Simplifying Neural Networks by Soft Weight-Sharing

Steven J. Nowlan
Computational Neurobiology Laboratory, The Salk Institute, P.O. Box 5800, San Diego, CA 92186-5800 USA

Geoffrey E. Hinton
Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4

One way of simplifying neural networks so they generalize better is to add an extra term to the error function that will penalize complexity. Simple versions of this approach include penalizing the sum of the squares of the weights or penalizing the number of nonzero weights. We propose a more complicated penalty term in which the distribution of weight values is modeled as a mixture of multiple gaussians. A set of weights is simple if the weights have high probability density under the mixture model. This can be achieved by clustering the weights into subsets with the weights in each cluster having very similar values. Since we do not know the appropriate means or variances of the clusters in advance, we allow the parameters of the mixture model to adapt at the same time as the network learns. Simulations on two different problems demonstrate that this complexity term is more effective than previous complexity terms.

1 Introduction

A major problem in training artificial neural networks is to ensure that they will generalize well to cases that they have not been trained on. Some recent theoretical results (Baum and Haussler 1989) have suggested that in order to guarantee good generalization the amount of information required to specify directly the output vectors of all the training cases must be considerably larger than the number of independent weights in the network. In many practical problems there is only a small amount of labeled data available for training and this creates problems for any approach that uses a large, homogeneous network in order to avoid the detailed task analysis required to design a network with fewer independent weights and a specific architecture that is appropriate to the task. As a result, there has been much recent interest in techniques that can train large networks with relatively small amounts of labeled data and still provide good generalization performance.

One way to achieve this goal is to reduce the effective number of free parameters in the network. A number of authors (e.g., Rumelhart *et al.* 1986, chapter 8; Lang *et al.* 1990; le Cun 1989, 1987) have proposed the idea of *weight sharing*, in which a single weight is shared among many connections in the network so that the number of adjustable weights in the network is much less than the number of connections. This approach is effective when the problem being addressed is quite well understood, so that it is possible to specify, in advance, which weights should be identical (le Cun *et al.* 1990).

Another approach is to use a network with too many weights, but to stop the training before overlearning on the training set has occurred (Morgan and Bourlard 1989; Weigend *et al.* 1990). In addition to the usual training and testing sets, a validation set is used. When performance on the validation set starts to *decrease* the network is beginning to overfit the training set and training is stopped. Some experience with this technique has suggested that its effectiveness can be quite sensitive to the particular stopping criterion used (Weigend *et al.* 1990).[1]

Yet another approach to the generalization problem attempts to remove excess weights from the network, either during or after training, to improve the generalization performance. Mozer and Smolensky (1989) and le Cun *et al.* (1990) have both proposed techniques in which a network is initially trained with an excess number of parameters and then a criterion is used to remove redundant parameters. The reduced network is then trained further. The cycle of reduction and retraining may be repeated more than once. The approach of Mozer and Smolensky (1989) estimates the relevance of individual *units* to network performance and removes redundant units and their weights. The method of le Cun *et al.* (1990) uses second-order gradient information to estimate the sensitivity of network performance to the removal of each weight, and removes the least critical weights.

An older and simpler approach to removing excess weights from a network is to add an extra term to the error function that penalizes complexity:

$$\text{cost} = \text{data-misfit} + \lambda\,\text{complexity} \tag{1.1}$$

During learning, the network is trying to find a locally optimal trade-off between the data-misfit (the usual error term) and the complexity of the net. The relative importance of these two terms can be estimated by finding the value of λ that optimizes generalization to a validation set. Probably the simplest approximation to complexity is the sum of the squares of the weights, $\sum_i w_i^2$. Differentiating this complexity measure leads to simple *weight decay* (Plaut *et al.* 1986) in which each weight

[1]This approach is *not* the way in which cross-validation is usually used in the statistics community. Usually a cross-validation set is used to determine the scale factor applied to a regularization term added to the optimization to prevent overfitting. We will see examples of this in the following sections.

decays toward zero at a rate that is proportional to its magnitude. This decay is countered by the gradient of the error term, so weights that are not critical to network performance, and hence always have small error gradients, decay away leaving only the weights necessary to solve the problem. At the end of learning, the magnitude of a weight is exactly proportional to its error derivative, which makes it particularly easy to interpret the weights (see, for example, Hinton 1986). Minimizing $\sum_i w_i^2$ is a well-known technique when fitting linear regression models that have too many degrees of freedom. One justification is that it minimizes the sensitivity of the output to noise in the input, since in a linear system the variance of the noise in the output is just the variance of the noise in the input multiplied by the squared weights (Kohonen 1977).

The use of a $\sum_i w_i^2$ penalty term can also be interpreted from a Bayesian perspective.[2] The "complexity" of a set of weights, $\lambda \sum_i w_i^2$, may be described as its negative log probability density under a radially symmetric gaussian prior distribution on the weights. The distribution is centered at the origin and has variance $1/\lambda$. For multilayer networks, it is hard to find a good theoretical justification for this prior, but Hinton (1987) justifies it empirically by showing that it greatly improves generalization on a very difficult task. More recently, MacKay (1991) has shown that even better generalization can be achieved by using different values of λ for the weights in different layers.

2 A More Complex Measure of Network Complexity

One potential drawback of $\sum_i w_i^2$ as a penalty term is that it can favor two weak interactions over one strong one. For example, if a unit receives input from two units that are highly correlated with each other, its behavior will be similar if the two connections have weights of w and 0 or weights of $w/2$ and $w/2$. The penalty term favors the latter because

$$\left(\frac{w}{2}\right)^2 + \left(\frac{w}{2}\right)^2 < w^2 + 0^2 \tag{2.1}$$

If we wish to drive small weights toward zero without forcing large weights away from the values they need to model the data, we can use a prior which is a mixture of a narrow (n) and a broad (b) gaussian, both centered at zero.

$$p(w) = \pi_n \frac{1}{\sqrt{2\pi}\sigma_n} e^{-w^2/2\sigma_n^2} + \pi_b \frac{1}{\sqrt{2\pi}\sigma_b} e^{-w^2/2\sigma_b^2} \tag{2.2}$$

where π_n and π_b are the mixing proportions of the two gaussians and are therefore constrained to sum to 1.

[2]R. Szeliski, personal communication (1985).

Assuming that the weight values were generated from a gaussian mixture, the conditional probability that a particular weight, w_i, was generated by a particular gaussian, j, is called the *responsibility*[3] of that gaussian for the weight and is given by

$$r_j(w_i) = \frac{\pi_j p_j(w_i)}{\sum_k \pi_k p_k(w_i)} \tag{2.3}$$

where $p_j(w_i)$ is the probability density of w_i under gaussian j.

When the mixing proportions of the two gaussians are comparable, the narrow gaussian gets most of the responsibility for a small weight. Adopting the Bayesian perspective, the cost of a weight under the narrow gaussian is proportional to $w^2/2\sigma_n^2$. As long as σ_n is quite small there will be strong pressure to reduce the magnitude of small weights even further. Conversely, the broad gaussian takes most of the responsibility for large weight values, so there is much less pressure to reduce them. In the limiting case when the broad gaussian becomes a uniform distribution, there is almost no pressure to reduce very large weights because they are almost certainly generated by the uniform distribution. A complexity term very similar to this limiting case has been used successfully by Weigend *et al.* (1990) to improve generalization for a time series prediction task.[4]

There is an alternative justification for using a complexity term that is a mixture of a uniform distribution and a narrow, zero-mean gaussian. The negative log probability is approximately constant for large weights but smoothly approaches a much lower value as the weight approaches zero. So the complexity cost is a smoothed version of the obvious discrete cost function that has a value of zero for weights which are zero and a value of 1 for all other weights. This smoothed cost function is suitable for gradient descent learning, whereas the discrete one is not.

3 Adaptive Gaussian Mixtures and Soft Weight-Sharing

A mixture of a narrow, zero-mean gaussian with a broad gaussian or a uniform distribution allows us to favor networks with many near-zero weights, and this improves generalization on many tasks, particularly those in which there is some natural measure of locality that determines which units need to interact and which do not. But practical experience with hand-coded weight constraints has also shown that great improvements can be achieved by constraining particular subsets of the weights to share the same value (Lang *et al.* 1990; le Cun 1989). Mixtures of zero-mean gaussians and uniforms cannot implement this type of symmetry constraint. If however, we use multiple gaussians and allow their means

[3]This is more commonly referred to as the *posterior probability* of gaussian j given weight w_i.

[4]See Nowlan (1991) for a precise description of the relationship between mixture models and the model used by Weigend *et al.* (1990).

and variances to adapt as the network learns, we can implement a "soft" version of weight-sharing in which the learning algorithm decides for itself which weights should be tied together. (We may also allow the mixing proportions to adapt so that we are not assuming all sets of tied weights are the same size.)

If we know, in advance, that two connections should probably have the same weight, we can introduce a complexity penalty proportional to the squared difference between the two weights. But if we do not know which pairs of weights should be the same it is harder to see how to favor solutions in which the weights are divided into subsets and the weights within a subset are nearly identical. We now show that a mixture of gaussians model can achieve just this effect. The basic idea is that a gaussian that takes responsibility for a subset of the weights will squeeze those weights together since it can then have a lower variance and hence assign a higher probability density to each weight. If the gaussians all start with high variance, the initial division of weights into subsets will be very soft. As the variances shrink and the network learns, the decisions about how to group the weights into subsets are influenced by the task the network is learning to perform.

4 An Update Algorithm

To make the intuitive ideas of the previous section a bit more concrete, we may define a cost function of the general form given in (1.1):

$$C = \frac{K}{\sigma_y^2} \sum_c \frac{1}{2}(y_c - d_c)^2 - \sum_i \log \left[\sum_j \pi_j p_j(w_i) \right] \tag{4.1}$$

where σ_y^2 is the variance of the squared error and each $p_j(w_i)$ is a gaussian density with mean μ_j and standard deviation σ_j. We will optimize this function using some form of gradient descent to adjust the w_i as well as the mixture parameters π_j, μ_j, and σ_j, and σ_y.[5]

The partial derivative of C with respect to each weight is the sum of the usual squared error derivative and a term due to the complexity cost for the weight:

$$\frac{\partial C}{\partial w_i} = \frac{K}{\sigma_y^2} \sum_c (y_c - d_c) \frac{\partial y_c}{\partial w_i} - \sum_j r_j(w_i) \frac{(\mu_j - w_i)}{\sigma_j^2} \tag{4.2}$$

The derivative of the complexity cost term is simply a weighted sum of the difference between the weight value and the center of each of the

[5] $1/\sigma_y^2$ may be thought of as playing the same role as λ in equation 1.1 in determining a trade-off between the misfit and complexity costs. σ_y is re-estimated as learning proceeds so this trade-off is not constant. K is a factor that adjusts for the effective number of degrees of freedom (or number of well determined parameters) in a problem. For the simulations described here its value was close to 1.0 and was determined by cross-validation.

gaussians. The weighting factors are the *responsibility* measures defined in equation 2.3 and if over time a single gaussian claims most of the responsibility for a particular weight the effect of the complexity cost term is simply to pull the weight toward the center of the responsible gaussian. The strength of this force is inversely proportional to the variance of the gaussian. Notice also that since the derivative of the complexity cost term is actually a sum of forces exerted by each gaussian, the *net* force exerted on the weight can be very small even when the forces exerted by some of the individual gaussians are quite large (e.g., a weight placed midway between two gaussians of equal variance has zero net force acting on it). This allows one to set up initial conditions in which each gaussian accounts quite well for at least some of the weights in the network, but the overall force on any weight due to the complexity term is negligible so the weights are initially driven primarily by the derivative of the data-misfit term.

The partial derivatives of C with respect to the means and variances of the gaussians in the mixture have similar forms:

$$\frac{\partial C}{\partial \mu_j} = \sum_i r_j(w_i)\frac{(\mu_j - w_i)}{\sigma_j^2} \tag{4.3}$$

$$\frac{\partial C}{\partial \sigma_j} = -\sum_i r_j(w_i)\frac{(\mu_j - w_i)^2 - \sigma_j^2}{\sigma_j^3} \tag{4.4}$$

The derivative for μ_j simply drives μ_j toward the *weighted* average of the set of weights gaussian j is responsible for. Similarly the derivative for σ_j drives it toward the weighted average of the squared deviations of these weights about μ_j. The derivation of the partial of C with respect to the mixing proportions is slightly less straightforward since we must worry about maintaining the constraint that the mixing proportions must sum to 1. Appropriate use of a Lagrange multiplier and a bit of algebraic manipulation leads to the simple expression:

$$\frac{\partial C}{\partial \pi_j} = \sum_i \left[1 - \frac{r_j(w_i)}{\pi_j}\right] \tag{4.5}$$

Once again the result is intuitive; π_j is moved toward the average responsibility of gaussian j for all of the weights.

The partial derivatives of C with respect to each of the mixture parameters are simple enough that, for fixed values of the responsibilities, the exact minimizer can be found analytically with ease (for example, the minimizer for equation 4.3 is simply $\mu_j = \sum_i r_j(w_i)w_i / \sum_i r_j(w_i)$). This suggests that one could proceed by simply recomputing the $r_j(w_i)$ after each weight update and setting the μ_j, σ_j, and π_j to their exact minimizers given the current $r_j(w_i)$. In fact the process of recomputing the $r_j(w_i)$ and then setting all the parameters to their analytic minimizers corresponds to one iteration of the *EM* algorithm applied to mixture estimation (Dempster *et al.* 1977). This is a sensible and quite efficient algorithm to use for estimating the mixture parameters when we are dealing with a *stationary* data distribution. However, in the case we are considering it is clear that the "data" we are modeling, the set of w_i, does not have a stationary distribution. In order to avoid stability problems, it is very important that the rate of change of our mixture parameters be tied to the rate of change of the weights themselves. For this reason, we choose to update all of the parameters (w_i, μ_j, σ_j, π_j) *simultaneously* using a conjugate gradient descent procedure.[6]

Before considering applications of the method outlined above we need to consider briefly the issue of initializing all of our parameters appropriately. It is well known that maximum likelihood methods for fitting mixtures can be very sensitive to poor initial conditions (McLachlan and Basford 1988). For example, if one component of a mixture initially has little responsibility for any of the weights in the network, its mixing proportion is driven rapidly toward zero and it is very difficult to recover from this situation. Fortunately, in the case of a network we usually know the initial weight distribution and so we can initialize the mixture appropriately. Commonly, we initialize the network weights so they are uniformly distributed over an interval $[-W, W]$. In this case we may initialize the means of the gaussians so they are spaced evenly over the interval $[-W, W]$, and set all of the variances equal to the spacing between adjacent means and the mixing proportions equal to each other. This ensures that each component in the mixture initially has the same total responsibility over the entire set of weights,[7] and also produces sufficient counterbalance between the forces from each gaussian so most of the weights in the network initially receive very little net force from the complexity measure. This initialization procedure is used for all of the simulations discussed in this paper.

There is one additional trick used in the simulations discussed in this paper. The variances of the mixture components, σ_j^2, must of course be restricted to be positive and in addition if the variance of any component is allowed to approach 0 too closely the likelihood may become unbounded. To maintain the positivity constraint and at the same time make it difficult for the variance of a component to approach 0, we define the variance of the components in terms of a set of auxiliary variables:

$$\sigma_j^2 = e^{\gamma_j} \tag{4.6}$$

where the value of γ_j is unrestricted. The gradient descent is performed on the set of γ_j rather than directly on the σ_j.[8]

[6]Any method of gradient descent could be used in the parameter update, however the conjugate gradient technique is quite fast and avoids the need to tune optimization parameters such as step size or momentum rate.

[7]There is a minor edge effect for the two most extreme components.

[8]The use of the γ_j may be thought of simply as a technique for getting a better conditioned optimization problem.

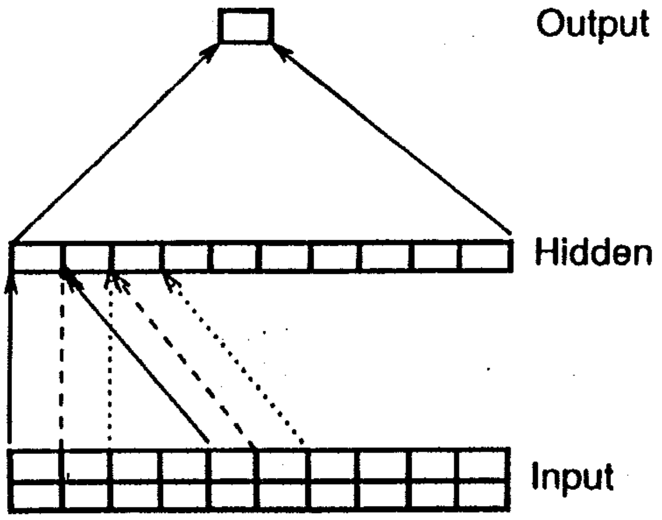

Figure 1: Shift detection network used for generalization simulations. The output unit was connected to a bias unit and the 10 hidden units. Each hidden unit was connected to the bias unit and to 8 input units, 4 from the first block of 10 inputs and the corresponding 4 in the second block of 10 inputs. The solid, dashed, and dotted lines show the group of input units connected to the first, second, and third hidden units, respectively.

5 Results on a Toy Problem

In this section, we report on some simulations which compare the generalization performance of networks trained using the cost criterion given in equation 4.1 to networks trained in three other ways:

- No cost term to penalize complexity.
- No explicit complexity cost term, but use of a validation set to terminate learning.
- The complexity cost term used by Weigend *et al.* (Weigend *et al.* (1990).[9]

The problem chosen for this comparison was a 20 input, one output shift detection network (see Fig. 1). The network had 20 input units, 10 hidden units, and a single output unit and contained 101 weights. The first 10 input units in this network were given a random binary pattern, and the second group of 10 input units were given the same pattern

[9]With a fixed value of λ chosen by cross validation.

Table 1: Summary of Generalization Performance of Five Different Training Techniques on the Shift Detection Problem.

Method	Train % correct	Test % correct
Backpropagation	100.0 ± 0.0	67.3 ± 5.7
Cross-validation	98.8 ± 1.1	83.5 ± 5.1
Weight decay	100.0 ± 0.0	89.8 ± 3.0
Soft-share — 5 component	100.0 ± 0.0	95.6 ± 2.7
Soft-share — 10 component	100.0 ± 0.0	97.1 ± 2.1

circularly shifted by 1 bit left or right. The desired output of the network was $+1$ for a left shift and -1 for a right shift.

A data set of 2400 patterns was created by randomly generating a 10 bit string, and choosing with equal probability to shift the string left or right.[10] The data set was divided into 100 training cases, 1000 validation cases, and 1300 test cases. The training set was deliberately chosen to be very small ($< 5\%$ of possible patterns) to explore the region in which complexity penalties should have the largest impact.

Networks were trained with a conjugate gradient technique and, except for the networks trained using cross-validation, training was stopped as soon as 100% correct performance was achieved on the training set. For the networks trained with cross-validation, training was stopped when three consecutive weight updates[11] produced an increase in the error on the validation set and the weights were then reset to the weights which achieved the lowest error on the validation set before testing for generalization. For the technique described in Weigend *et al.* (1990), $\lambda = 5.0 \times 10^{-5}$ in all simulations.[12] Simulations using equation 4.1 were performed with gaussian mixtures containing 5 and 10 components. Each component had its own mean (μ_j), variance (σ_j^2), and mixing proportion (π_j). The parameters of the mixture distribution were continuously reestimated as the weights were changed as was the normalizing factor for the squared error (σ_y).

Ten simulations were performed with each method, starting from ten different initial weight sets (i.e., each method used the same ten initial weight configurations). The simulation results are summarized in Table 1. The first column indicates the method used in training the network, while the second and third columns present the performance on the training and test sets respectively (plus or minus one standard deviation).

[10]Since there are only 2048 distinct cases, this set of 2400 did contain some duplicates.

[11]A weight update refers to the final weight change accepted at the end of a single line search.

[12]This value was selected by performing simulations with λ ranging between 1.0×10^{-1} and 1.0×10^{-9} and choosing the value of λ that gave the best performance on the cross-validation set.

All three methods which employ some form of weight modeling performed significantly better on the test set than networks trained using backpropagation without a complexity penalty ($p \gg 0.9999$).[13] The network trained with cross-validation also performs better than a network trained without a complexity penalty ($p > 0.995$). The two soft-sharing models perform better than cross-validation ($p > 0.995$ for the 5 component mixture, $p > 0.999$ for the 10 component mixture). The evidence that the form of weight decay of Weigend *et al.* is superior to cross-validation on this problem is very weak ($p < 0.9$). Finally, the two soft-sharing models are significantly better than the weight decay model ($p > 0.999$ for the 10 component mixture and $p > 0.99$ for the 5 component mixture). The difference in the performance of the 5 and 10 component mixtures is not significant.

A typical set of weights learned by the soft-sharing model with a 10 component mixture is shown in Figure 2 and the final mixture density is shown in Figure 3. The weight model in this case contains four primary weight clusters: large magnitude positive and negative weights and small magnitude positive and negative weights. These four distinct classes may also be seen clearly in the weight diagram (Fig. 2).

What is perhaps most interesting about the mixture probability density shown in Figure 3 is that it *does not* have a significant component with mean 0. The classical assumption that the network contains a large number of inessential weights that can be eliminated to improve generalization is not appropriate for this problem and network architecture. This may explain why the weight decay model used by Weigend *et al.* (1990) performs relatively poorly in this situation.

6 Results on a Real Problem

The second task chosen to evaluate the effectiveness of the cost criterion of equation 4.1 was the prediction of the yearly sunspot average from the averages of previous years. This task has been well studied as a time-series prediction benchmark in the statistics literature (Priestley 1991a,b) and has also been investigated by Weigend *et al.* (1990) using a cost criterion similar to the one discussed in Section 2.

The network architecture used was identical to the one used in the study by Weigend *et al.* The network had 12 input units, which represented the yearly average from the preceding 12 years, 8 hidden units, and a single output unit, which represented the prediction for the average number of sunspots in the current year. Yearly sunspot data from 1700 to 1920 was used to train the network to perform this one-step prediction task, and the evaluation of the network was based on data from 1921 to

[13]All statistical comparisons are based on a t test with 19 degrees of freedom. p denotes the probability of rejecting the hypothesis that the two samples being compared have the same mean value.

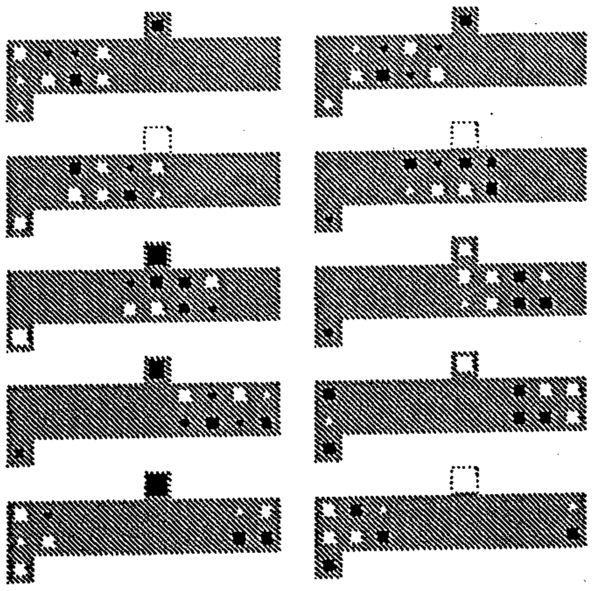

Figure 2: A diagram of weights discovered for the shift problem by a model which employed a 10 component mixture for the complexity cost. Black squares are negative weights and white squares are positive, with the size of the square proportional to the magnitude of the weight. Weights are shown for all 10 hidden units. The bottom row of each block represents the bias, the next two rows are the weights from the 20 input units, and the top row is the weight to the output unit.

1955.[14] The evaluation of prediction performance used the *average relative variance* (*arv*) measure discussed in Weigend *et al.* (1990):

$$arv(S) = \frac{\sum_{k \in S}(\text{target}_k - \text{prediction}_k)^2}{\sum_{k \in S}(\text{target}_k - \text{mean}_S)^2} \tag{6.1}$$

where S is a set of target values and mean_S is the average of those target values.

Simulations were performed using the same conjugate gradient method used in the previous section. Complexity measures based on gaussian mixtures with 3 and 8 components were used and 10 simulations were performed with each (using the same training data but different initial weight configurations). The results of these simulations are summarized in Table 2 along with the best result obtained by Weigend *et al.* (1990) (WRH), the bilinear autoregression model of Tong and Lim (1980)

[14]The authors wish to thank Andreas Weigend for providing his version of this data to work with.

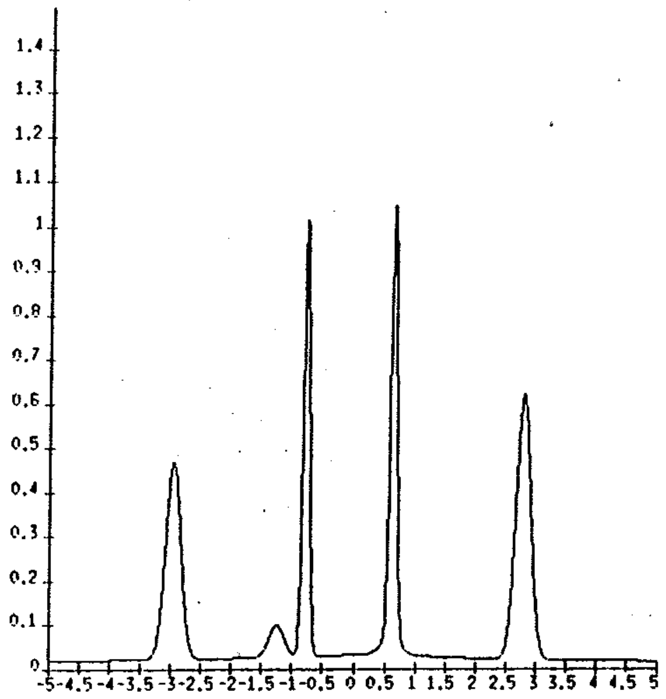

Figure 3: Final mixture probability density for the set of weights shown in Figure 2. Five of the components in the mixture can be seen as distinct bumps in the probability density. Of the remaining five components, two have been eliminated by having their mixing proportions go to zero and the other three are very broad and form the baseline offset of the density function.

(TAR),[15] and the multilayer RBF network of He and Lapedes (1991) (RBF). All figures represent the *arv* on the test set. For the mixture complexity models, this is the *average* over the 10 simulations, plus or minus one standard deviation.

Since the results for the models other than the mixture complexity trained networks are based on a single simulation it is difficult to assign statistical significance to the differences shown in Table 2. We may note however, that the difference between the 3 and 8 component mixture complexity models is significant ($p > 0.95$) and the differences between the 8 component model and the other models are much larger.

Weigend *et al.* point out that for time series prediction tasks such as the sunspot task a much more interesting measure of performance is the ability of the model to predict more than one time step into the future.

[15]This was the model favored by Priestley (1991a) in a recent evaluation of classical statistical approaches to this task.

Table 2: Summary of average relative variance of five different models on the one-step sunspot prediction problem.

Method	Test *arv*
TAR	0.097
RBF	0.092
WRH	0.086
Soft-share — 3 Comp.	0.077 ± 0.0029
Soft-share — 8 Comp.	0.072 ± 0.0022

One way to approach the multistep prediction problem is to use *iterated single-step prediction*. In this method, the predicted output is fed back as input for the next prediction and all other input units have their values shifted back one unit. Thus the input typically consists of a combination of actual and predicted values.

We define the predicted value for time t, obtained after I iterations to be $\hat{x}_{t,I}$. The prediction error will depend not only on I but also on the time $(t - I)$ when the iteration was started. In order to account for both effects, Weigend *et al.* suggested the *average relative I-times iterated prediction variance* as a performance measure for iterated prediction:

$$\frac{1}{\hat{\sigma}^2}\frac{1}{M}\sum_{m=1}^{M}(x_{m+I} - \hat{x}_{m+I,I}) \tag{6.2}$$

where M is the number of different start times for iterated prediction and $\hat{\sigma}$ is the estimated standard deviation of the set of target values. In Figure 4 we plot this measure (computed over the test set from 1921 to 1955) as a function of the number of prediction iterations for the simulations using the 3 and 8 component complexity measures, the Tong and Lim model (TAR), and the model from Weigend *et al.*, which produced the lowest single step *arv* (WRH). The plots for the 3 and 8 component complexity models are the averages over 10 simulations with the error bars indicating the plus or minus one standard deviation intervals. Once again, the differences between the 3 and 8 component models are significant for all numbers of iterations.

The differences between the adaptive gaussian complexity measure and the fixed complexity measure used by Weigend *et al.* are not as dramatic on the sunspot task as they were in the shift detection task. The explanation for this may be seen in Figures 5 and 6, which show a typical set of weights learned by the soft-sharing model with 8 mixture components and the corresponding final mixture probability density. The distinct weight groups seen clearly in the shift detection task (Fig. 2) are not as apparent in the weights for the sunspot task and the final weight distribution for the sunspot task is very smeared out except for one very strong sharp component near 0. It is clear that the fixed model assumed

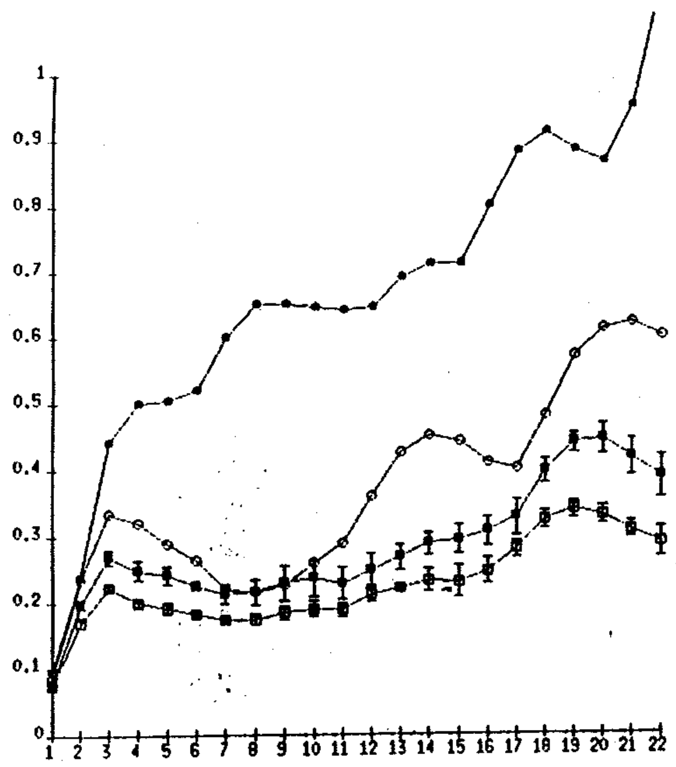

Figure 4: Average relative *I*-times iterated prediction variance versus number of prediction iterations for the sunspot time series from 1921 to 1955. Closed circles represent the TAR model, open circles the WRH model, closed squares the 3 component complexity model, and open squares the 8 component complexity model. One deviation error bars are shown for the 3 and 8 component complexity models.

by Weigend *et al.* is much more appropriate for the sunspot prediction task than it was for the shift detection task.

7 A Minimum Description Length Perspective

A number of authors have suggested that when attempting to approximate an unknown function with some parametric approximation scheme (such as a network), the proper measure to optimize combines an estimate of the cost of the misfit with an estimate of the cost of describing the parametric approximation (Akaike 1973; Rissanen 1978; Barron and Barron 1988). Such a measure is often referred to as a minimum description

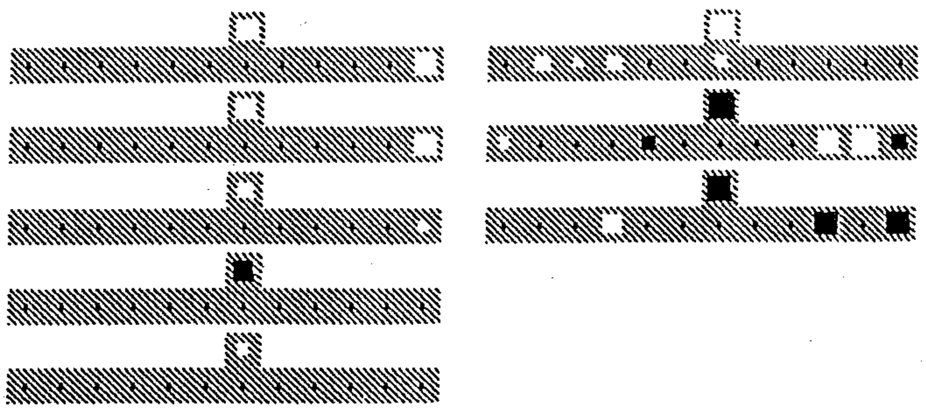

Figure 5: A diagram of weights discovered for the sunspot prediction problem by a model which employed an 8 component mixture for the complexity cost. Weights are shown for all 8 hidden units. For each unit, the weights coming from the 12 inputs are shown in a row with the single weight to the output immediately above the row. The biases of the hidden units, which are not shown, were, with one exception, small negative numbers very close in value to most of the other weights in the network. The first three units in the left column all represent the simple rule that the number of sunspots depends on the number in the previous year. The last two units in this column compute a simple moving average. The three units on the right represent more interesting rules. The first captures the 11 year cycle, the second recognizes when a peak has just passed, and third appears to prevent the prediction from rising too soon if a peak happened 9 years ago and the recent activity is low.

length criterion (MDL), and typically has the general form

$$MDL = \sum_{\text{messages}} -\log p(\text{message}) + \sum_{\text{parameters}} -\log p(\text{parameter}).$$

For a supervised network, the parameters are the weights and the messages are the desired outputs. If we assume that the output errors are gaussian and that the weights are encoded using a mixture of gaussians probability model the description length is approximated by equation 4.1.

The expression in equation 4.1 does not include the cost of encoding the means and variances of the mixture components or the mixing proportions of the mixture density. Since the mixture usually contains a small number of components (fewer than 10 usually) and there are only three parameters associated with each component, the cost of encoding these parameters is negligible compared to the cost of encoding the

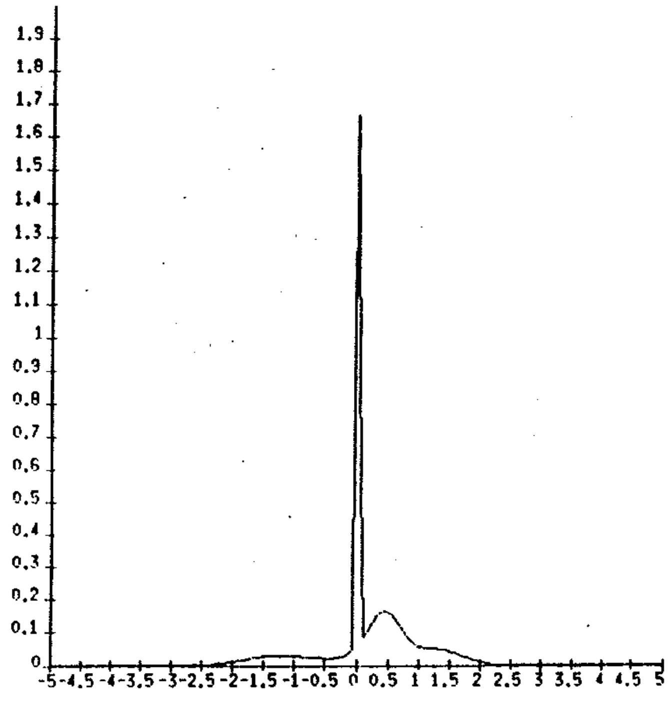

Figure 6: Final mixture probability density for the set of weights shown in Figure 5. The density is dominated by a narrow component centered very near zero, with the remaining components blending into a skewed distribution with a peak around 0.5.

weights in most networks of interest.[16] In addition, since the *number* of components in the distribution does not change during the optimization, if the component parameters are all encoded with the same fixed precision, the cost of the mixture parameters is simply a constant offset, which is ignored in the optimization.[17]

There is one important aspect of estimating the cost of describing a weight that we have ignored. We have assumed that the cost of a weight is the negative logarithm of a probability *density* function evaluated at the weight value, but this ignores the accuracy with which the weight must be described. We are really interested in the probability *mass* of a particular small interval of values for the parameter, and this means that we should integrate our density function over this interval to estimate the cost of each weight. We have implicitly assumed that this integration

[16]In order to provide enough data to fit the mixture density, one should have an order of magnitude more weights than components in the mixture.

[17]This ignores the possibility of not encoding the parameters of components whose mixing proportions approach 0.

region has the same (infinitesimal) width for every weight, and so the probability of a weight is simply proportional to the density function. This ignores the fact that most networks are generally much more sensitive to small changes in some weight values than others, so some weights need to be encoded more accurately than others.[18]

The sensitivity of a network to a small change in a weight is determined by the curvature of the error surface. One could evaluate the curvature by computing the Hessian and make the width of the integration region for each weight inversely proportional to the curvature along each weight dimension. To be perfectly accurate, one would need to integrate the joint probability density function for all of the weights over a region determined by the complete Hessian (since the directions of maximum curvature are often not perfectly aligned with the weight axes). This process would be computationally very costly, and an adequate approximation might be obtainable by using a diagonal approximation to the Hessian and treating each weight independently (as advocated by le Cun *et al.* 1990). We see no reason why our method of estimating the probability density should not be combined with a method for estimating the integration interval. For small intervals, this is particularly easy since the probability mass is approximately the width of the interval times the height of the density function so these two terms are additive in the log probability domain.

8 A Bayesian Perspective

As a number of authors have recently pointed out (Weigend *et al.* 1991; Nowlan 1991; MacKay 1991; Buntine and Weigend 1991), equation 1.1 can be derived from the principles of Bayesian inference. If we have a set of models $M_1, M_2, \ldots$, which are competing to account for the data, Bayesian inference is concerned with how we should update our belief in the relative plausibility of each of these models in light of the data D. If $P(M_i \mid D)$ is the plausibility of model M_i given we have observed D, Bayes rule states

$$P(M_i \mid D) = \frac{P(D \mid M_i)P(M_i)}{P(D)} \tag{8.1}$$

where $P(D \mid M_i)$ is a measure of how well model i predicts the data and $P(M_i)$ is our belief in the plausibility of model i before we have seen any data. Here $P(D)$ is simply a normalizing factor to ensure our beliefs add up to one. If we are only interested in comparing alternate models, $P(D)$ can be ignored and in the log domain equation 8.1 becomes equation 1.1 with the data-misfit cost equal to $\log P(D \mid M_i)$ and the complexity

[18]Other things being equal, we should prefer networks in which the outputs are less sensitive to the precise weight values, since then the weight values can be encoded imprecisely without causing large output errors.

cost equal to $\log P(M_i)$. If we are only considering a single network architecture, $P(M_i)$ becomes a prior distribution over the set of possible weights.[19]

What equation 8.1 highlights is that in the Bayesian framework our complexity cost should be *independent* of our data. This is certainly true when the complexity is the sum of the squares of the weights, and also holds for models such as the one used by Weigend *et al.* However, the mixture densities discussed in this paper are clearly not independent of the data and cannot be regarded as classical Bayesian priors.

The complexity cost we are using corresponds more closely to a Bayesian *hyperprior* (Jaynes 1986; Gull 1988). We have specified a particular family of distributions from which the prior will be drawn but have left the parameters of the prior (π_j, μ_j, σ_j) undetermined. Members of this family of distributions have the common feature of favoring sets of weights in which the weights *in a set* are clustered about a small number of values.[20] When using a hyperprior, we can deal with the *hyperparameters* either by marginalizing over them (in effect, integrating them out) (Buntine and Weigend 1991), or by allowing the data (i.e., the weights) to determine their values *a posteriori*.[21] We have used this second approach, which is advocated by Gull (1988), who has shown that the use of such flexible hyperpriors can lead to considerable improvement in the quality of image reconstructions (Gull 1989; Skilling 1989) compared to the use of more classical priors.

The trick of optimizing γ_j rather than σ_j (discussed at the end of Section 4) may also be justified within the Bayesian framework. To estimate our hyperparameters, we should properly specify prior distributions for each. If these priors are uninformative,[22] then the estimated values of the hyperparameters are determined entirely by the data. A parameter like σ_j is known as a *scale* parameter (it affects the width of the distribution) while parameters like μ_j are known as *location* parameters (they affect the position of the distribution). (See Jeffreys 1939 for further discussion.) An uninformative prior for a location parameter is uniform in the parameter, but an uninformative prior for a scale parameter is uniform in the log of the parameter (i.e., uniform in γ_j rather than σ_j, Gull 1988). It is more consistent from this perspective to treat γ_j and μ_j similarly, rather than σ_j and μ_j.

[19]Much more interesting results are obtained when we apply this framework to making choices among many architectures, see MacKay for some elegant examples (MacKay 1991).

[20]The locations of these clusters will generally be different for different sets of weights.

[21]In principle, both approaches will lead to the same posterior distribution over the weights and the same ultimate choice of weights for the network. The difference lies in whether we are searching over a joint space of weights and hyperparameters or using prior analytic simplifications to reduce the search to some manifold in weight space alone.

[22]A prior that contains no initial bias except for a possible range constraint.

9 Summary

The simulations we have described provide evidence that the use of a more sophisticated model for the distribution of weights in a network can lead to better generalization performance than a simpler form of weight decay, or techniques that control the learning time. The better generalization performance comes at the cost of greater complexity in the optimization of the weights. The effectiveness of the technique is likely to be somewhat problem dependent, but one advantage offered by the more sophisticated model is its ability to automatically adapt the model of the weight distribution to individual problems.

Acknowledgments

This research was funded by grants from the Ontario Information Technology Research Center and the Canadian Natural Science and Engineering Research Council and the Howard Hughes Medical Institute. Hinton is the Noranda fellow of the Canadian Institute for Advanced Research.

References

Akaike, H. 1973. Information theory and an extension of the maximum likelihood principle. In *Proceedings 2nd International Symposium on Information Theory*, B. N. Petrov and F. Csaki, eds., pp. 267–281. Akademia Kiado, Budapest, Hungary.

Barron, A. R., and Barron, R. L. 1988. Statistical learning networks: A unifying view. In *1988 Symposium on the Interface: Statistics and Computing Science*, Reston, Virginia, April 21–23.

Baum, E. B., and Haussler, D. 1989. What size net gives valid generalization? *Neural Comp.* **1**, 151–160.

Buntine, W. L., and Weigend, A. S. 1991. Bayesian back-propagation. *Complex Systems* 5(6), 603–643.

Dempster, A. P., Laird, N. M., and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Proc. R. Stat. Soc. Ser. B* **39**, 1–38.

Gull, S. F. 1988. Bayesian inductive inference and maximum entropy. In *Maximum Entropy and Bayesian Methods in Science and Engineering*, G. J. Ericksor and C. R. Smith, eds. Kluwer Academic, Dordrecht, The Netherlands.

Gull, S. F. 1989. Developments in maximum entropy data analysis. In *Maxi mum Entropy and Bayesian Methods (8th Workshop)*, J. Skilling, ed., pp. 53–71 Kluwer Academic, Dordrecht, The Netherlands.

He, X., and Lapedes, A. 1991. *Nonlinear Modelling and Prediction by Successiv Approximation Using Radial Basis Functions*. Tech. Rep. LA-UR-91-1375, Lo Alamos National Laboratory.

Hinton, G. E. 1986. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 1–12. Erlbaum, Hillsdale, NJ.

Hinton, G. E. 1987. Learning translation invariant recognition in a massively parallel network. In *Proc. Conf. Parallel Architectures and Languages Europe*, pp. 1–13. Eindhoven, The Netherlands.

Jaynes, E. T. 1986. Bayesian methods: General background. In *Maximum Entropy and Bayesian Methods in Applied Statistics*, J. H. Justice, ed., pp. 1-25. Cambridge University Press, Cambridge.

Jeffreys, H. 1939. *Theory of Probability*. Oxford University Press, Oxford. Later editions 1948, 1961, 1983.

Kohonen, T. 1977. *Associative Memory: A System-Theoretical Approach*. Springer, Berlin.

Lang, K. J., Waibel, A. H., and Hinton, G. E. 1990. A time-delay neural network architecture for isolated word recognition. *Neural Networks* **3**, 23–43.

LeCun, Y. 1987. Modèles connexionnistes de l'apprentissage. Ph.D. thesis, Université Pierre et Marie Curie, Paris, France.

LeCun, Y. 1989. *Generalization and Network Design Strategies*. Tech. Rep. CRG-TR-89-4, University of Toronto.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. 1990. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, ed., pp. 396–404. Morgan Kaufmann, San Mateo, CA.

LeCun, Y., Denker, J., Solla, S., Howard, R. E., and Jackel, L. D. 1990. Optimal brain damage. In *Advances in Neural Information Processing Systems* **2**, 598–605, D. S. Touretzky, ed. Morgan Kaufmann, San Mateo, CA.

MacKay, D. J. C. 1991. Bayesian modeling and neural networks. Ph.D. thesis, Computation and Neural Systems, California Institute of Technology, Pasadena, CA.

McLachlan, G. J., and Basford, K. S. 1988. Chapters 1 and 2. In *Mixture Models: Inference and Applications to Clustering*, G. J. McLachlan and K. E. Basford, eds., pp. 1–69. Marcel Dekker, New York.

Morgan, N., and Bourlard, H. 1989. *Generalization and Parameter Estimation in Feedforward Nets: Some Experiments*. Tech. Rep. TR-89-017, International Computer Science Institute, Berkeley, CA.

Mozer, M. C., and Smolensky, P. 1989. Using relevance to reduce network size automatically. *Connection Sci.* **1**(1), 3–16.

Nowlan, S. J. 1991. Soft competitive adaptation: Neural network learning algorithms based on fitting statistical mixtures. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Plaut, D. C., Nowlan, S. J., and Hinton, G. E. 1986. *Experiments on Learning by Backpropagation*. Tech. Rep. CMU-CS-86-126, Carnegie Mellon University, Pittsburgh, PA.

Priestley, M. B. 1991a. *Non-linear and Non-stationary Time Series Analysis*. Academic Press, San Diego.

Priestley, M. B. 1991b. *Spectral Analysis and Time Series*. Academic Press, San Diego.

Rissanen, J. 1978. Modeling by shortest data description. *Automatica* 14, 465–471.

Rumelhart, D. E., McClelland, J. L., and the PDP research group. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* Vols. I and II. MIT Press, Cambridge, MA.

Skilling, J. 1989. Classic maximum entropy. In *Maximum Entropy and Bayesian Methods (8th Workshop),* J. Skilling, ed. Kluwer Academic, Dordrecht, The Netherlands.

Tong, H., and Lim, K. S. 1980. Threshold autoregression, limit cycles, and cyclical data. *J. R. Stat. Soc. Ser. B* 42, 245–253.

Weigend, A. S., Rumelhart, D. E., and Huberman, B. A. 1990. Predicting the future: A connectionist approach. In *Proceedings of the 1990 Connectionist Models Summer School,* T. J. Sejnowski, G. E. Hinton, and D. S. Touretzky, eds., pp. 105–116. Morgan Kaufmann, San Mateo, CA.

Weigend, A. S., Rumelhart, D. E., and Huberman, B. A. 1991. Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing Systems* 3, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, eds., pp. 875–882. Morgan Kaufmann, San Mateo, CA.

Received 15 October 1991; accepted 2 December 1991.