) \equiv y \log \frac{y}{f(\mathbf{x}, \mathbf{w})} + (1 - y) \log \frac{1 - y}{1 - f(\mathbf{x}, \mathbf{w})} \tag{3.21}$$

Substituting the quadratic error function of equation 3.20 into equation 3.16 and denoting $f(\mathbf{x}^\mu, \mathbf{w})$ by f_μ, we have, after some algebraic manipulation,

$$P \approx T(\mathcal{P}_0 + \mathcal{P}_1 + \mathcal{P}_2) \tag{3.22}$$

where

$$\mathcal{P}_0 = 1 \tag{3.23}$$

$$\mathcal{P}_1 = \frac{1}{N}\sum_{\mu,i}\left|\frac{\partial f_\mu}{\partial x_i^\mu}\right|^2 \tag{3.24}$$

$$\mathcal{P}_2 = \frac{1}{N}\sum_{\mu,i}[f_\mu - y^\mu]\frac{\partial^2 f_\mu}{\partial^2 x_i^\mu} \tag{3.25}$$

For simplicity, we have assumed here again that the strength of the noise is T for all components. If we take the cross-entropy error function of equation 3.21 instead of the quadratic error function, the penalty function again takes the form of equation 3.22 with

$$\mathcal{P}_0 = \frac{1}{N}\sum_{\mu}[y^\mu(1-y^\mu)]^{-1} \tag{3.26}$$

$$\mathcal{P}_1 = \frac{1}{N}\sum_{\mu,i}\frac{f_\mu^2 - 2y^\mu f_\mu + y^\mu}{f_\mu^2(1-f_\mu)^2}\left|\frac{\partial f_\mu}{\partial x_i^\mu}\right|^2 \tag{3.27}$$

$$\mathcal{P}_2 = \frac{1}{N}\sum_{\mu,i}\frac{f_\mu - y^\mu}{f_\mu(1-f_\mu)}\frac{\partial^2 f_\mu}{\partial^2 x_i^\mu} \tag{3.28}$$

In line with our assumption, each component of the noise contributes to P independently as can be seen from equation 3.16. The $\mathcal{P}_0$ term in equation 3.22 is due to the noise on the desired output; the $\mathcal{P}_1$ and $\mathcal{P}_2$ terms are due to noise in the input.

Because the penalty induced by noise on the desired output, $\mathcal{P}_0$, does not depend on $\mathbf{w}$, it has no influence on the minimum of the cost function. Therefore, noise in the desired output does not influence the neural-network function $f(\mathbf{x}, \mathbf{w})$ obtained at the end of a training procedure. *Zero-mean and constant-variance noise added to the desired output thus have no effect on generalization.*

In contrast to $\mathcal{P}_0$, the penalty terms induced by the input noise, $\mathcal{P}_1$ and $\mathcal{P}_2$, are $\mathbf{w}$ dependent. The term $\mathcal{P}_1$ of equation 3.24 is obviously positive. Because the cross-entropy error function is only used for classification problems for which y, $f(\mathbf{x}, \mathbf{w}) \in [0, 1]$, it is easy to show that $\mathcal{P}_1$ of equation 3.27 is also positive. The $\mathcal{P}_1$ terms favor a slow-varying input–output mapping $f(\mathbf{x}, \mathbf{w})$ and penalize fast varying ones. Reed *et al.* (1992) and Matsuoka (1992) previously obtained the $\mathcal{P}_1$ term of equation 3.24. However, owing to an inconsistency in their approaches, they failed to find the $\mathcal{P}_2$ term. Bishop (1995) also obtained the $\mathcal{P}_2$ term, but he concluded that the $\mathcal{P}_2$ term vanishes. However, his argument crucially depends on the fact that the conditional probability $p(y \mid \mathbf{x})$ of a target output y given an input $\mathbf{x}$, is known at an *arbitrary* input $\mathbf{x}$, which is true only in the case of an infinite training set. His conclusion therefore does not apply to problems for which one has only a finite training set. The penalty term $\mathcal{P}_1$ induced by the input noise closely resembles the penalty terms that are commonly used in the regularization methods (Poggio and Girosi 1990). In particular, the $\mathcal{P}_1$ terms of equation 3.24 and equation 3.27 take the form of $||Lf||^2$ where L is a differentiation operator and $|| \cdot ||$ represents a norm in a function space. $\mathcal{P}_2$ depends both on the fitting residues $f(\mathbf{x}.\mathbf{w}) - y$ and the second-order derivatives of f with respect to $\mathbf{x}$; it can, in general, be either positive or negative. It is the $\mathcal{P}_2$ term that distinguishes training with input noise from a regularization approach.

In general, the importance of $\mathcal{P}_1$ should outweigh that of $\mathcal{P}_2$ whenever the fitting residues are reasonably small or the function is very smooth. In this case, the penalty term induced by the input noise measures the sensitivity of the neural-network output against a small variation in the input. The noise strength balances the task of fitting the training examples on the one hand and the task of not overfitting them on the other hand. Noise in the inputs hence constrains the neural-network output to be a smooth function of its input.

It is worthwhile to point out that Drucker and Le Cun (1992) recently proposed a procedure by the name "double backpropagation" to minimize the sum of E and the $\mathcal{P}_1$ term in equation 3.22. It is thus related to on-line learning with input noise in the case where the variance of the noise is small and the neural network has learned sufficiently from the training set.

4 Training with Weight Noise

We have seen that on-line backpropagation training with data noise is equivalent to enlarging the training set with synthetic data. The added synthetic data force the neural network function to be less sensitive to input variations. Therefore, they avoid overfitting and possibly improve generalization. In this section, we study the effect of weight noise on generalization with particular reference to the quadratic error function.

The convergence property of training with weight noise turns out to be difficult to analyze. In the following, we bypass the converge analysis and directly compute the weight-noise induced penalty terms using equation 3.11. The justification for this is that weight noise, in the weak limit, can be treated as a type of output noise, albeit unusual.

Weight noise, like data noise, affects the $\Delta\mathbf{w}$ in the weight-update equation only through the neural-network output function $f(\mathbf{x}, \mathbf{w})$ and its derivatives with respect to $\mathbf{w}$. The neural-network output $f(\mathbf{x}, \mathbf{w})$ appears in the quadratic error function only in the form of $y^\mu - f(\mathbf{x}^\mu, \mathbf{w})$. Hence, any change of $f(\mathbf{x}^\mu.\mathbf{w})$ can always be replaced by a change of y^μ. In this way, we can treat the weight noise as a special type of noise in the desired output.

In the presence of weight noise ξ, the neural-network output is given by $f(\mathbf{x}, \mathbf{w} + \xi)$. Expanding $f(\mathbf{x}, \mathbf{w} + \xi)$ as a power series in ξ, we obtain

$$f(\mathbf{x}, \mathbf{w} + \xi) = f(\mathbf{x}, \mathbf{w}) + \frac{\partial f(\mathbf{x}, \mathbf{w})}{\partial w_i}\xi_i + \frac{1}{2}\frac{\partial^2 f(\mathbf{x}, \mathbf{w})}{\partial w_i \partial w_j}\xi_i\xi_j + \cdots \tag{4.1}$$

where summation over repeated indices is implied. Denoting by ζ_o the noise in the desired output that is induced by the weight noise (and using again the implied summation convention), we have

$$\zeta_o = \frac{\partial f(\mathbf{x}, \mathbf{w})}{\partial w_i}\xi_i + \frac{1}{2}\frac{\partial^2 f(\mathbf{x}, \mathbf{w})}{\partial w_i \partial w_j}\xi_i\xi_j + \cdots \tag{4.2}$$

Assuming the different components of the noise are identically independently distributed, and have a zero mean and variance $2T_i$, we obtain to the leading order in T_i,

$$\langle\zeta_o\rangle = \sum_i T_i \frac{\partial^2 f(\mathbf{x}, \mathbf{w})}{\partial^2 w_i} + O(T^2) \tag{4.3}$$

$$\langle\zeta_o^2\rangle = 2\sum_i T_i \left|\frac{\partial f(\mathbf{x}, \mathbf{w})}{\partial w_i}\right|^2 + O(T^2) \tag{4.4}$$

For gaussian noise and uniform noise, it can be shown (much as we did in the previous section) that the terms that are neglected in computing $\langle\zeta_o\rangle$ and $\langle\zeta_o^2\rangle$ are indeed of higher order in T. Repeating the steps in the calculations leading from equation 3.11 to equation 3.16 and noting that ζ_o now has a nonzero mean, we obtain an induced penalty term

$$P = \frac{1}{N}\sum_{\mu,i} T_i \frac{\partial^2 f(\mathbf{x}^\mu, \mathbf{w})}{\partial^2 w_i}\frac{\partial e(z^\mu)}{\partial y^\mu} + \frac{1}{N}\sum_{\mu,i} T_i \left|\frac{\partial f(\mathbf{x}^\mu, \mathbf{w})}{\partial w_i}\right|^2 \frac{\partial^2 e(z^\mu)}{\partial^2 y^\mu} \tag{4.5}$$

In deriving equation 4.5 we implicitly assumed that all the higher moments of ζ_o, i.e, $\{\langle\zeta_o^n\rangle : n > 2\}$, are negligible compared to the mean and variance of ζ_o in the limit $T \ll 1$. This can be verified for the first few higher moments; however, we do not have a general proof. Substituting equation 3.20 into equation 4.5, and for brevity denoting $f(\mathbf{x}^\mu, \mathbf{w})$ by f_μ, we have

$$P = \frac{T}{N}\sum_{\mu,i} \frac{\partial^2 f_\mu}{\partial^2 w_i}[y^\mu - f_\mu] + \frac{T}{N}\sum_{\mu,i}\left|\frac{\partial f_\mu}{\partial w_i}\right|^2 \tag{4.6}$$

The first term of equation 4.6 depends on the fitting residues. It does not have a definite sign. The second term in equation 4.6 is clearly always positive. In the case of relative small fitting residues, the positive definite term outweighs the other term in equation 4.6. In such a case, the weight noise penalizes weight configurations leading to neural networks that have large sensitivity with respect to weight variations. Therefore, training with weight noise should increase the tolerance of neural networks to faults in the weight connections (Clay and Sequin 1992; Murray and Edwards 1994). The fault-tolerance property is particularly relevant to the hardware implementation of neural-network weights using analog circuits in which one must deal with possible electronic noise.

Note the similarity between equation 4.6 and equations 3.24 and 3.25. This can be understood from the fact that both the input noise and the weight noise propagate through the neural network, affecting the $\Delta\mathbf{w}$ through the neural-network output function $f(\mathbf{x}, \mathbf{w})$ in analogous ways. Despite such formal similarity between input noise and weight noise, their effects on the generalization can be rather different. Introducing a roughness penalty with respect to input variations is familiar to those following the regularization approach. However, applying a roughness penalty with respect to parameter (weight) variations is less usual from the viewpoint of regularization, since it is the input–output mapping, not the weight-output mapping, that directly determines the generalization performance. In particular, smoothness in the weight-output mapping is not equivalent to smoothness in the input–output mapping.

To illustrate the above points, let us consider a neural network that has a linear output unit and no hidden layers. Denoting the weights by $\mathbf{w}$ and with the convention $x_0 \equiv 1$, we have $f(\mathbf{x}, \mathbf{w}) = \mathbf{x} \cdot \mathbf{w}$. According to equations 3.24 and 4.6, the penalty terms that are induced, respectively, by the input noise and the weight noise take the following form:

$$P = \begin{cases} T|\mathbf{w}|^2 & \text{input noise} \\ T\sum_\mu |\mathbf{x}^\mu|^2/N & \text{weight noise} \end{cases} \tag{4.7}$$

The input-noise penalty term in equation 4.7 coincides with that used in weight decay and in ridge regression. In contrast to the input-noise penalty, the weight-noise penalty term is a constant; hence, it has no effect on the generalization performance.

To infer how weight noise influences the input–output mapping and hence the generalization performance of a network with hidden layers, we need to compute the penalty terms of equation 4.6 in more detail. Noticing that each component of the noise contributes to the penalty P of equation 4.6 additively, their contributions may be separately computed. The weight-noise penalty can thus be expressed as a sum $P_o + P_h$, where P_o represents the contribution from output-layer noise (noise added to the fan-in weights of the output layer) and P_h the contribution from hidden-layer noise (noise added to the fan-in weights of the hidden layers). In the following, we compute P_o and P_h.

4.1 Weight Noise Added to the Output Layer. Consider now an output unit. Let ω_0 be its bias and ω_j be the weight connecting it to the jth hidden unit. Let I be its sum of inputs including the bias and $h(x)$ its transfer function. For the output unit, we have $f(\mathbf{x}, \mathbf{w}) = h(I)$, where

$I = \sum_j a_j \omega_j$ and $a_0 \equiv 1$. We thus have

$$\frac{\partial f(\mathbf{x}, \mathbf{w})}{\partial \omega_j} = h'(I) a_j \tag{4.8}$$

where the prime denotes differentiation with respect to I. Notice that the outputs of the hidden neurons, the a_js, do not depend on the weights of the hidden layer, the ω_js. A second differentiation therefore leads to

$$\frac{\partial^2 f(\mathbf{x}, \mathbf{w})}{\partial^2 \omega_j} = h''(I) |a_j|^2 \tag{4.9}$$

Substituting equations 4.8 and 4.9 into equation 4.6, we obtain

$$P_o = \frac{T}{N} \sum_{j,\mu} |a_j(z^\mu)|^2 \left[h''(I)(y^\mu - f_\mu) + h'(I)^2 \right] \tag{4.10}$$

where we have made the dependence of a_j on the training patterns explicit. The $j = 0$ term in equation 4.10 is due to the output-bias noise. Assuming $|h''(I)(y^\mu - f_\mu)| < h'(I)^2$, it follows from equation 4.10 that the output-layer noise favors *small activations at the hidden units* and *small derivatives at the output unit.*

The penalty term in equation 4.10 charges each hidden unit a penalty in proportion to the square of its activation $|a_j|^2$. A hidden unit with a very small activation a_j would contribute little to the network output. In this way, weight noise limits the number of hidden units that can be used to fit the training data. Equation 4.10 further shows that output-layer noise favors neural networks that have small output derivatives $|h'(I)|$. For a sigmoidal output unit [$h(I) = \tanh(I)$ or logistic], the output derivative is smallest when the output unit operates close to its two saturation states, i.e., $|I| \gg 1$. The desire to have large I (small output derivatives), however, is not in harmony with the desire to have small hidden-layer activations, since $I = \sum_j a_j \omega_j$. If the output unit is linear, which is often the case for regression problems, the derivative factor is irrelevant since $h'(I) = 1$ and $h''(I) = 0$. The only effect of the output-layer noise is then to reduce hidden-layer activations. In such a case, the output-bias noise contributes a constant term T to P_o, and has no effect on generalization.

4.2 Weight Noise Added to the Hidden Layer. For simplicity, we consider neural networks that have only one hidden layer. The extension to more hidden layers is straightforward. Denote the weight connection between the ith input unit and the jth hidden unit by w_{ji}. In a way similar to that in which the error gradient $\partial e(\mathbf{w})/\partial \mathbf{w}$ in the backpropagation algorithm (Rumelhart *et al.* 1986) is computed using the chain rule of differentiation, we have

$$\frac{\partial f(\mathbf{x}, \mathbf{w})}{\partial w_{ji}} = h'(I) H'(I_j) \omega_j x_i \tag{4.11}$$

where $H(x)$ is the transfer function in the hidden layer, ω_j the weight connection between hidden unit j and the output unit, and $I_j = \sum_i w_{ji}x_i$. A second differentiation yields

$$\frac{\partial^2 f(\mathbf{x}.\mathbf{w})}{\partial^2 w_{ji}} = |x_i|^2 \left[H'(I_j)^2 h''(I)\omega_j^2 + h'(I)H''(I_j)\omega_j\right] \tag{4.12}$$

Combining equations 4.11 and 4.12 with equation 4.6 and carrying out the summation over the inputs, we have

$$\begin{aligned} P_{\text{h}} &= \frac{T}{N}\sum_{\mu,j} |\mathbf{x}^\mu|^2 (y^\mu - f_\mu)\left[H'(I_j)^2 h''(I)\omega_j^2 + h'(I)H''(I_j)\omega_j\right] \\ &\quad + \frac{T}{N}\sum_{\mu,j} |\mathbf{x}^\mu|^2 h'(I)^2 H'(I_j)^2 \omega_j^2 \end{aligned} \tag{4.13}$$

where the summation j is over all the hidden units, and that of μ over all the training patterns. In the following, we assume that $|y^\mu - f^\mu| \leq 1$ for most of the training examples, and investigate the effect of P_{h} accordingly.

We see from equation 4.13 that *hidden-layer noise penalizes large derivatives and large weights at the output layer as well as large derivatives at the hidden layer.* Large derivatives at the output layer are already penalized by the output-layer noise; hidden-layer noise thus reinforces such a penalty. Penalizing large weights at the output layer is the same as favoring small outgoing weights at the hidden layer. A hidden unit that has both a small outgoing weight ω_j and a small activation a_j, which is encouraged by the output-layer noise, has negligible contribution to the network output. Therefore, the combined effect of penalizing large output-layer weights and large hidden-layer activations is to encourage the neural network to use fewer hidden units. However, penalizing large derivatives at the hidden layer works against penalizing large activations at the hidden layer, which is imposed by the output-layer noise, since small derivatives, i.e., small $|H'(I_j)|$ are accompanied by large outputs $|a_j| = |H(I_j)|$. It is plausible that those hidden units that have a relatively large activation, and are thus essential in fitting the training data, are encouraged to operate close to their two saturation states by hidden-layer noise. Those hidden units that have a relatively small activation, and are thus less essential in fitting the training data, are likely to be made redundant. We note also that the hidden-layer penalty is weighted by the length of the input vector $|\mathbf{x}^\mu|^2$. Therefore, the effects of the hidden-layer noise could be more pronounced at large inputs than at small inputs.

In summary, the main effects of weight noise are (1) reducing the number of hidden units, and (2) encouraging sigmoidal units, especially in the output layer, to operate in the saturation states, i.e., firmly on or off. The first effect limits the number of hidden units in a network and hence can prevent overfitting. This way of preventing overfitting is rather different from the way in which input noise prevents overfitting; it bears a resemblance to weight-decay methods (e.g., Chauvin 1989). The second effect is related to the effect of input noise, since a small derivative at the output unit contributes to a small $|\partial f(\mathbf{x}, \mathbf{w})/\partial \mathbf{x}| = h'(I)|\partial I/\partial \mathbf{x}|$ and hence a smoother input–output mapping (except at the class boundary). If these two mechanisms to prevent overfitting do not interfere with learning the essential input–output mapping from the training set, one could expect improved generalization performance. The degree of improvement, however, is likely to be problem dependent.

Although we performed the analysis of the effects of data and weight noise for the case of on-line backpropagation training, essentially the same type of analysis may be applied to study the effects of noises in normal backpropagation training. In fact, that application is given in the Appendix; the results show that adding data noise to the batch (normal) version of the backpropagation algorithm has the same effect on generalization as adding noise to the on-line backpropagation algorithm.

5 Training with Langevin Noise

Both the data and weight noise affect the dynamics of training through $e(z, \mathbf{w})$ during the evaluation of $\Delta\mathbf{w}$. In contrast, the Langevin noise bypasses the neural network and the error function; it is directly injected to the weight changes as indicated in equation 2.7. Because the effect of Langevin noise on the training differs fundamentally from the effect of data and weight noise, a completely different analysis is required. We analyze the Langevin noise using methods of statistical mechanics.

The weight update rule for training with Langevin noise reads (Hertz *et al.* 1989; Rögnvaldsson 1994; Guillerm and Cotter 1991)

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\nabla E(\mathbf{w}) + \xi\sqrt{2T\eta} \tag{5.1}$$

where the noise ξ is a gaussian random variable with mean zero and unit standard deviation. Let $\Delta t = \eta$ be small and constant. Equation 5.1 can be viewed as a discretised version of the following continuous-time Langevin equation (Gillespie 1992; Seung *et al.* 1992):

$$d\mathbf{w} = -\nabla E(\mathbf{w})dt + \xi\sqrt{2Tdt} \tag{5.2}$$

If one regards $\mathbf{w}$ as the coordinate vector of a Brownian particle, then equation 5.2 describes the dynamics of the particle at a temperature T. The particle starts near the origin of the weight space, when $\mathbf{w}$ is initialized to a small value at the beginning of the training. It then diffuses randomly away from the origin by random walks. The gradient of the training error biases the walk toward a minimum of E.

Imagine an ensemble of identical networks that all start from the same initial weight and evolve according to equation 5.1. Owing to the presence of noise, each network in the ensemble will have a different

$\mathbf{w}(t)$ at a given time t. The state of the whole ensemble at any moment can be described by a distribution $\mathcal{G}(\mathbf{w}, t)$. It is a standard result (van Kampen 1981; Gillespie 1992) that the Langevin equation 5.2 is equivalent to a Fokker–Planck equation for $\mathcal{G}(\mathbf{w}, t)$. The stationary solution of the Fokker–Planck equation is described by a Gibbs distribution

$$\mathcal{G}(\mathbf{w}) = \frac{1}{Z} \exp\left[-E(\mathbf{w})/T\right] \tag{5.3}$$

where $Z = \int d\mathbf{w} \exp\left[-E(\mathbf{w})/T\right]$.

It is common to start training with a relatively large amount of Langevin noise and then to reduce it gradually to zero. When the noise is slowly reduced, the Brownian dynamics will lead to a new Gibbs distribution at a lower temperature T. At a very low temperature T, the Gibbs distribution of equation 5.3 will be highly peaked at the global minimum of $E(\mathbf{w})$. Thus, the Brownian dynamics minimizes $E(\mathbf{w})$ as the temperature is gradually reduced.

Training by Brownian dynamics with annealing likewise minimizes $E(\mathbf{w})$, the end configuration of the training being the global minimum of $E(\mathbf{w})$. Therefore, *annealed Langevin noise has no regularization effect.* Langevin noise would lead to improved generalization only if the network is not oversized with respect to the amount of training data.

In fact, the minimization procedure should be generally applicable to problems where gradient information is available. Geman and Hwang (1986) and Kushner (1987) proved the asymptotic convergence to the global minimum of $E(\mathbf{w})$, provided that the temperature is reduced slowly. Global minimization by Brownian dynamics shares the same annealing ideas as the simulated-annealing algorithm of Kirkpatrick *et al.* (1983). The simulated-annealing algorithm, however, uses the Metropolis algorithm rather than Brownian dynamics to generate a Gibbs distribution. Performing simulated annealing by the Brownian-dynamics method instead of by the Metropolis method can have advantages, because the Brownian-dynamics method uses the gradient information. The Brownian-dynamics method can therefore be expected to be more efficient than the Monte Carlo method. Note, however, that the Metropolis algorithm can handle both continuous and discrete variables, whereas the Brownian-dynamics method is suitable only for continuous variables.

6 Experimental Results

In this section, we describe experiments we have performed on two trifling problems, to investigate the following points: (1) Whether input noise constrains the neural-network function to be smooth and hence can improve generalization and output noise has no effect on generalization; (2) how weight noise affects generalization; and (3) whether Langevin noise helps to avoid local minima.

6.1 A Regression Problem. Consider a one-dimensional scalar function defined by

$$y = \sin 3(x + 0.8)^2 + E \tag{6.1}$$

Equation 6.1 specifies a data-generation process in which we included gaussian noise with zero mean and a standard deviation of 0.4 to simulate measurement errors on y. A training set consisting of 15 data points, $\{(x^\mu, y^\mu) : \mu = 1, 2 \ldots 15\}$, was generated using equation 6.1 with uniformly spaced x^μ in the interval $[-1, 1]$. A neural network with one input unit, 15 hidden units, and one output unit, denoted by N1-15-1, was used to fit a curve through the points. The transfer function for the hidden units was taken to be $\tanh(x)$. Since we are dealing with a regression problem, the transfer function for the output unit was taken to be linear, i.e., x. Neural networks with the aforementioned architecture were trained using noise-injection training algorithms and compared with neural networks that were trained by minimizing the quadratic error $E(\mathbf{w})$.

We define a generalization error E_g as

$$E_g = \frac{1}{4} \int_{-1}^{1} [f(x, \mathbf{w}) - \langle y(x) \rangle]^2 \, dx \tag{6.2}$$

In the limit of infinite training examples, the training error $E(\mathbf{w})$ approaches E_g.

6.1.1 Data Noise. One practical issue in carrying out experiments on data noise is that the on-line backpropagation training algorithm converges very slowly. (With noise injection, we need more than five million iterations to obtain convergence.) Another issue is the need to tune the training parameters to obtain convergence. The conjugate-gradient algorithm was therefore found to be a better training alternative. In that case, the $\mathcal{E}(\mathbf{w})$ of equation 3.5 is approximated by a sum over a finite number of synthetic data that are generated according to the density $\hat{g}(z)$ of equation 3.1. This sum essentially represents a Monte Carlo integration of $\mathcal{E}$. To limit the approximation error, a large synthetic data set is needed. We found that $1000 \times N$ synthetic data points are sufficient for the present problem, in which $N = 15$ is the number of training examples. To avoid local minima, each training session was repeated four times with a different starting point. The one that had the smallest training error was used.

The results of training with input noise are shown in Figure 1. The dashed line represents the neural-network function $f(x, \mathbf{w})$ determined by minimizing $E(\mathbf{w})$ using the conjugate-gradient algorithm. The gray line corresponds to the neural-network function that is obtained by training with input noise ($T = 0.005$). The data generation model $\langle y \rangle$ is represented by the solid line. Notice that the dashed line passes through

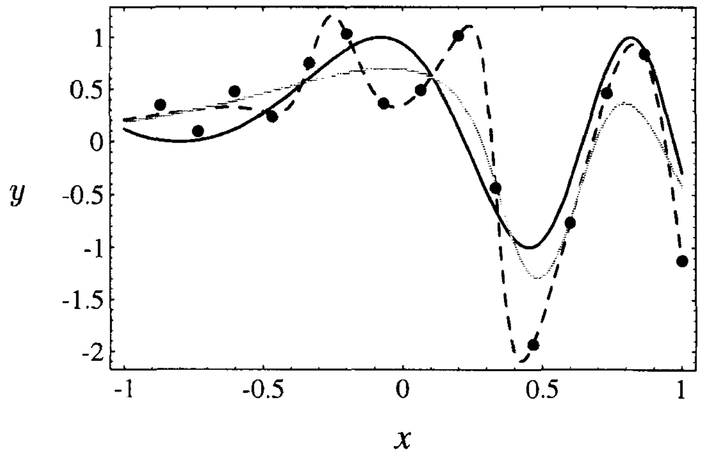


Figure 1: Smoothing effect of input noise. The dots represent the training examples. The underlying model $\langle y \rangle$ that generates the training set is given by the solid line. The gray and dashed lines represent the neural-network functions obtained from training with and without input noise, respectively. See the text for a full explanation.

almost all the training patterns; it overfits the training set. The gray line only globally satisfies the training set and avoids fitting the noise present in the training data. The distance between the gray line and the solid line is clearly smaller than the distance between the dashed line and the solid line. Therefore, the neural network trained with input noise generalizes better than the neural network trained without noise. We found that training with input noise reduces the generalization error for this problem by as much as 61% (Table 1).

Notice that training with input noise forces the neural-network function to be smooth. A smoother neural-network function avoids overfitting the training set and leads to better generalization, as predicted in Section 3. It follows from equation 3.22 that the variance of the injected input noise controls the degree of smoothing of the neural-network function. The larger the variance ($2T$), the smoother the neural network function will be. This is confirmed by our simulation. We have trained the neural network with input noise levels corresponding to $T = 0.002$, 0.005, and 0.009. We found that the neural-network functions become progressively smoother with increasing T. The generalization error decreases at first with increasing noise level, but then it increases with increasing noise level. A value of $T = 0.005$ gives the best generalization results for

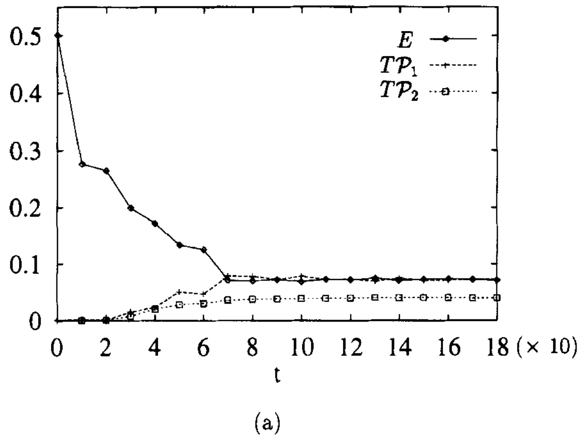
Table 1: Generalization Performance of Training with Noise for the Regression Problem

Noise type	Training error E (10^{-2})	Generalization error E_g (10^{-2})	Reduction in generalization error (%)	Variance $2T(10^{-2})$
No noise	0.25	12.0		0
Input noise	1.9	6.22	48	0.4
Input noise	7.1	4.69	61	1
Input noise	11	5.54	54	1.8
Langevin noise	3.4E-7	11.4	5	

the present problem. For this value of T, the standard deviation of the noise is $0.75\Delta x$, where $\Delta x = 2/15$ is the average interdatum distance in the training set.

To investigate the relation between the new cost function $\mathcal{E}$, the standard error function E, and the noise-induced penalty functions $\mathcal{P}_1$ and $\mathcal{P}_2$, we measured their values as a function of the time (number of iterations) during minimizing $\mathcal{E}$ using the conjugate-gradient method. Results obtained for a single training process with $T = 0.005$ are plotted in Figure 2a and b. At the start of training, the neural network is initialized with small weights. In that case, the neural-network function is quasi-linear with a small slope. This is reflected by the very small values of $T\mathcal{P}_1$ and $T\mathcal{P}_2$ at the start of training (Fig. 2a). As training proceeds, the standard error E decreases, while the penalty terms $T\mathcal{P}_1$ and $T\mathcal{P}_2$ increase. At some stage of training (70 iterations in this case), the reduction of E is counterbalanced by the increase in the penalty terms. This explains the ability of input noise to avoid overfitting. In Figure 2b, we have plotted the new cost function $\mathcal{E}$, together with $E + T(\mathcal{P}_1 + \mathcal{P}_2)$ and the generalization error E_g, as a function of the training time. The figure shows that E_g decreases almost monotonically with training time. It also indicates that the difference between $\mathcal{E}$ and E is well approximated by $T(\mathcal{P}_1 + \mathcal{P}_2)$.

In contrast to the input-noise case, we find practically no difference between the neural-network functions obtained from training with and without the output noise. This result is in agreement with our predictions as given in Section 3. It is instructive to examine the synthetic data generated by the input noise (Fig. 3a) and by the output noise (Fig. 3b). The neural-network functions are forced to vary slowly with x by the synthetic data that are generated by the input noise, since there is an interval of x that corresponds to a few values of y, as can be seen from Figure 3a. In comparison, the synthetic data generated by the output noise spread vertically about the original data point. These synthetic data *share* the same best fit with the original training set. That is why the output noise does not regularize and hence has no effect on generalization.



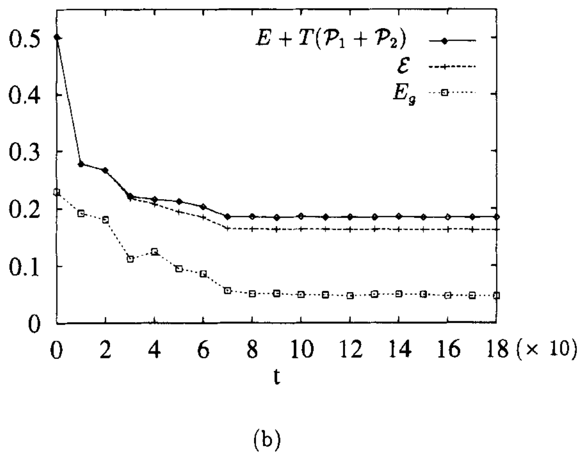


Figure 2: Time evolution of the standard training error E and the penalty terms induced by the input noise. See the text for a full explanation.

6.1.2 Weight Noise. To investigate the effects of the weight noise, we have trained neural networks using the on-line backpropagation algorithm with noise added to all the weights, including the biases. We varied T from 2×10^{-4} to 0.02. The learning rate η was set to 0.01. Without noise addition, the training error decreases to a value of order 10^{-3} after 50,000 × 15 weight updates. It eventually decreases to about 10^{-8}

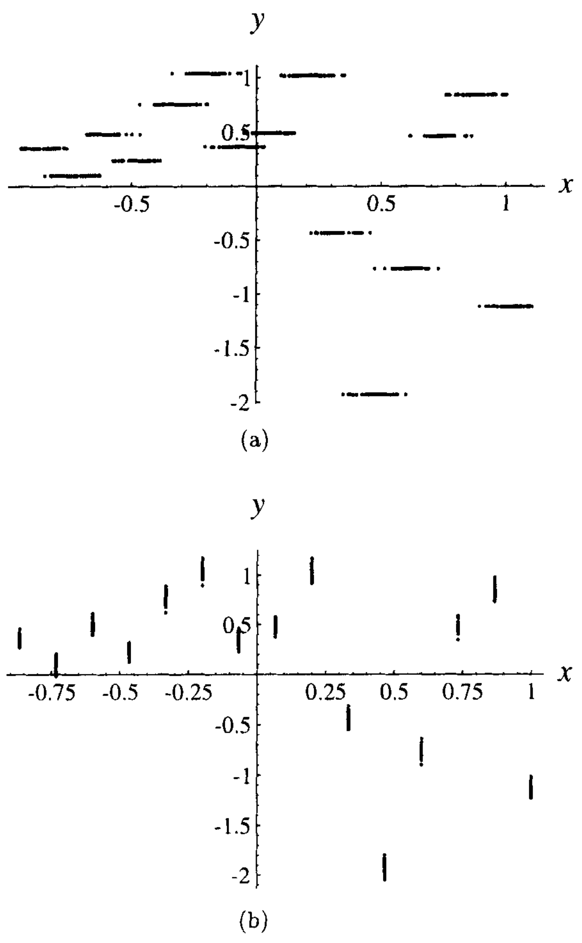


Figure 3: (a) Synthetic data generated by the input noise; (b) synthetic data generated by the output noise.

after very long iterations (500,000 × 15 weight updates). With noise addition, the convergence is even slower. The slowness in convergence of the backpropagation training implies that one cannot be sure that one has reached complete convergence. To minimize the potential regularization effect associated with early stopping (Sjöberg and Ljung 1992), all the neural networks have been trained with the same number of iterations (50,000 × 15 weight updates) instead of waiting for each to reach complete convergence. We found that fluctuations in the training error exist even after a very long training time. To reduce the dependence of our results on these fluctuations, the training and generalization errors are averaged over 10 weight configurations that are sampled at intervals of 100 weight updates just before training is stopped.

We found fewer improvements on the generalization performance comparing with the case of input noise. The largest reduction in the generalization error was found to be 25% at $T = 0.005$, which is significantly less than the 61% achieved using the input noise. At low noise levels ($T \leq 2 \times 10^{-4}$), the noise has very little effect. At high noise levels ($T > 0.02$), the neural-network function attains a quasilinear form and the generalization performance deteriorates.

To verify the predicted mechanisms by which the weight noise improves generalization, we examined the hidden-layer activations and derivatives. With no noise injection, it was found that all 15 hidden units contribute to the network output. At $T = 1.25 \times 10^{-3}$, on average only five hidden units out of 15 had nonnegligible contributions. When the noise level was increased to $T = 0.01$, the number of active hidden units was further reduced to three. This result stands in contrast to training with input noise, where most of the hidden units contribute to the network output. We further found that those active hidden units had activations close to either -1 or 1. The foregoing observations are in agreement with our predictions that weight noise reduces the number of hidden units and encourages small derivatives at sigmoidal units.

6.1.3 Langevin Noise. The neural-network function obtained using Brownian dynamics with a gradually decreasing T is shown as the dashed curve in Figure 4. The temperature (T) is decreased exponentially with the number of iterations during training. The solid curve represents the neural-network function obtained using the conjugate-gradient algorithm. We see that the Langevin noise has indeed no regularization effect, although it is more effective in finding the global minimum of $E(\mathbf{w})$ as manifested by the perfect fit of the training set (shown by the dots).

The training errors and generalization errors for the various neural-network functions are given in Table 1.

6.2 A Classification Problem. Our second example is a two-class classification problem. Through this example, we demonstrate how the input noise and weight noise affect the decision boundaries and how they improve a neural network's generalization performance for classification problems.

The classification problem under consideration is defined by two overlapping bivariate normal distributions. Let $N(\mu, \Sigma)$ be a bivariate normal distribution with mean μ and covariance matrix Σ. The joint probability

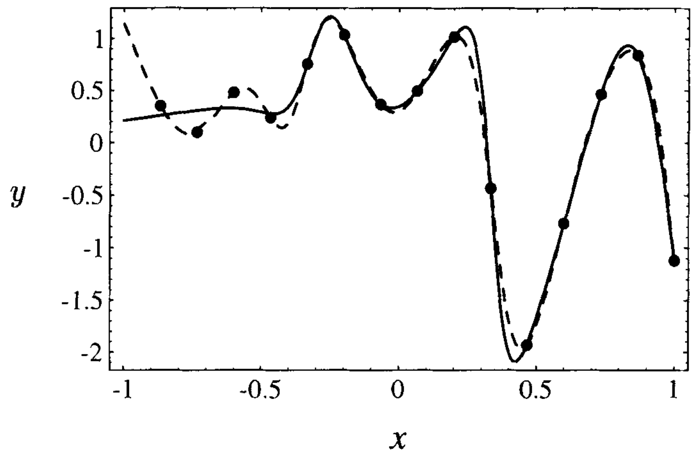


Figure 4: Training with Langevin noise. The dashed and solid curves represent the neural-network functions obtained using Brownian dynamics and the conjugate-gradient algorithm, respectively. Note that the neural network function obtained using the Brownian dynamics (the dashed curve) fits the training set perfectly. This suggests that the dashed curve is at the global minimum of $E(\mathbf{w})$ whereas the solid curve is at a local minimum.

that x will be found and that it belongs to the first class, Class A, is given by

$$P(A,\mathbf{x}) = \frac{1}{2}N(\boldsymbol{\mu}_A, \Sigma) \tag{6.3}$$

The joint probability that x will be found and that it belongs to the second class, Class B, is given by

$$P(B,\mathbf{x}) = \frac{1}{2}N(\boldsymbol{\mu}_B, \Sigma) \tag{6.4}$$

The means and covariances appearing in equation 6.3 and equation 6.4 are given by

$$\boldsymbol{\mu}_A = (1,-1); \qquad \boldsymbol{\mu}_B = (-1,1); \qquad \text{and } \Sigma = \mathbf{I} \tag{6.7}$$

where $\mathbf{I}$ is the two-dimensional identity matrix. Assuming the cost of misclassifying Class A is the same as that of misclassifying Class B, one can derive the optimal classification rule for a two-class classification problem (e.g., Pao 1989):

$$\text{classify } \mathbf{x} \text{ as} \begin{cases} \text{Class } A & \text{if } P(A,\mathbf{x}) > P(B,\mathbf{x}) \\ \text{Class } B & \text{otherwise} \end{cases} \tag{6.8}$$

The decision boundary implied by equation 6.8 is thus described by the equation $P(A.\mathbf{x}) = P(B.\mathbf{x})$. Using equations 6.3 and 6.4, we find that the ideal boundary for the present problem is described by the equation $x_1 = x_2$. A total of 40 training examples were generated, 20 from each class. We assigned a desired output of −0.9 to Class A and 0.9 to Class B. The architecture of the neural network was chosen to be N2-10-1. Both the hidden and the output units are assigned a $\tanh(x)$ transfer function. The neural network had 40 parameters (weights). With the chosen architecture, the number of weights in the neural network matches the number of training examples. Training was performed using the conjugate-gradient algorithm in all cases. The decision boundaries of the neural network are taken to be $f(\mathbf{x}.\mathbf{w}) = 0$ for the present classification problem. Gaussian noise with zero mean was used in all cases.

6.2.1 *Data Noise.* In agreement with our results on the regression problem and with our theoretical analysis, the input noise also makes the neural-network function smoother for classification problems. With an appropriately chosen noise strength T, the neural-network decision boundary comes close to the ideal one. As a consequence, it is found that the generalization performance is significantly improved.

The results of training without noise injection are presented in Figure 5. The neural-network function $f(\mathbf{x}.\mathbf{w})$ is plotted in Figure 5a and the training set and the decision boundary are shown in Figure 5b. The diamonds and the stars represent the training examples from Class A and Class B, respectively. The ideal and the neural-network classification boundaries are represented by the thin and thick lines, respectively. The figure shows that the neural network adapts so much to the training set that it classifies the training set perfectly. It does so by forming complex boundaries, however. Poor generalization is evident, since the neural-network decision boundary differs considerably from the ideal one, as can be seen from Figure 5b.

The results of training with input noise ($T = 0.125$) are presented in Figure 6. It is clear from a comparison of Figures 5 and 6 that input noise smoothes the neural-network function. Input noise causes the decision boundaries to be less well adapted to the training set data and to be closer to the ideal one. Improved generalization is apparent. To quantify the generalization performance, we measured the classification error on a large test set (10,000 data points) that was generated in the same way as the training set. The generalization performance of this training with input noise is summarized in Table 2. As it can be seen from the table, training with noise reduces the misclassification rate from 27 to 9%.

6.2.2 *Weight Noise.* We found, in agreement with previously reported results (Murray and Edwards 1994), that weight noise improves the generalization performance for classification problems. The lowest misclas-

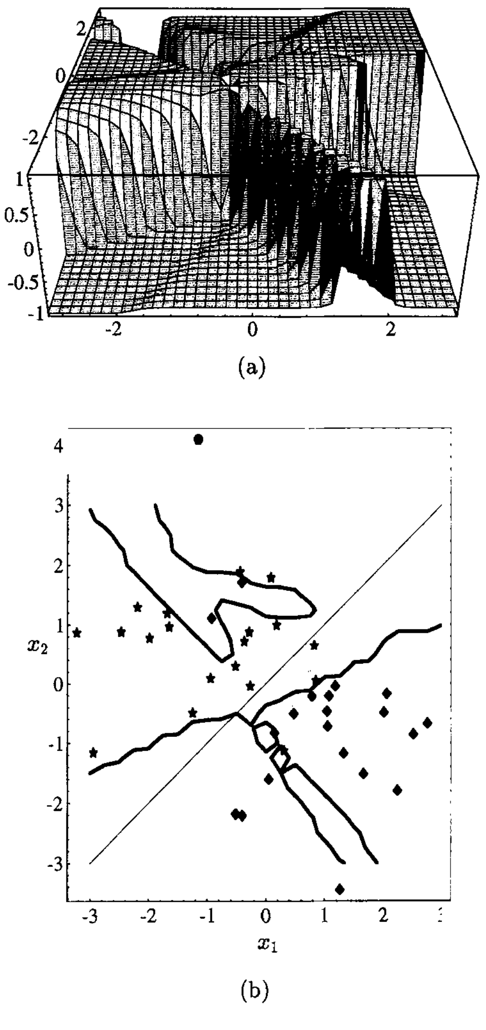


Figure 5: Training without noise injection for a classification problem: (a) the neural-network function; (b) the classification boundaries and the training data. The thick line and the thin line represent, respectively, the neural-network classification boundary and the ideal classification boundary. The diamonds and stars denote the training examples from Class A and Class B, respectively. Note that the neural network makes a perfect classification on the training set by forming complex classification boundaries.

Table 2: Generalization Performance of Training with Input Noise for the Classification Problem[a]

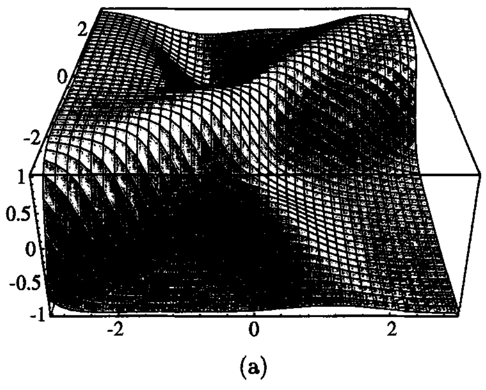
Misclassification on training set (%)	Misclassification on test set (%)	Noise level T
0	27	0
3	17	0.02
10	9	0.125

[a]Bayes classification error for this problem is 8%. Note that, with a noise strength $T = 0.125$, the generalization performance differs from the optimal classification by only 1%.

sification rate that we found on the test set was 11%, which is comparable to the 9% misclassification rate that was achieved using input noise (Tables 2 and 3). For $T > 0.018$, the network functions on one side of the classification boundary had an almost constant -1 value, which changed abruptly to a constant 1 value on the other side of the boundary. This is in line with our prediction that weight noise forces a small derivative $|h'(I)|$ at the output unit, which in turn leads to flat network outputs. In contrast to the regression problem, we found that even with noise addition, all 10 hidden units contribute to the network output. This finding is also in line with the theory that penalizing large derivatives at the sigmoidal units weakens the effect of penalizing large hidden-layer activation. For the present problem, we may conclude that the smoothing effect, i.e., the small derivative effect, is the primary mechanism in improving generalization. To verify this, we applied noise only to the output bias since output-bias noise generates a penalty $P = T\sum_\mu [h''(I)(y^\mu - f_\mu) + h'(I)^2]/N$ (cf. equation 4.10), which favors small $|h'(I)|$ without affecting the hidden activations. As shown in Table 3, applying noise only to the output bias indeed improved the generalization as much as applying noise to all the weights. The best generalization was achieved at a temperature of $T_0 = 0.125$.

7 Conclusions

In this article, we have studied three types of noise injection: data noise, weight noise, and Langevin noise. A rigorous analysis is presented of how the various types of noise affect the learning cost function. The noise-induced penalty functions were computed and analyzed in the weak-noise limit. Their properties were related to the generalization performance of training using noise. Experiments were performed on a regression and a classification problem to illustrate and to compare with formal analysis.



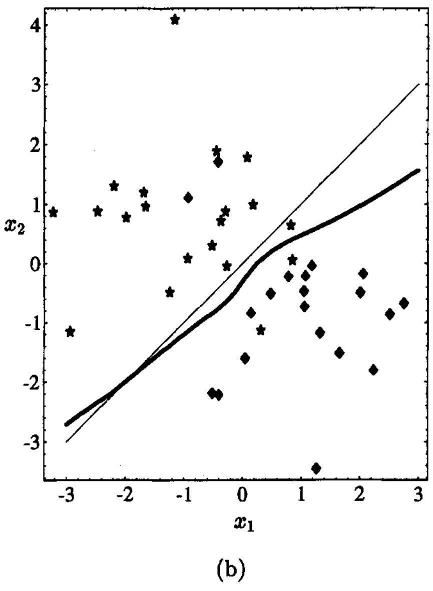


Figure 6: Same as Figure 5 except that the neural network is trained with input noise ($T = 0.125$): (a) the neural-network function; (b) the classification boundaries and the training set. Note that the neural-network function is smoother than that of Figure 5, and that the decision boundary is closer to the ideal one.

Input noise is shown to add two penalty terms to the standard error function. One is identical to a smoothing term as found in regularization theory, while a second term depends on the fitting residuals. The

Table 3: Generalization Performance of Training with Weight Noise for the Classification Problem[a]

Noise type	Misclassification on training set (%)	Misclassification on test set (%)	Noise level T_o	Noise level T_h
Output bias	0	23.3	0.005	0
	7.5	11.1	0.125	0
All weights	0	24	0.011	0.011
	7.5	10.8	0.018	0.018
	7.5	11	0.045	0.045

[a]T_o and T_h denote the temperature at the output and the hidden layer, respectively.

main effect of the input-noise induced penalty terms is in constraining the neural network to be less sensitive to variations in the input data. This smoothing effect could be beneficial to the generalization performance. In contrast to input noise, output noise results only in a constant term in the cost function, and hence does not improve the generalization performance.

We showed that weight noise induces changes in the cost function that are formally similar to that of the input noise. The penalty functions that are induced respectively by the input noise and the weight noise would be identical if differentiations with respect to the inputs and differentiations with respect to the weights are interchanged. Despite such formal similarities, we argued that weight noise has a different effect on the generalization performance than the input noise. Weight noise constrains the neural network to be less sensitive to variations in the weights instead of variations in the inputs. We further demonstrated that weight noise encourages sigmoidal units to operate close to saturation and discourages both large weights in the output layer and large activations in the hidden layer.

On both test problems, input noise significantly reduces the generalization error. On our classification problem, weight noise improves the generalization as much as the input noise. However, in the test-case regression problem weight noise improves the generalization substantially less than input noise. Owing to the limited scale of the test case, this may not be generally true.

We argued that training with annealed Langevin noise results in a global minimization similar to that of simulated annealing. Therefore, training with annealed Langevin noise has no regularization effect, although it could be effective in finding the global minimum, as demonstrated by our experiment.

Appendix: Noise Injection During Normal Backpropagation Training

In this appendix, we show how the analysis presented in Section 3 can be extended to the normal backpropagation training.

Define a vector that represents the whole training set by $\mathbf{Z}_0 \equiv (z^1, z^2, \ldots, z^\mu, \ldots, z^N)$ and make the dependency of $E(\mathbf{w})$ on the training set explicit by denoting it by $E(\mathbf{Z}_0, \mathbf{w}) \equiv E(\mathbf{w})$. Let $\mathbf{Z}$ be a vector in the same space as $\mathbf{Z}_0$. The standard error $E(\mathbf{w})$ can again be written into the form of equation 3.3, with the correspondence $\mathbf{u}$ to $\mathbf{Z}$, $c(\mathbf{u}, \mathbf{w})$ to $E(\mathbf{Z}, \mathbf{w})$ and $g(\mathbf{Z}) = \delta(\mathbf{Z} - \mathbf{Z}_0)$. By applying the stochastic gradient-descent algorithm to this form of $E(\mathbf{w})$, we rederive the backpropagation training algorithm. Therefore, both the on-line backpropagation and the normal backpropagation training algorithm can be treated as special cases of the stochastic gradient-descent algorithm.

Consider now a set of N data points represented by $\mathbf{Z} = (Z_1, Z_2, \ldots, Z_N)$ where Z_μ denotes a data point that we have until now denoted by z^μ. In the vector space of $\mathbf{Z}$, the training set is represented by a single vector $\mathbf{Z}_0 = (z^1, z^2, \ldots, z^N)$. Recall that the backpropagation training is described by the following weight update equation:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} E(\mathbf{Z}_0, \mathbf{w}) \tag{A.1}$$

This training algorithm, although deterministic in nature, may be treated as a special case of the stochastic gradient-descent algorithm by interpreting $\mathbf{Z}_0$ as a realization of the following distribution:

$$g(\mathbf{Z}) = \delta(\mathbf{Z} - \mathbf{Z}_0)$$

Bearing the above in mind, it is clear that with the data noise injection procedure detailed in equation 2.5, the training algorithm is now described by

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} E(\mathbf{Z}, \mathbf{w}) \tag{A.2}$$

where $\mathbf{Z}$ is randomly drawn from the density

$$\hat{g}(\mathbf{Z}) = \prod_{\mu=1}^{N} \rho(Z_\mu - z^\mu) \tag{A.3}$$

Therefore, the cost function that is minimized by equation A.2 has the form

$$C(\mathbf{w}) = \int E(\mathbf{Z}, \mathbf{w}) \hat{g}(\mathbf{Z})\, d\mathbf{Z} \tag{A.4}$$

Substituting equations 2.2 and A.3 into equation A.4 after simplifying the integration over $\mathbf{Z}$, we have

$$C(\mathbf{w}) = \frac{1}{N} \sum_{\mu=1}^{N} \int e(Z_\mu, \mathbf{w}) \rho(Z_\mu - z^\mu)\, dZ_\mu \tag{A.5}$$

Renaming the dummy integration variable Z_μ in equation A.5 z, we obtain

$$C(\mathbf{w}) = \frac{1}{N}\sum_{\mu=1}^{N}\int e(z,\mathbf{w})\rho(z - z^{\mu})\,dz \tag{A.6}$$

This is exactly the $\mathcal{E}(\mathbf{w})$ given by equation 3.5, with $\hat{g}(z)$ given by equation 3.1.

Acknowledgments

I thank Wim Schinkel for many helpful suggestions that have lead to numerous improvements in the presentation of this work.

References

Arfken, G. 1985. *Mathematical Methods for Physicists*. Academic Press, New York.

Bishop, C. M. 1995. Training with noise is equivalent to Tikhomov regularization. *Neural Comp.* **7**, 108–116.

Bottou, L. 1991. Stochastic gradient learning in neural networks. *NEURO-NIMES'91*, EC2, Nanterre, France, 687–606.

Chauvin, Y. 1989. A back-propagation algorithm with optimal use of hidden units. In *Advances in Neural Information Processing System I*, D. Touretsky, ed., pp. 519–526. Morgan Kaufmann, San Mateo, CA.

Clay, R., and Sequin, C. 1992. Fault tolerance training improves generalization and robustness. *Proc. Int. Joint Conf. Neural Networks*, IEEE Neural Council, Baltimore, I-769–774.

Drucker, H., and Le Cun, Y. 1992. Improving generalisation performance using double back-propagation. *IEEE Trans. Neural Networks* **3**, 991–997.

Geman, S., and Hwang, C. 1986. Diffusion for global optimization. *SIAM J. Control Optim.* **25**, 1031–1043.

Gillespie, D. 1992. *Markov Processes*. Academic Press, London.

Guillerm, T., and Cotter, N. 1991. A diffusion process for global optimization in neural networks. *Proc. Int. Joint Conf. Neural Networks*, I-335.

Guyon, A., Vapnik, V., Boser, B., Bottou, L., and Solla, S. A. 1992. Structural risk minimization for character recognition. In *Advances in Neural Information Processing System 4 (NIPS 91)*, J. Moody et al., eds., pp. 471–479. Morgan Kaufmann, San Mateo, CA.

Hanson, S. J. 1990. A stochastic version of the delta rule. *Physica D* **42**, 265–272.

Hertz, J., Krogh, A., and Thorbergsson, G. 1989. Phase transitions in simple learning. *J. Phys. A: Math. Gen.* **22**, 2133–2150.

Hinton, G. E. 1986. Learning distributed representations of concepts. *Proc. Eighth Annu. Conf. Cog. Sci. Soc., Amherst*, 1–12.

Hinton, G. E., and van Camp, D. 1993. Keeping neural networks simple by minimizing the description length of the weights. *Sixth ACM Conf. Comp. Learning Theory (Santa Cruz)*, 5–13.

Holmström, L., and Koistinen, P. 1992. Using additive noise in back-propagation training. *IEEE Trans. Neural Networks* **3**, 24–38.

Kendall, G. D., and Hall, T. J. 1993. Optimal network construction by minimum description length. *Neural Comp.* **5**, 210–212.

Kendall, M., and Stuart, A. 1977. *The Advanced Theory of Statistics*, Vol. 1, 4th ed., Charles Griffin, London.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. 1983. Optimization by simulated annealing. *Science* **220**, 671–680.

Krogh, A., and Hertz, J. A. 1992. A simple weight decay can improve generalization. In *Advances in Neural Information Processing System 4 (NIPS 91)*, J. E. Moody *et al.*, eds., pp. 950–957. Morgan Kaufmann, San Mateo, CA.

Kushner, H. 1987. Asymptotic global behavior for stochastic approximation and diffusions with slowly decreasing noise effects: Global minimization via Monte Carlo. *SIAM J. Appl. Math.* **47**, 169–185.

MacKay, D. J. C. 1992. Bayesian interpolation. *Neural Comp.* **4**, 415–447.

Matsuoka, K. 1992. Noise injection into inputs in back-propagation learning. *IEEE Trans. Sys. Man Cybern.* **22**, 436–440.

Murray, A. F., and Edwards, P. J. 1993. Synaptic weight noise during multilayer perceptron training: Fault tolerance and training improvements. *IEEE Trans. Neural Networks* **4**, 722–725.

Murray, A. F., and Edwards, P. J. 1994. Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Trans. Neural Networks* **5**, 792–802.

NeuralWare. 1991. *Reference Guide: NeuralWorks Professional II/PlUS and NeuralWorks Explorer*. NeuralWare, Inc, Pittsburgh.

Nowlan, S. J., and Hinton, G. E. 1992. Simplifying neural networks by soft weight-sharing. *Neural Comp.* **4**, 473–493.

Pao, Y. H. 1989. *Adaptive Pattern Recognition and Neural Networks*, chap. 2, pp. 25–35. Addison-Wesley, Reading, MA.

Poggio, T., and Girosi, F. 1990. Networks for approximation and learning. *Proc. IEEE* **78**, 1481–1497.

Reed, R., Oh, S., and Marks, R. J., II. 1992. Regularization using jittered training data. *Proc. Int. Joint Conf. Neural Networks*, IEEE Neural Council, Baltimore, III–147.

Rissanen, J. 1989. *Stochastic Complexity in Statistical Inquiry*. World Scientific, Singapore.

Rögnvaldsson, T. 1994. On Langevin updating in multilayer perceptrons. *Neural Comp.* **6**, 916–926.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986. Learning representations by back-propagating errors. *Nature (London)* **323**, 533–536.

Seung, H., Sompolinsky, H., and Tishby, N. 1992. Statistical mechanics of learning from examples. *Phys. Rev. A* **45**, 6056–6091.

Sietsma, J., and Dow, R. 1988. Neural network pruning—why and how. *Proc. IEEE Int. Conf. Neural Networks* **I**, 325–333.

Sjöberg, J., and Ljung, L. 1992. Overtraining, regularisation, and searching for minimum in neural networks. *Proc. Symp. Adaptive Systems Control Signal Process.*, Grenoble, France.

van Kampen, N. G. 1981. *Stochastic Processes in Physics and Chemistry.* North-Holland, Amsterdam.

Weigend, A., Rumelhart, D., and Huberman, B. 1991. Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing System 3 (NIPS 90),* J. M. R. P. Lippmann and D. Touretsky, eds., pp. 875–882. Morgan Kaufmann, San Mateo, CA.

White, H. 1989. Learning in artificial neural networks: A statistical perspective. *Neural Comp.* **1**, 425–464.

Received January 10, 1995; accepted July 14, 1995.

This article has been cited by:

1. Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* **61**, 85-117. [CrossRef]
2. J.M. Alonso-Weber, M.P. Sesmero, A. Sanchis. 2014. Combining additive input noise annealing and pattern transformations for improved handwritten character recognition. *Expert Systems with Applications* . [CrossRef]
3. Pierre Baldi, Peter Sadowski. 2014. The dropout learning algorithm. *Artificial Intelligence* **210**, 78-122. [CrossRef]
4. Ahmet Alptekin, Olcay Kursun. 2013. Miss One Out: A Cross-Validation Method Utilizing Induced Teacher Noise. *International Journal of Pattern Recognition and Artificial Intelligence* 1351003. [CrossRef]
5. Taeho Jo. 2013. VTG schemes for using back propagation for multivariate time series prediction. *Applied Soft Computing* **13**, 2692-2702. [CrossRef]
6. Luqman R. Bachtiar, Charles P. Unsworth, Richard D. Newcomb, Edmund J. Crampin. 2013. Multilayer Perceptron Classification of Unknown Volatile Chemicals from the Firing Rates of Insect Olfactory Sensory Neurons and Its Application to Biosensor Design. *Neural Computation* **25**:1, 259-287. [Abstract] [Full Text] [PDF] [PDF Plus]
7. Kevin Ho, Hsia-Ching Chang, Wei-Bin Lee. 2013. Effect of Noise on Gradient Systems. *International Journal of Computer Theory and Engineering* 841-845. [CrossRef]
8. Adam P. Piotrowski, Jarosław J. Napiorkowski. 2013. A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling. *Journal of Hydrology* **476**, 97-111. [CrossRef]
9. Adam P. Piotrowski, Pawel M. Rowinski, Jaroslaw J. Napiorkowski. 2012. Comparison of evolutionary computation techniques for noise injected neural network training to estimate longitudinal dispersion coefficients in rivers. *Expert Systems with Applications* **39**, 1354-1361. [CrossRef]
10. T.A. Choudhury, N. Hosseinzadeh, C.C. Berndt. 2011. Artificial Neural Network application for predicting in-flight particle characteristics of an atmospheric plasma spray process. *Surface and Coatings Technology* **205**, 4886-4895. [CrossRef]
11. Sultan Uddin Ahmed, Md. Shahjahan, Kazuyuki Murase. 2011. Injecting Chaos in Feedforward Neural Networks. *Neural Processing Letters* . [CrossRef]
12. Kevin Ho, Chi-Sing Leung, John Sum. 2011. Objective Functions of Online Weight Noise Injection Training Algorithms for MLPs. *IEEE Transactions on Neural Networks* **22**, 317-323. [CrossRef]
13. Jing Yang, Xu Yu, Zhi-Qiang Xie, Jian-Pei Zhang. 2010. A novel virtual sample generation method based on Gaussian distribution. *Knowledge-Based Systems* . [CrossRef]

14. Israel Gonzalez-Carrasco, Angel Garcia-Crespo, Belen Ruiz-Mezcua, Jose Luis Lopez-Cuadrado. 2010. An optimization methodology for machine learning strategies and regression problems in ballistic impact scenarios. *Applied Intelligence* . [CrossRef]

15. Taeho Jo. 2010. The effect of mid-term estimation on back propagation for time series prediction. *Neural Computing and Applications* **19**, 1237-1250. [CrossRef]

16. Kevin I.-J Ho, Chi-Sing Leung, John Sum. 2010. Convergence and Objective Functions of Some Fault/Noise-Injection-Based Online Learning Algorithms for RBF Networks. *IEEE Transactions on Neural Networks* **21**, 938-947. [CrossRef]

17. Adam Petrie, Thomas R. Willemain. 2010. The snake for visualizing and for counting clusters in multivariate data. *Statistical Analysis and Data Mining* n/a-n/a. [CrossRef]

18. Israel Gonzalez-Carrasco, Angel Garcia-Crespo, Belen Ruiz-Mezcua, Jose Luis Lopez-Cuadrado. 2009. Dealing with limited data in ballistic impact scenarios: an empirical comparison of different neural network approaches. *Applied Intelligence* . [CrossRef]

19. Chao Tan, Hui Chen, Chengyun Xia. 2009. Early prediction of lung cancer based on the combination of trace element analysis in urine and an Adaboost algorithm. *Journal of Pharmaceutical and Biomedical Analysis* **49**, 746-752. [CrossRef]

20. Richard M. Zur, Yulei Jiang, Lorenzo L. Pesce, Karen Drukker. 2009. Noise injection for training artificial neural networks: A comparison with weight decay and early stopping. *Medical Physics* **36**, 4810. [CrossRef]

21. G ZHANG. 2007. A neural network ensemble method with jittered training data for time series forecasting. *Information Sciences* **177**, 5329-5346. [CrossRef]

22. GEORGE D. MAGOULAS, ARISTOKLIS ANASTASIADIS. 2006. APPROACHES TO ADAPTIVE STOCHASTIC SEARCH BASED ON THE NONEXTENSIVE q-DISTRIBUTION. *International Journal of Bifurcation and Chaos* **16**, 2081-2091. [CrossRef]

23. S GUESSASMA, M BOUNAZEF, P NARDIN. 2006. Neural computation analysis of alumina–titania wear resistance coating. *International Journal of Refractory Metals and Hard Materials* **24**, 240-246. [CrossRef]

24. Salvatore Ingrassia, Isabella Morlini. 2005. Neural Network Modeling for Small Datasets. *Technometrics* **47**, 297-311. [CrossRef]

25. P. Chandra, Y. Singh. 2004. Feedforward Sigmoidal Networks&#8212;Equicontinuity and Fault-Tolerance Properties. *IEEE Transactions on Neural Networks* **15**, 1350-1366. [CrossRef]

26. Zhe Chen, Simon Haykin. 2002. On Different Facets of Regularization Theory. *Neural Computation* **14**:12, 2791-2846. [Abstract] [PDF] [PDF Plus]

27. Edward Wilson, Stephen M. Rock. 2002. Gradient-based parameter optimization for systems containing discrete-valued functions. *International Journal of Robust and Nonlinear Control* **12**:10.1002/rnc.v12:11, 1009-1028. [CrossRef]

28. PITOYO HARTONO, SHUJI HASHIMOTO. 2002. EXTRACTING THE PRINCIPAL BEHAVIOR OF A PROBABILISTIC SUPERVISOR THROUGH NEURAL NETWORKS ENSEMBLE. *International Journal of Neural Systems* **12**, 291-301. [CrossRef]

29. A. Alessandri, M. Sanguineti, M. Maggiore. 2002. Optimization-based learning with bounded error for feedforward neural networks. *IEEE Transactions on Neural Networks* **13**, 261-273. [CrossRef]

30. Vicente Ruiz de Angulo, Carme Torras. 2001. Architecture-Independent Approximation of Functions. *Neural Computation* **13**:5, 1119-1135. [Abstract] [PDF] [PDF Plus]

31. A Nakashima. 2001. Error correcting memorization learning for noisy training examples. *Neural Networks* **14**, 79-92. [CrossRef]

32. R.I. LEVIN, N.A.J. LIEVEN, M.H. LOWENBERG. 2000. MEASURING AND IMPROVING NEURAL NETWORK GENERALIZATION FOR MODEL UPDATING. *Journal of Sound and Vibration* **238**, 401-424. [CrossRef]

33. Y. Grandvalet. 2000. Anisotropic noise injection for input variables relevance determination. *IEEE Transactions on Neural Networks* **11**, 1201-1212. [CrossRef]

34. Katsuhisa Hirokawa, Kazuyoshi Itoh, Yoshiki Ichioka. 2000. Invariant Pattern Recognition Using Neural Networks Combined with Optical Wavelet Preprocessor. *Optical Review* **7**, 284-293. [CrossRef]

35. Chuan Wang, J.C. Principe. 1999. Training neural networks with additive noise in the desired signal. *IEEE Transactions on Neural Networks* **10**, 1511-1517. [CrossRef]

36. P.J. Edwards, A.F. Murray. 1998. Fault tolerance via weight noise in analog VLSI implementations of MLPs-a case study with EPSILON. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* **45**, 1255-1262. [CrossRef]