Prediction Intervals for Neural Networks via Nonlinear Regression

Richard D. De Veaux
Department of Mathematics
Williams College
Williamstown, MA 01267
(deveaux@williams.edu)

Jennifer Schumi
Department of Statistics
Iowa State University
Ames, IA 50010
(jschumi@iastate.edu)

Jason Schweinsberg
Department of Statistics
University of California at Berkeley
Berkeley, CA 94720
(jason@stat.berkeley.edu)

Lyle H. Ungar
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389
(ungar@cis.upenn.edu)

Standard methods for computing prediction intervals in nonlinear regression can be effectively applied to neural networks when the number of training points is large. Simulations show, however, that these methods can generate unreliable prediction intervals on smaller datasets when the network is trained to convergence. Stopping the training algorithm prior to convergence, to avoid overfitting, reduces the effective number of parameters but can lead to prediction intervals that are too wide. We present an alternative approach to estimating prediction intervals using weight decay to fit the network and show via a simulation study that this method may be effective in overcoming some of the shortcomings of the other approaches.

KEY WORDS: Backpropagation; High-dimensional data; Nonparametric regression; Smoothing.

Multilayer feedforward neural networks are flexible statistical models that are widely used to model high-dimensional, nonlinear data. [For a general introduction to the field of neural networks for statisticians, see either Bishop (1995) or Ripley (1996)]. These are "black box" models, so called because the parameters are generally uninterpretable. This is not a problem for many applications of neural networks in which the emphasis is on prediction rather than on model building and process understanding. For example, the post office might want to sort hand-written zip codes by machine automatically. If an algorithm is able to do this effectively and accurately, the exact model used is of little interest.

Loosely speaking, with enough parameters, these networks are able to approximate any reasonable function. [For a precise definition and the necessary regularity conditions, see Hornik, Stinchcombe, and White (1989)]. This is obviously a good property because one wants to fit the data at hand, the so-called "training data," as well as possible. Statisticians, however, are well aware of the dangers of overfitting to the training data. The ability to generalize, to predict previously unseen data (the "test data"), is in direct conflict with the ability to fit the training data accurately. This phenomenon, known to statisticians as the bias–variance trade-off in model selection, has a long history and a variety of methods to help deal with it.

It is this problem that is often overlooked by users of neural networks and from which the conflict between them and statisticians often arises. In many software realizations of neural networks, the method for choosing the number of parameters is not well implemented. To avoid overfitting, users are often left on their own to perform some sort of cross-validation. Worse, many vendors of neural-network software fail to mention the generalization problem at all and use faithful reproduction of the *training* data as a selling point for the algorithms.

Unfortunately, many users, in spite of (or indeed because of) this, take the predictions given by the network on faith, without regard to the uncertainty inherent in the predictions. As with regression methods, neural-network prediction accuracy varies with data density and noise. Methods of calculating prediction intervals based on resampling have been developed (Baxt and White 1995) but suffer major limitations due to the computational demands because they often require months of computer time.

The multilayer feedforward neural-network model can be viewed as a nonlinear regression model. Thus, standard asymptotic theory from nonlinear regression can, in principal, be applied (Hwang and Ding 1997). Simulation results to be presented here, however, indicate that a straightforward application of the theory works poorly in practice. We will show that a regularization method known as weight decay may overcome this problem. The derivation of these prediction intervals based on weight decay is straightforward and, in fact, has appeared (Ripley 1994) without derivation. Our objective in this article is twofold. We first present the derivation of these intervals and then proceed to test them via simulation against intervals derived from two

TECHNOMETRICS, NOVEMBER 1998, VOL. 40, NO. 4

other popular methods of regularizing networks—namely, architecture choice and early stopping.

The remainder of this article is divided into seven sections. Sections 1 and 2 briefly describe multilayer feedforward networks and review the asymptotic theory for calculating prediction intervals in nonlinear regression. Section 3 describes the experimental design used in our Monte Carlo simulation study and the results that indicate the inadequacies of the prediction intervals obtained by architecture choice and early stopping. We show in Section 4 that weight decay can also provide prediction intervals, and in Section 5 we show that this method tends to give substantially better prediction intervals than the other methods in our simulation. We illustrate the weight-decay intervals on a real data example in Section 6, where we compare them with prediction intervals obtained from a generalized additive model fit, and we summarize our findings in Section 7.

1. MULTILAYER FEEDFORWARD NEURAL NETWORKS

Multilayer feedforward neural networks are multivariate statistical models, for L response variables, $y_1, \ldots, y_L$, based on J predictor variables, $x_1, \ldots, x_J$. The model has several "layers," each consisting of original or constructed variables. Usually there are three layers—the inputs (predictors), a group of constructed variables known as the hidden layer, and the outputs (responses). We will restrict discussion in the rest of this article to three-layer networks.

Each variable in a layer is referred to as a node (or neuron or processing unit). In general, a node takes as input a transformed linear combination of the outputs of the nodes in the layer below it. It then sends as output a transformation of itself that serves as one of the inputs to a node in the next layer. This process of transformation is known as "activation." The transformation functions, called activation or transfer functions, are usually taken to be sigmoidal (S-shaped) or linear.

To be more explicit, suppose we have J inputs, K hidden nodes, and L outputs. Let each node in the hidden layer be denoted by $a_k, k = 1, \ldots, K$. Each hidden node, a_k, is a linear combination of the input variables, x_j:

$$a_k = \sum_{j=1}^{J} w_{1jk} x_j + \theta_k.$$

The x_j are usually standardized but otherwise untransformed input variables (although in some software implementations one can transform the inputs as well). The θ_k term is a parameter that plays the role of the intercept with the unfortunate name of the "bias node." The w_{mjk} are parameters to be estimated called weights, where $m = 1, 2$ indicates the layer. We can get rid of the artificial distinction between weights and bias nodes by defining $x_0 = 1$ and starting the summation at 0 instead of 1:

$$a_k = \sum_{j=0}^{J} w_{1jk} x_j.$$

Each node is then transformed by the activation function $g(\ )$. (In the literature these functions are often denoted by the unfortunate choice of notation $\sigma(\ )$ because of their sigmoidal shape.) We denote the output of the node a_k, by $z_k = g(a_k)$. Then we form a linear combination of these outputs: $b_l = \sum_{k=0}^{K} w_{2kl} z_k$, where $z_0 = 1$. The lth response y_l is then a transformation of the b_l: $y_l = \tilde{g}(b_l)$, where we have used (following Bishop 1995) $\tilde{g}(\ )$ for the activation function to underscore the fact that the activation function for the response may be different from that of the hidden layer. Putting this all together gives

$$y_l = \tilde{g}\left(\sum_{k=0}^{K} w_{2kl} g\left(\sum_{j=0}^{J} w_{1jk} x_j\right)\right), \tag{1}$$

or, leaving the bias nodes explicitly in the equation,

$$y_l = \tilde{g}\left(\sum_{k=1}^{K} w_{2kl} g\left(\sum_{j=1}^{J} w_{1jk} x_j + \theta_{1j}\right) + \theta_{2k}\right), \tag{2}$$

Thus, the response y_l is just a transformed linear combination of transformed linear combinations of the predictors. It is this flexible form with many parameters that gives the network its universal approximation property—that is, the ability to fit a wide variety of functions.

For the hidden layer, the sigmoid for the activation function is usually chosen to be the logistic function, $g(x) = 1/(1 + e^{-x})$, or the hyperbolic tangent function, $g(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$. For the output layer, the choice of activation function depends on the response. If the response is dichotomous, or bounded, the activation function $\tilde{g}$ is usually taken to be sigmoidal as well. Otherwise, it is taken to be linear (the identity), leaving the response as a direct linear combination of the outputs of the hidden nodes. The predictions are given by Equation (1) after the parameters (the weights including the bias nodes), w, are estimated. For the rest of the article we will use the logistic function for $g(\ )$, and because we consider only continuous responses, we will use the identity function for $\tilde{g}(\ )$.

The parameters w are estimated by minimizing the residual sum of squares (RSS) over all responses and observations:

$$\text{RSS} = \sum_{l} \sum_{i} (y_{li} - \hat{y}_{li})^2, \tag{3}$$

where i ranges over the observations and l over the responses. (For simplicity we shall restrict our attention to the case of a single output $l = 1$ for the rest of the article, although generalization to multiple responses is straightforward.) Because the equations are nonlinear in the parameters (unless, trivially, *both* activation functions are linear), the optimization is a nonlinear least squares problem, which can be solved by a variety of standard nonlinear optimization methods. A popular algorithm is the backpropagation algorithm, originally proposed by Werbos (1974), which is a variant of steepest descent. These networks are, in fact, often referred to as backpropagation networks, confusing

the model used with the optimization routine used for estimating the parameters. Many other methods, including conjugate gradient methods, are used today. Some of the more sophisticated programs use an adaptive fitting procedure, in which at each step a method is selected based on current information obtained from the numerical derivatives.

Because the estimation methods are numerical, they all require starting values, which are typically chosen to be random and small. Herein lies one reason for standardizing the inputs. If left in their original scales, an appropriate range of starting values would have to be separately determined for each input. Instead, the inputs are typically assumed to be standardized with random starting values selected from a Gaussian with mean 0 and small standard deviation. (Values of σ near .1 are common.)

Because the number of parameters is so large, overfitting will likely occur if the algorithm is continued until convergence. To prevent this, various strategies are employed. First, the number of parameters can be reduced. There is a large literature on this problem of model selection for neural networks, which is known in the neural-network community by the colorful name *optimal brain surgery,* in reference to the network's alleged similarity to the brain (e.g., see Hassibi, Stork, Wolff, and Watanabe 1994). The general problem of choosing the number of layers, number of neurons, and form of the activation functions is referred to as the choice of network architecture. Although there are many strategies for choosing the architecture, we shall consider the simple procedure of choosing the number of nodes in the hidden layer, which can be done by cross-validation. To do this, the data are repeatedly divided into two sets, one for training (estimation) and the other for testing (generalization). The number of nodes is chosen so as to minimize the average (over many random choices of training and test sets) of the residual sum of squared error on the test sets.

A second method starts with many nodes, deliberately overparameterizing the problem, ensuring enough flexibility to fit the data extremely accurately if the network were allowed to train until convergence. To avoid overfitting, rather than reduce the number of parameters, the optimization routine used to estimate the weights is simply stopped early, before convergence. Finding the optimal number of iterations is again accomplished by cross-validation. This time, however, the RSS's on both the training and test sets are monitored as the algorithm proceeds. For the training sets, the RSS decreases, essentially monotonically, as the number of iterations increases and the fitting algorithm eventually converges. For the test sets, however, a different pattern emerges. The RSS first decreases as the weights move away from their initial random starting values. Eventually, however, the algorithm starts to overfit to the training data and the RSS on the test set increases. It continues to increase as the algorithm is allowed to fit the training data better and better until convergence. After convergence on the training data is obtained, a plot of the average RSS on the test sets versus the number of iterations is examined. The number of iterations that minimizes the average RSS on the test sets is chosen. The weights are now reestimated, using the entire dataset but stopping the algorithm at the chosen number of iterations. This somewhat ad hoc procedure is known as the early-stopping method.

The third method, known as weight decay, adds a penalty term to the RSS before minimization:

$$\sum_i (y_i - \hat{y}_i)^2 + \alpha \sum w^2, \tag{4}$$

where the last sum is over all the weights in the network. The parameter α, known as the weight-decay parameter, can also be found by cross-validation. The penalization is reminiscent of other regularization methods in regression such as ridge regression and smoothing spline fitting. There is also a Bayesian interpretation of weight decay, which we present briefly in Section 4. In Section 2 we proceed to develop the prediction-interval theory for the unpenalized RSS criterion before returning to weight decay in Section 4.

2. PREDICTION–INTERVAL THEORY FOR NONLINEAR REGRESSION

The problem of estimating the parameters in Equation (1) using the least squares objection function (3) can be viewed as a nonlinear regression problem. The standard methods that give asymptotic prediction intervals for nonlinear regression should, in theory, apply directly to neural networks. This section briefly reviews that theory. Although the derivation is straightforward, we have included it here for completeness. [See also Seber and Wild (1989) for a more thorough treatment of the subject.]

Consider a nonlinear regression model defined by

$$\mathbf{y} = f(\mathbf{X}, \mathbf{w}) + \boldsymbol{\varepsilon}, \tag{5}$$

where $\mathbf{w}$ is a parameter vector and $\boldsymbol{\varepsilon} \sim \mathrm{N}(\mathbf{0}, \sigma^2\mathbf{I})$. [For the neural network, f is given by the right side of Eq. (1).] We denote the estimated parameters by $\hat{\mathbf{w}}$, choosing $\hat{\mathbf{w}}$ to minimize the sum of squared residuals $\mathrm{RSS} = (\mathbf{y} - \hat{\mathbf{y}})'(\mathbf{y} - \hat{\mathbf{y}})$, where $\hat{\mathbf{y}}$ is the vector of predicted values.

Near the true parameter values $\mathbf{w}^*$, f can be approximated by a Taylor series: $\hat{\mathbf{y}} = f(\hat{\mathbf{w}}) f(\mathbf{w}^*) + \mathbf{J}(\hat{\mathbf{w}} - \mathbf{w}^*)$, where $\mathbf{J}$ is a matrix whose ijth entry is $\partial f(\mathbf{x}_i)/\partial w_j$, evaluated at the true parameter vector, $\mathbf{w}^*$, and where $\mathbf{x}_i$ is the ith row of $\mathbf{X}$, corresponding to the ith data point. (Note that the standard methods for estimating the weights in neural networks, such as backpropagation and conjugate gradient methods, calculate $\mathbf{J}$ as part of their procedures.) Using this, we approximate the residual sum of squares RSS by

$$\begin{aligned} \mathrm{RSS} &\approx (\mathbf{y} - f(\mathbf{w}^*) + \mathbf{J}\mathbf{w}^* - \mathbf{J}\hat{\mathbf{w}})'(\mathbf{y} - f(\mathbf{w}^*) \\ &\qquad + \mathbf{J}\mathbf{w}^* - \mathbf{J}\hat{\mathbf{w}}) \\ &= (\mathbf{k} - \mathbf{J}\hat{\mathbf{w}})'(\mathbf{k} - \mathbf{J}\hat{\mathbf{w}}), \end{aligned} \tag{6}$$

where

$$\mathbf{k} = \mathbf{y} - f(\mathbf{w}^*) + \mathbf{J}\mathbf{w}^* = \boldsymbol{\varepsilon} + \mathbf{J}\mathbf{w}^*. \tag{7}$$

It follows from setting the derivatives with respect to $\hat{\mathbf{w}}$ equal to 0 that

$$\hat{\mathbf{w}} \approx (\mathbf{J}'\mathbf{J})^{-1}\mathbf{J}'\mathbf{k}. \tag{8}$$

From this, the asymptotic variance of the estimated parameters is

$$\text{var}(\hat{\mathbf{w}}) \approx \sigma^2(\mathbf{J}'\mathbf{J})^{-1}. \tag{9}$$

Given the estimated parameters $\hat{\mathbf{w}}$ and a new data point $\mathbf{x}_0$, we approximate the predicted value of f at $\mathbf{x}_0$ by

$$\hat{y}_0 = f(\mathbf{x}_0, \hat{\mathbf{w}}) \approx f(\mathbf{x}_0, \mathbf{w}^*) + \mathbf{g}_0'(\hat{\mathbf{w}} - \mathbf{w}^*), \tag{10}$$

where $\mathbf{g}_0$ is a vector whose ith entry is $\partial f(\mathbf{x}_0)/\partial w_j$, evaluated at the true parameter vector $\mathbf{w}^*$. It follows that an asymptotic estimate of the prediction variance at $\mathbf{x}_0$ is given by

$$\text{var}(\hat{y}|\mathbf{x}_0) \approx \sigma^2\mathbf{g}_0'(\mathbf{J}'\mathbf{J})^{-1}\mathbf{g}_0. \tag{11}$$

We can estimate $\mathbf{J}$ and $\mathbf{g}_0$ consistently by evaluating the partial derivatives at $\hat{\mathbf{w}}$. To estimate σ^2, we take the expected value of Equation (6) and find

$$\begin{aligned} E[\text{RSS}] &\approx E[(\mathbf{k} - \mathbf{J}\hat{\mathbf{w}})'(\mathbf{k} - \mathbf{J}\hat{\mathbf{w}})] \\ &\approx E[\mathbf{k}'(\mathbf{I} - \mathbf{J}(\mathbf{J}'\mathbf{J})^{-1}\mathbf{J}')'(\mathbf{I} - \mathbf{J}(\mathbf{J}'\mathbf{J})^{-1}\mathbf{J}')\mathbf{k}] \\ &= E[\mathbf{k}'(\mathbf{I} - \mathbf{J}(\mathbf{J}'\mathbf{J})^{-1}\mathbf{J}')\mathbf{k}] \\ &= \mathbf{w}^{*\prime}\mathbf{J}'(\mathbf{I} - \mathbf{J}(\mathbf{J}'\mathbf{J})^{-1}\mathbf{J}')\mathbf{J}\mathbf{w}^* \\ &\quad + \sigma^2\text{tr}(\mathbf{I} - \mathbf{J}(\mathbf{J}'\mathbf{J})^{-1}\mathbf{J}') \\ &= \sigma^2(n - p). \end{aligned} \tag{12}$$

Here, n is the number of data points and p is the number of parameters. Thus, an asymptotically unbiased estimate of σ^2 is

$$s^2 = \text{RSS}/(n - p). \tag{13}$$

Using the fact that $s\sqrt{(1 + \mathbf{g}_0'(\mathbf{J}'\mathbf{J})^{-1}\mathbf{g}_0)}$ has approximately a t_{n-p} distribution, we obtain prediction intervals of half width:

$$t_{n-p}s\sqrt{(1 + \mathbf{g}_0'(\mathbf{J}'\mathbf{J})^{-1}\mathbf{g}_0)}. \tag{14}$$

Hwang and Ding (1997) addressed the application of this theory to the multilayer feedforward neural networks of the form shown in Equation (1). They showed that, although the individual parameters (the weights in the network) are not identifiable, the prediction interval half width shown in Equation (14) is still asymptotically valid when the network is trained to convergence.

Other methods for computing confidence intervals in nonlinear regression have been proposed. A comparison of three such methods was given by Donaldson and Schnabel (1987), who showed that the alternatives tend to perform better in simulations than the methods discussed in this section. Their study, however, was performed on models in which confidence intervals on relatively few parameters are of central concern. It is not evident how to apply these methods to the case at hand, in which there may be hundreds or even thousands of parameters but in which the interest is only in the predictions. For example, the likelihood method provides a confidence region for $\mathbf{w}$ given by the set $R = \{\mathbf{w}: (\text{SSR}(\mathbf{w}) - \text{SSR}(\hat{\mathbf{w}}))/s^2p < F_{p,n-p,1-\alpha}\}$. A prediction interval for y_0 is then given by $I = \{f(\mathbf{x}_0, \mathbf{w}) : \mathbf{w} \in R\}$. This method has the advantage of not relying on a linear approximation. In a model as complex as a neural network, however, it is computationally intensive to compute the set I because of the difficulty in determining, given any real number s, whether there exists a vector of weights $\mathbf{w} \in R$ such that $f(x_0, \mathbf{w}) = s$.

3. A MONTE CARLO SIMULATION STUDY FOR TESTING PREDICTION INTERVALS

3.1 Experimental Design and Procedure

To test the prediction intervals, we used a Monte Carlo simulation study in four factors. The four factors, each varied at two levels, were type of function (linear or nonlinear), number of predictors in the dataset (5 or 20), dataset size (low or high), and error size (σ = .1 or .25). The design was a 2^4 factorial design as shown in Table 1 replicated 100 times. For each replication, 1,000 values of the input variables were drawn at random from the uniform distribution on the unit hypercube. The same 100 training sets of 1,000 observations were used for all 16 settings, although for the low level of the dataset size factor we used only the first 200 or 400 observations, as will be explained. The errors were generated randomly from a Gaussian distribution with mean 0 and standard deviation .1 or .25 for each replicate, depending on the setting.

One hundred test sets were also generated and were then used for all the 16 settings. Every test set consisted of 1,000 points, selected using Latin hypercube sampling (e.g., see Owen 1994). Explicitly, for each input variable, 1,000 random numbers were selected, one in each interval $[(n - 1)/1{,}000, n/1{,}000]$, where n is an integer between 1 and 1,000. These numbers were then shuffled at random and used as the values for the input variable at the 1,000 test points. This sampling procedure leads to an estimate of the average performance of the prediction intervals that is asymptotically better than one obtained by generating the test points using simple uniform random sampling.

We chose two types of functions to test, one linear and one nonlinear. The linear function used was

$$\mathbf{y} = a\mathbf{X}\mathbf{1} + \varepsilon, \tag{15}$$

where $\mathbf{1}$ denotes a column vector of ones of the appropriate length. We used $a = 1$ when there were 5 inputs (predictions) and $a = .5$ when there were 20 inputs to keep the variance of the outputs constant. With this function, we used 200 points for the low level and 1,000 points for the high level of the dataset size factor.

Table 1. The Experimental Design

Factor	*Low setting*	*High setting*
Function	Linear (see below)	Nonlinear (see below)
Number of predictors	5	20
Observations	200 or 400	1,000
Noise level	σ = .1	σ = .25

To choose the network size in practice, one would cross-validate on each dataset encountered. For 1,600 runs, this proved to be too computationally expensive because it involves training several different networks of varying sizes to convergence for each run. Instead, for each setting, we selected three datasets out of the 100 replicates for cross-validation. We then used the number of nodes selected by these datasets for all the replicates at that setting. Because all 100 replicates are generated by the same function and error structure, this procedure should not affect the general simulation conclusions. Moreover, this method was used by all three methods for their cross-validation strategies. For the linear function we found that a network with a hidden layer of six nodes was chosen as the optimal network size for nearly all the settings when the function was linear. For simplicity, we kept the network size constant at 6 for all the linear runs.

The nonlinear function took the form

$$\mathbf{y} = 1.5\cos\left(\frac{2\pi}{\sqrt{3}}\sqrt{(\mathbf{X}_1 - .5)^2 + (\mathbf{X}_2 - .5)^2 + (\mathbf{X}_3 - .5)^2}\right) + \varepsilon. \tag{16}$$

Only the first three predictors are used to generate the function, and it is radially symmetric in these three variables. The factor $2\pi/\sqrt{3}$ was chosen so that only a single bump of the cosine function would be modeled, and the factor 1.5 was chosen to make the standard deviation of $f(\mathbf{X})$ approximately the same as that of the linear function. Using a similar limited cross-validation strategy as for the linear function, we found that 12 nodes provided consistently good prediction ability on the test sets when using the nonlinear function. We had intended to use 200 observations for the low level of dataset size, but when there were 20 input variables, the network could not come up with any reasonable fit to the function, perhaps because the network had 265 weights to estimate—significantly more than the number of observations available. Therefore, we used 400 for the lower value of the number of points for these runs.

The networks were trained on the 200, 400, or 1,000 training points, enabling predictions and 95% prediction intervals to be obtained for the test points. At each of the test points, we then computed the probability that an observation of the dependent variable would lie within the prediction interval, using a Gaussian cdf with mean equal to the predicted value and standard deviation equal to the estimated residual standard error. These probabilities were then averaged over all 1,000 points and over the 100 repetitions to determine whether 95% of the points fall in the intervals.

Because the intervals are prediction intervals rather than confidence intervals for the parameters, one cannot rely on the central limit theorem to ensure normality. Inherent in the derivation of prediction intervals shown in Section 2 is the assumption that the errors are Gaussian. Intervals obtained by resampling methods would avoid this assumption but with a neural network would be computationally prohibitive in most applications (see Baxt and White 1995).

The matrix $\mathbf{J}'\mathbf{J}$, although shown to be nonsingular in theory by Hwang and Ding (1997), is often very nearly singular when the number of parameters is large. This is especially true in the stopping-early method because there the model is deliberately overparameterized to ensure that the network has enough flexibility to fit the function when trained to convergence. This leads to instability, which can make individual prediction intervals several orders of magnitude too wide in extreme cases. It was necessary to use a pseudo-inverse to get even sensible results in many cases. In some extreme cases, one might even want to fit a network with more parameters than data points, in which case $\mathbf{J}'\mathbf{J}$ will be singular, so no prediction intervals can be generated without using a pseudo-inverse.

3.2 Results

Figure 1 shows the results from the experiment, training the network to convergence and then calculating prediction intervals using the method from nonlinear regression described in Section 2. In this and the following two figures, the dots show the percentage of points falling within the prediction intervals for each run of each level of the experimental design, averaging over the levels of the other factors. The x's, shown with linking lines, indicate the mean percentage for that factor level. For example, increasing σ^2 from .0100 to .0625 changes the percentage of points in the interval from around 94% to around 90%. Notice that the coverage is below 95% on the average, but it is closest to the nominal 95% coverage for 1,000 points. It is also closer to 95% for fewer input variables and smaller standard deviation.

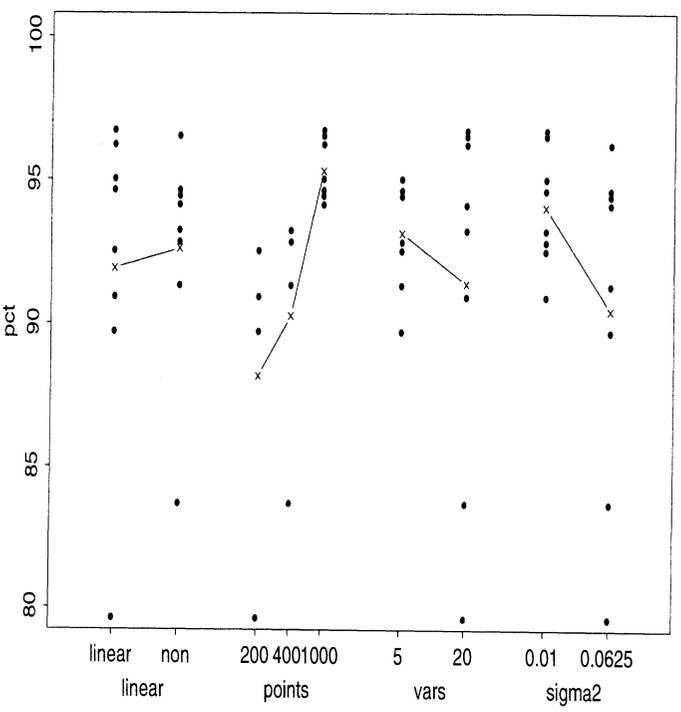

Figure 1. Main-Effects Plot for Training to Convergence Method. The x's, shown with linking lines, indicate the mean percentage coverage for each factor level.

The data from the experiment are also reproduced in Table 2. Here the column labeled "s^2" gives the mean estimate of σ^2, the column "Error" gives the median of the mean squared distances from the predicted values to the true function values at the test points, thus providing a measure of the accuracy of the fit, and the column "Bad" indicates the number of runs that were discarded because the optimization converged quickly to values nowhere near the minimum. The column "Pct." indicates the average percentage of points that fell within the prediction intervals.

We see that, when training is continued until convergence, the prediction intervals tend to be too narrow in the small samples. In the cases in which the coverage falls far short of 95%, s^2 underestimates σ^2. Although $\text{RSS}/(n-p)$ is an asymptotically unbiased estimate of σ^2, the local linear approximation on which this theory is based is not very accurate in small samples when there are many parameters and relatively few predictor variables. The model tends to overfit, causing $\text{RSS}/(n-p)$ to underestimate σ^2 and thus making the prediction intervals too narrow. When 1,000 points are used to train the network, the prediction intervals perform much better, as might be expected. In some cases, though, the prediction intervals are actually too wide. This can occur when the network becomes trapped in a local minimum, causing it to fit a smoother curve than it would at the the global minimum.

As an alternative to choosing the number of nodes in the hidden layer, one can use the early-stopping method, stopping the training algorithm short of convergence. To reiterate, in practice one would determine the number of iterations by cross-validation on each dataset, randomly dividing the data into training and test sets. Then, for each division, one would train the network using the training data, recording the RSS on the test set. The number of iterations giving the minimum average RSS on all the test sets would then be chosen. The network would then be retrained on all the data, stopping the convergence at this number of iterations. For each of the 16 settings, we used 3 of the 100 replicates as described for architecture choice and checked the error on the test sets every 10 iterations. The process was repeated 10 times on each of the three datasets. The number of iterations leading to the smallest mean squared error over the 30 runs was used to train the 100 replicates for that setting. The results are shown in Figure 2 and Table 3.

Table 2. Simulation Results Training to Convergence

Function	Pts.	Vars	σ^2	s^2	Error	Bad	Pct.
Linear	200	5	.0100	.0090	.0048	0	92.5
Linear	200	5	.0625	.0521	.0447	1	89.7
Linear	200	20	.0100	.0131	.0323	5	90.9
Linear	200	20	.0625	.0448	.3054	3	79.6
Linear	1,000	5	.0100	.0100	.0004	1	95.0
Linear	1,000	5	.0625	.0613	.0038	1	94.6
Linear	1,000	20	.0100	.0115	.0009	21	96.7
Linear	1,000	20	.0625	.0673	.0051	10	96.2
Nonlinear	400	5	.0100	.0103	.0083	4	92.8
Nonlinear	400	5	.0625	.0568	.0617	6	91.3
Nonlinear	400	20	.0100	.0135	.0447	1	93.2
Nonlinear	400	20	.0625	.0410	.2884	1	83.7
Nonlinear	1,000	5	.0100	.0113	.0053	9	94.6
Nonlinear	1,000	5	.0625	.0628	.0117	9	94.4
Nonlinear	1,000	20	.0100	.0173	.0096	13	96.5
Nonlinear	1,000	20	.0625	.0668	.0393	8	94.1

Early stopping generally produced prediction intervals that were too wide, although they were closer in larger samples. To understand this, we reexamine the estimated error variance, $s^2 = \text{RSS}/(n-p)$. Here p is the number of parameters that, for early stopping, is extremely large and, in fact, may be larger than n. By stopping the algorithm before convergence, the weights are shrunk toward 0, their approximate starting values, which reduces the effective number of parameters in the model. Thus, the number of effective parameters will be less than the number of parameters, and dividing RSS by $n-p$ tends to overestimate σ^2, resulting in overly conservative prediction intervals. One solution to this problem would be to estimate the effective number of parameters; unfortunately, no fully adequate method is available. For example, Moody (1992) provided an attractive approach, but it requires that σ^2 be known, which is unrealistic.

4. PREDICTION-INTERVAL THEORY USING WEIGHT DECAY

Another alternative to regularizing the network is to use a weight decay to prevent overfitting (see also Bishop 1995). That is, instead of minimizing RSS, we minimize $\text{RSS} + \alpha \sum_{i=1}^{p} w_i^2$. Although we will consider this method as an ad hoc method for regularizing the network, this approach can also be justified on Bayesian grounds. Suppose the weight vector w is given a prior distribution that is $\text{N}(0, \sigma^2/\alpha I)$. Then the posterior distribution is proportional to

$$p(w)p(y|w) \propto \exp -\alpha/2\sigma^2 \sum_{i=1}^{p} w_i^2 \exp -1/2\sigma^2$$

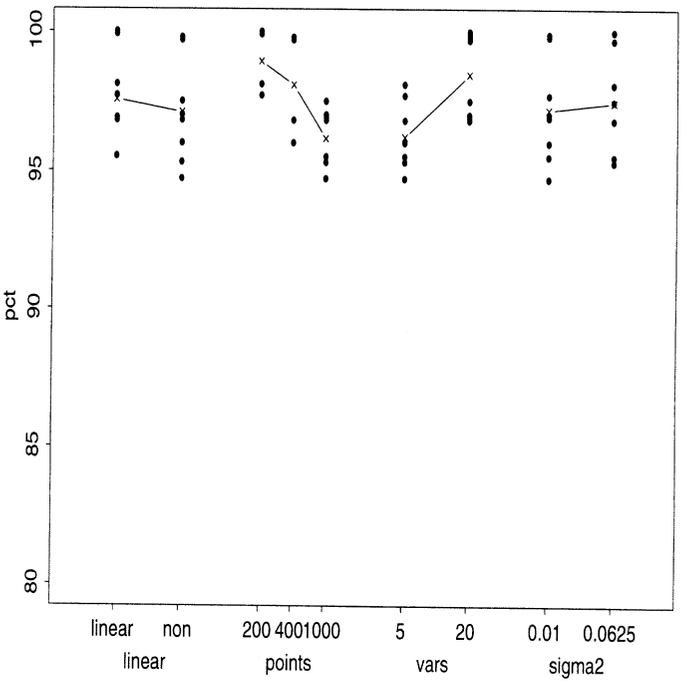

Figure 2. Main-Effects Plot for Stopping-Early Method.

Table 3. Simulation Results Stopping Training to Avoid Overfitting

Function	Pts.	Vars.	σ^2	Iters.	s^2	Error	Bad	Pct.
Linear	200	5	.0100	80	.0119	.0019	0	97.7
Linear	200	5	.0625	40	.0769	.0045	0	98.1
Linear	200	20	.0100	460	.0199	.0056	2	99.9
Linear	200	20	.0625	210	.1490	.0264	2	100.0
Linear	1,000	5	.0100	110	.0104	.0003	2	95.5
Linear	1,000	5	.0625	80	.0642	.0010	2	95.5
Linear	1,000	20	.0100	400	.0138	.0007	20	96.9
Linear	1,000	20	.0625	330	.0690	.0037	6	96.8
Nonlinear	400	5	.0100	330	.0138	.0067	3	96.0
Nonlinear	400	5	.0625	210	.0754	.0194	4	96.8
Nonlinear	400	20	.0100	1070	.0344	.0268	4	99.8
Nonlinear	400	20	.0625	790	.1278	.0883	1	99.7
Nonlinear	1,000	5	.0100	990	.0115	.0029	4	94.7
Nonlinear	1,000	5	.0625	350	.0671	.0092	11	95.3
Nonlinear	1,000	20	.0100	2540	.0183	.0091	10	97.0
Nonlinear	1,000	20	.0625	830	.0870	.0263	10	97.5

$$\times \sum_{i=1}^{n} (f(x_i, w) - y_i)^2$$

$$= \exp -1/2\sigma^2 \left(\mathrm{RSS} + \alpha \sum_{i=1}^{p} w_i^2 \right). \quad (17)$$

Therefore, the value for $\mathbf{w}$ that maximizes the posterior probability is the $\mathbf{w}$ obtained by minimizing RSS $+$ $\alpha \sum_{i=1}^{p} w_i^2$.

In the nonlinear case, if we use the approximation

$$\hat{y} \approx f(\mathbf{X}, \mathbf{w}^*) + \mathbf{J}(\hat{\mathbf{w}} - \mathbf{w}^*), \quad (18)$$

we choose $\hat{\mathbf{w}}$ to minimize RSS $+\alpha \sum_{i=1}^{p} w_i^2 \approx (\mathbf{k} - \mathbf{J}\hat{\mathbf{w}})'(\mathbf{k} - \mathbf{J}\hat{\mathbf{w}}) + \alpha \sum_{i=1}^{p} w_i^2$, where $\mathbf{k}$ is defined as in (7). Thus, the estimated parameters are

$$\hat{\mathbf{w}} \approx (\mathbf{J}'\mathbf{J} + \alpha\mathbf{I})^{-1}\mathbf{J}'\mathbf{k}. \quad (19)$$

It follows that

$$\mathrm{var}(\hat{y}|\mathbf{x}_0)$$

$$\approx \sigma^2(1 + \mathbf{g}_0'(\mathbf{J}'\mathbf{J} + \alpha\mathbf{I})^{-1}\mathbf{J}'\mathbf{J}(\mathbf{J}'\mathbf{J} + \alpha\mathbf{I})^{-1}\mathbf{g}_0). \quad (20)$$

Defining

$$\mathbf{H} = \mathbf{J}(\mathbf{J}'\mathbf{J} + \alpha\mathbf{I})^{-1}\mathbf{J}', \quad (21)$$

we compute

$$\begin{aligned} E[\mathrm{RSS}] &= E[(\mathbf{y} - \hat{\mathbf{y}})'(\mathbf{y} - \hat{\mathbf{y}})] \\ &\approx E[(\mathbf{k} - \mathbf{J}\hat{\mathbf{w}})'(\mathbf{k} - \mathbf{J}\hat{\mathbf{w}})] \\ &\approx E[\mathbf{k}'(\mathbf{I} - \mathbf{H})'(\mathbf{I} - \mathbf{H})\mathbf{k}] \\ &= \sigma^2 \mathrm{tr}((\mathbf{I} - \mathbf{H})'(\mathbf{I} - \mathbf{H})) \\ &\quad + E[\mathbf{k}]'(\mathbf{I} - \mathbf{H})'(\mathbf{I} - \mathbf{H})E[\mathbf{k}] \\ &= \sigma^2 \mathrm{tr}((\mathbf{I} - \mathbf{H})'(\mathbf{I} - \mathbf{H})) \\ &\quad + E[\mathbf{k}(\mathbf{I} - \mathbf{H})]'E[\mathbf{k}(\mathbf{I} - \mathbf{H})] \\ &\approx \sigma^2 \mathrm{tr}((\mathbf{I} - \mathbf{H})'(\mathbf{I} - \mathbf{H})) \\ &\quad + E[(\mathbf{y} - \hat{\mathbf{y}})]'E[(\mathbf{y} - \hat{\mathbf{y}})]. \quad (22) \end{aligned}$$

The second term is the sum of the squared biases of the predictions at the data points (Buja, Hastie, and Tibshirani 1989).

If we assume that the bias is negligible, we can simply define

$$s^2 = \mathrm{RSS}/(n - \mathrm{tr}(2\mathbf{H} - \mathbf{H}^2)) \quad (23)$$

and make the half width of a 95% prediction interval at $\mathbf{x}_0$

$$t_{n-p^*} s \sqrt{1 + \mathbf{g}_0'(\mathbf{J}'\mathbf{J} + k\mathbf{I})^{-1}(\mathbf{J}'\mathbf{J})(\mathbf{J}'\mathbf{J} + k\mathbf{I})^{-1}\mathbf{g}_0}, \quad (24)$$

where $p^* = \mathrm{tr}(2\mathbf{H} - \mathbf{H}^2)$. This method should lead to good prediction intervals when the bias is small. If the bias is substantial, it may still lead to reasonable prediction intervals due to compensating errors. To see this, note that in Equation (23) the computed value for s^2 overestimates σ^2 because the bias is not added into the denominator. In Equation (24), however, if we used the true σ instead of s, the intervals would be too narrow because the bias term would not be included under the square-root sign. Neglecting the bias under the square-root sign introduces an error that works in the opposite direction from the error introduced from neglecting the bias in the estimation of σ^2. In this case, the two errors are in opposite directions and may compensate for each other.

5. SIMULATION RESULTS USING WEIGHT DECAY

For each of the 16 settings in the design, we first determined approximately the best weight-decay constant using cross-validation. Following the procedure used for early stopping, the same three training sets used there were used to choose the optimum weight-decay constant. That is, the weight-decay constant α leading to the lowest average error on the 30 test sets was used for all 100 replicates for that setting. In practice, one would estimate α for each dataset individually. The results are shown in Figure 3 and Table 4. Notice that in all cases the coverage is much closer to the nominal 95% level than the previous two methods.

Here we see several possible advantages to using weight-decay-based prediction intervals. First, the prediction intervals performed much more consistently with this method

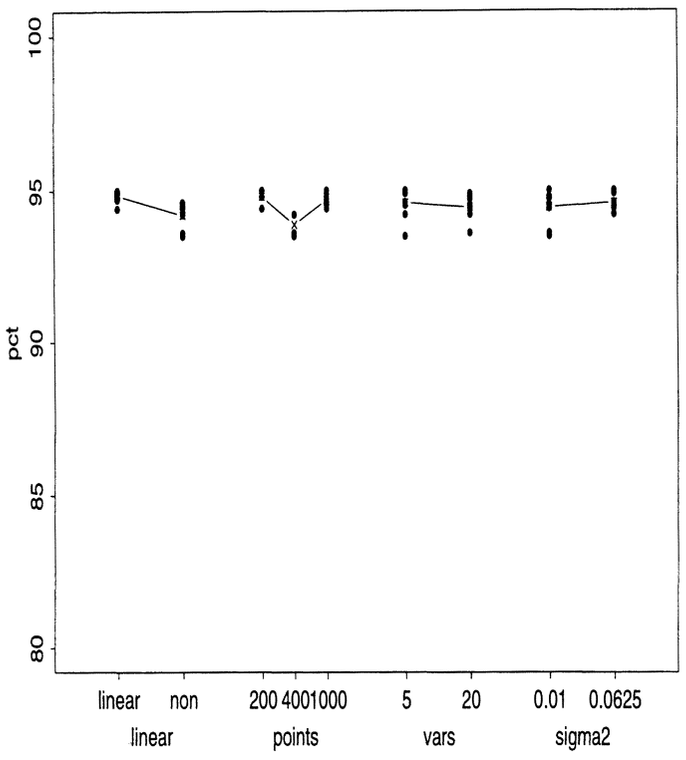

Figure 3. Main-Effects Plot for Weight-Decay Method.

than with the other methods, getting between 93.5% and 95.0% of the points in all 16 settings of the experimental design. The intervals were especially good on the linear function, although they tended to be slightly too small on the nonlinear function, especially when only 400 data points were used. Moreover, the error in fit tended to be smaller than when the network was regularized by early stopping, and there were fewer bad runs that had to be discarded. The method still tended to overestimate σ^2, mostly on the nonlinear function, because of model bias.

6. APPLICATION TO A REAL DATASET

We now apply the procedure to a set of 61 observations from a polymer-test facility (De Veaux, Psichogios, and Ungar 1993). The data were obtained from a pilot plant operation to evaluate the behavior of the process before scale-up was initiated. The dataset has 10 predictor variables $(x_1, \ldots, x_{10})$ and a single response variable y. It appears that the design in the predictor variables may be based on a sequential search in an effort to optimize the response, although, because the data are proprietary, no other information is available on the variables. The process is nonlinear, with some predictors important in predicting the response while others are insignificant. One of the predictors is a linear combination of several others, which makes the straightforward application of linear regression problematic. Due to the design, there are several highly influential points as well. Because neural networks are generally unable to shed light on the relationship between predictors and response, we will restrict our focus to only the prediction of the response and its uncertainty. We compare the predictions and the associated prediction intervals obtained from the weight-decay procedure with those from a nonparametric regression model. The data are available via ftp at ftp.cis.upenn.edu: pub/ungar/chemdata.

To fit the neural network to this small dataset and to avoid overfitting, tenfold cross-validation was performed on hidden layer sizes of 6, 12, and 18 nodes. The weight-decay constant was allowed to vary in the range .002 to .050, in increments of .002. In each run, 50 points were used to train and the remaining 11 were used to compute the prediction sum of squares. The best results on the prediction sum of squares were obtained using 18 nodes and $\mathbf{k} = .008$. The network was then trained using all 61 points with this value for the weight-decay constant. The model fit well, with a squared correlation between predictions, and observed at .906. Note that with 18 nodes, there are potentially many more parameters than observations. Keeping the weight decay constant guarantees that most of the parameters will be near 0. Two standard-error prediction intervals are shown in Figure 4. Because there are multiple inputs, we have ordered the responses in this plot.

As an alternative, the data were fit using multivariate adaptive regression splines (MARS), which selected a two-way interaction of x_4 and x_{10} and single terms in x_1, x_7, and x_8. We used this variable-selection information to construct

Table 4. Simulation Results Using Weight Decay

Function	*Pts.*	*Vars.*	σ^2	**k**	s^2	*Error*	*Bad*	*Pct.*
Linear	200	5	.0100	.03	.0102	.0007	0	95.0
Linear	200	5	.0625	.26	.0637	.0029	0	95.0
Linear	200	20	.0100	.18	.0118	.0033	0	94.8
Linear	200	20	.0625	.36	.0623	.0119	0	94.4
Linear	1,000	5	.0100	.06	.0100	.0002	0	95.0
Linear	1,000	5	.0625	.26	.0625	.0008	0	94.9
Linear	1,000	20	.0100	.12	.0102	.0011	0	94.7
Linear	1,000	20	.0625	1.04	.0640	.0034	0	94.9
Nonlinear	400	5	.0100	.01	.0114	.0053	0	93.5
Nonlinear	400	5	.0625	.08	.0669	.0145	0	94.2
Nonlinear	400	20	.0100	.08	.0191	.0221	0	93.6
Nonlinear	400	20	.0625	.26	.0793	.0504	1	94.2
Nonlinear	1,000	5	.0100	.01	.0110	.0023	0	94.5
Nonlinear	1,000	5	.0625	.04	.0641	.0075	1	94.6
Nonlinear	1,000	20	.0100	.06	.0150	.0094	0	94.4
Nonlinear	1,000	20	.0625	.30	.0722	.0235	0	94.5

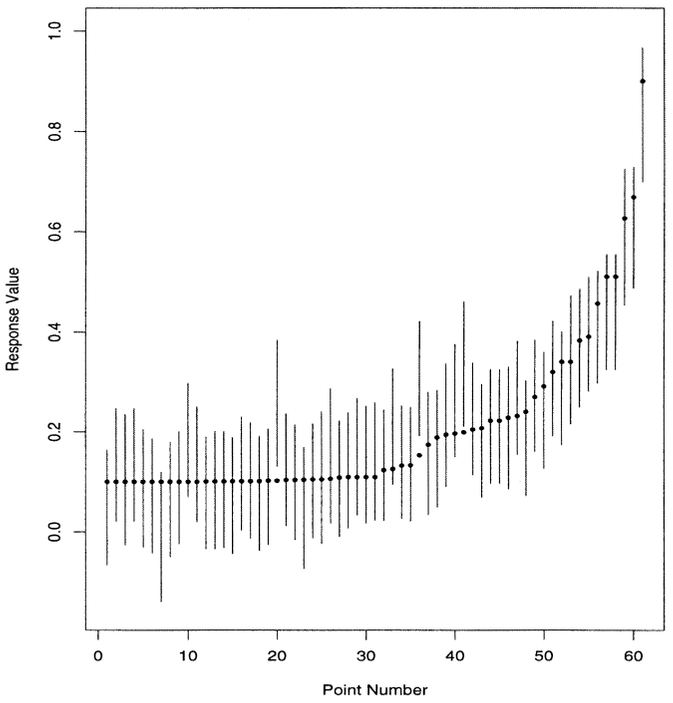

Figure 4. Prediction Intervals From a Neural Network With Weight Decay. The response values on the x axis have been arbitrarily ordered from smallest to largest.

a generalized additive model using loess fits in x_1, x_7, x_8, and the x_4, x_{10} surface. Specifically, we used the GAM model implementation in S-PLUS. The fit was similar to the neural-network fit, with a squared correlation between predictions and observed at .929. Of course, unlike neural networks, a regression model provides information on the relationship between the predictors and the response. A plot of each term from the GAM model is shown in Figure 5. The two standard-error prediction intervals for the GAM model are shown in Figure 6.

The prediction intervals are shown for both models superimposed in Figure 7. Here, rather than show the intervals as line segments, we have connected the endpoints of the intervals to highlight the differences between the two models. The MARS/GAM fit appears to be a slightly smoother fit than the neural-network model. Of course, the amount of smoothing performed by MARS/GAM can be chosen by cross-validation as well. We have simply used the default smoothing parameters here. The average size of the prediction intervals for the two models is very close, .235 for the neural network and .253 for the GAM model. We reemphasize that, of course, the neural-network model provides only predictions. But, this exercise does indicate that properly cross-validated neural networks can give prediction intervals of roughly the same size as a sophisticated nonparametric statistical model without the work of specifying the form of the model.

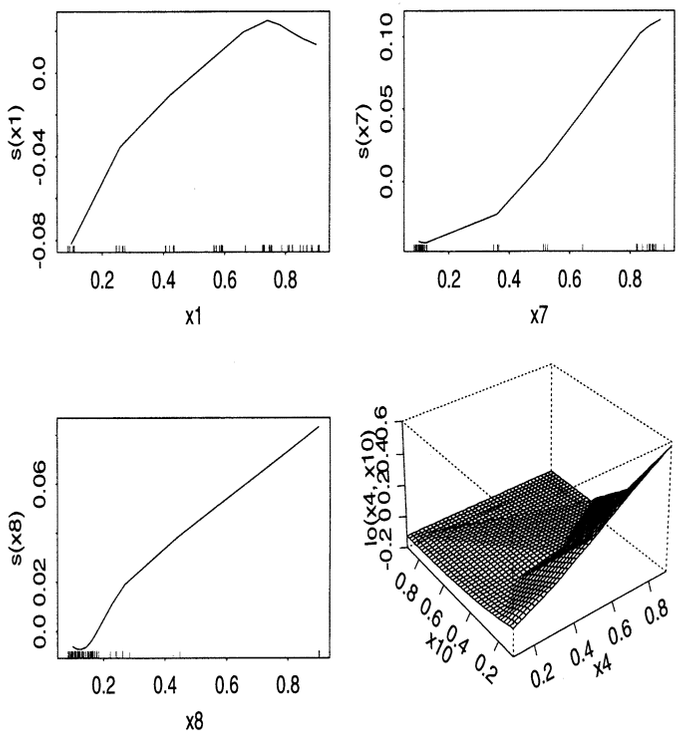

Figure 5. Plots of Curves and Surfaces From a MARS/GAM Fit. MARS was used to select the predictors and the interaction order, which were then used as the basis of the GAM model. For the four selected predictors, a plot showing how the response depends on each predictor is shown.

7. SUMMARY

Although the asymptotic theory for computing prediction intervals in nonlinear regression can be applied to neural networks, we have shown that several problems are encountered when applying this method to small samples. Note that "small" here is relative to the size of the network.

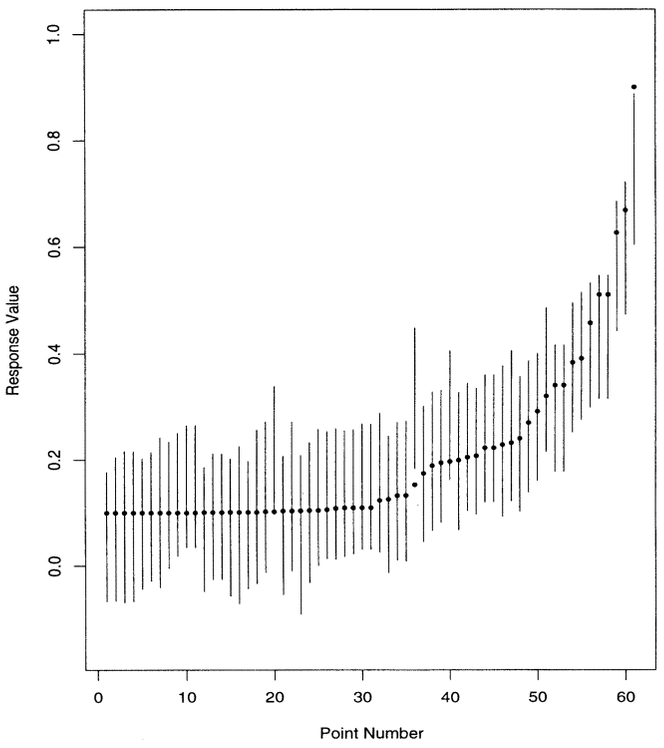

Figure 6. Prediction Intervals From the GAM Fit. Response values are ordered in the same way as in Figure 4.

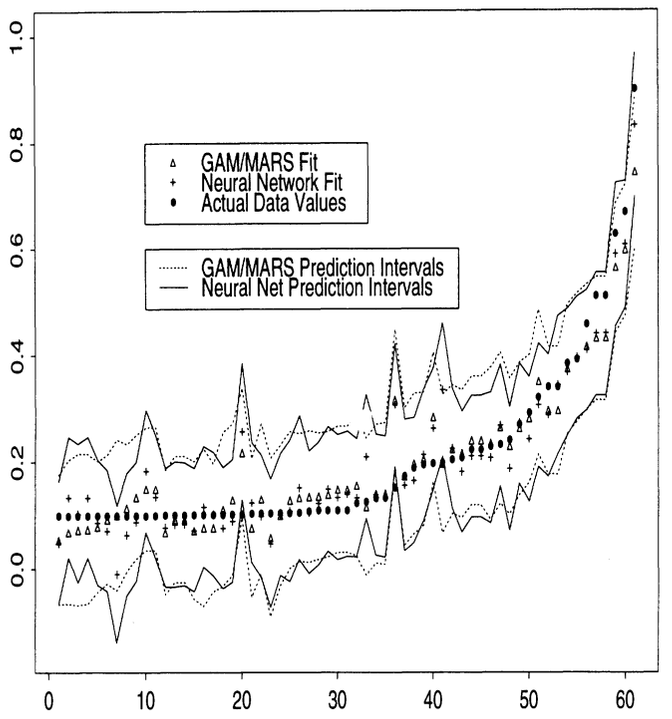

Figure 7. Prediction Intervals From Both Methods Superimposed for Comparison Purposes.

For a network with 256 weights, 400 observations is still a small sample. It is not uncommon for neural networks with tens of thousands of weights to be used. In such cases tens of thousands of observations can still be a "small" sample for the asymptotic theory. Because the $\mathbf{J}'\mathbf{J}$ matrix can be nearly singular, the estimates of the prediction variance at some points can be unstable, leading to unreasonable estimates. Moreover, there is a tendency either for the curve to fit closer to the points than predicted by the asymptotic theory because the linear approximation is not valid or for the curve to be smoother than predicted because of convergence to local minima. Furthermore, if training is stopped short of convergence, the prediction intervals will be too wide because the number of parameters overestimates the number of effective parameters. In large samples, however, these prediction intervals appear to work adequately.

We have used weight decay as a regularization method for neural networks and have derived the associated prediction intervals, based on standard nonlinear regression theory. Through a limited Monte Carlo simulation study, we have shown that this method may have the potential to ameliorate some of the deficiencies of other methods. In our study, the 95% prediction intervals consistently achieved their nominal coverage, although much more research is needed to discover under what circumstances this will hold in practice.

ACKNOWLEDGMENTS

We gratefully acknowledge the comments of two referees, the associate editor, and the editor who helped to improve the presentation of this article. We would also like to acknowledge the work of David Shellington, who helped with the simulations during a Summer Research Workshop at Williams College. Support for this work was partially funded by National Science Foundation grants DMS-9504451 (De Veaux, Schweinsberg, and Shellington) and CTS95-04407 (Ungar) and by grants from ICI Ltd. (De Veaux and Ungar) and FirstUSA Bank (De Veaux and Schumi).

[Received March 1997. Revised May 1998.]

REFERENCES

Baxt, W. G., and White, H. (1995), "Bootstrapping Confidence Intervals for Clinical Input Variable Effects in a Network Trained to Identify the Presence of Acute Myocardial Infarction," *Neural Computation*, 7, 624–638.

Bishop, C. M. (1995), *Neural Networks for Pattern Recognition*, Oxford, U.K.: Clarendon Press.

Buja, A., Hastie, T. J., and Tibshirani, R. J. (1989), "Linear Smoothers and Additive Models," *The Annals of Statistics*, 17, 453–555.

De Veaux, R. D., Psichogios, D. C., and Ungar, L. H. (1993), "A Comparison of Two Nonparametric Estimation Schemes: MARS and Neural Networks," *Computers in Chemical Engineering*, 17, 819–837.

Donaldson, J. R., and Schnabel, R. B. (1987), "Computation Experience With Confidence Regions and Confidence Intervals for Nonlinear Least Squares," *Technometrics*, 29, 67–82.

Hassibi, B., Stork, D. G., Wolff, G., and Watanabe, T. (1994), "Optimal Brain Surgeon: Extensions and Performance Comparisons," in *Advances in Neural Information Processing Systems 6*, San Mateo, CA: Morgan Kaufmann, pp. 263–270.

Hornik, K., Stinchcombe, M., and White, H. (1989), "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks*, 2, 359–366.

Hwang, J. T. G., and Ding, A. A. (1997), "Prediction Intervals for Artificial Neural Networks," *Journal of the American Statistical Association*, 92, 748–757.

Moody, J. (1992), "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems," in *Advances in Neural Information Processing Systems 4*, San Mateo, CA: Morgan Kaufmann, pp. 847–854.

Owen, A. B. (1994), "Controlling Correlations in Latin Hypercube Samples," *Journal of the American Statistical Association*, 89, 1517–1522.

Ripley, B. D. (1994), "Statistical Ideas for Selecting Network Architectures," in *Neural Networks: Artificial Intelligence and Industrial Applications*, eds. B. Kappen and S. Gielen, New York: Springer-Verlag, pp. 183–190.

——— (1996), *Pattern Recognition and Neural Networks*, Cambridge, U.K.: Cambridge Press.

Seber, G. A. F., and Wild, C. J. (1989), *Nonlinear Regression*, New York: Wiley.

Werbos, P. (1974), "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," unpublished Ph.D. dissertation, Harvard University, Dept. of Applied Mathematics.