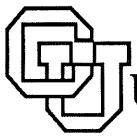
PREDICTIONS WITH CONFIDENCE INTERVALS (LOCAL ERROR BARS)

Andreas S. Weigend and David A. Nix

CU-CS-724-94



University of Colorado at Boulder

DEPARTMENT OF COMPUTER SCIENCE

Predictions with Confidence Intervals (Local Error Bars)

Andreas S. Weigend
Department of Computer Science
and Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430
andreas@cs.colorado.edu*

David A. Nix
Department of Computer Science
and Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430
dnix@cs.colorado.edu

***Abstract*—We present a new method for obtaining local error bars, i.e., estimates of the confidence in the predicted value that depend on the input. We approach this problem of nonlinear regression in a maximum likelihood framework. We demonstrate our technique first on computer generated data with locally varying, normally distributed target noise. We then apply it to the laser data from the Santa Fe Time Series Competition. Finally, we extend the technique to estimate error bars for iterated predictions, and apply it to the exact competition task where it gives the best performance to date.**

1 Obtaining Error Bars Using a Maximum Likelihood Framework

1.1 Motivation and Concept

Feed-forward artificial neural networks are widely used and well-suited for nonlinear regression. They can be interpreted as predicting the expected value of the conditional target distribution as a function of (or "conditioned on") the input pattern (e.g., Buntine & Weigend, 1991). This target distribution in response to each input may also be viewed as an error model (Rumelhart *et al.*, 1994). Often an estimate of the mean of this conditional target distribution suffices; this is typically done using one output unit. However, we here present a method that gives more information than just the mean of that distribution.

Such additional information could be obtained by attempting to estimate the entire conditional target distribution with connectionist methods (e.g., "fractional binning," Srivastava & Weigend, 1994) or with non-connectionist methods such as a Monte Carlo on a Hidden Markov Model (Fraser & Dimitriadis, 1994). Nonparametric estimates of the shape of a conditional target distribution require large quantities of data. In contrast, our less data-hungry method assumes a specific parameterized form of the conditional target distribution (e.g., Gaussian) and gives us the value of the error bar (e.g., the width of the Gaussian) by finding those parameters that maximize the likelihood of the data given the model.

Specifically, when we use a network output y to approximate a function $f(\mathbf{x})$, we assume that the desired output (d) (i.e., the observed values) can be modeled by $d(\mathbf{x}) = f(\mathbf{x}) + n(\mathbf{x})$ where $n(\mathbf{x})$ is noise drawn from the assumed error model distribution. Just as the estimate of the mean of this distribution $y(\mathbf{x})$ is a function of the input, the variance σ^2 may also vary as a function of the location in input space $\mathbf{x}$. When the noise level varies over the input space (i.e., $\sigma^2(\mathbf{x})$ depends on $\mathbf{x}$; it is not a constant), not only do we want the network to learn an output function $y(\mathbf{x})$ that estimates the expectation value $\mu(\mathbf{x})$ of the conditional target distribution, but we also want to learn a function $v(\mathbf{x})$ that estimates the variance $\sigma^2(\mathbf{x})$ of that distribution.

Therefore, we simply add an auxiliary output unit, the v-unit, that computes $v(\mathbf{x})$, our estimate of $\sigma^2(\mathbf{x})$. Since $\sigma^2(\mathbf{x})$ must be positive, we choose an exponential activation function for $v(\mathbf{x})$ to naturally impose this bound: $v(\mathbf{x}) = \exp\left[\sum_k w_{vk} h_k(\mathbf{x}) + \beta\right]$, where β is the offset (or "bias"), and w_{vk} is the weight between hidden unit k and the v-unit. This architecture is sketched in Figure 1.

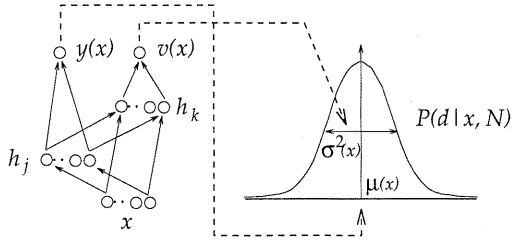


Figure 1: Architecture of the network with two output units. The y-unit predicts the conditional mean (i.e., given the input) of the output distribution. The v-unit predicts the conditional variance of that distribution. All weight layers have full connectivity. This architecture allows the v-unit to access both information in the input pattern itself and in the hidden unit representation formed while learning $y(\mathbf{x})$.

*http://www.cs.colorado.edu/homes/andreas/public_html/Home.html

Our network has two output units. For one of them, the y-unit, the target is easily available—it is simply given by d. But what is the target for the v-unit? We have to invent a target: this is effectively done by maximizing the likelihood of our network $\mathcal{N}$ given the data. Using Bayes' rule and assuming statistical independence of the errors, we equivalently minimize the negative log likelihood or cost $C = -\sum_i \ln P(d_i|\mathbf{x}_i, \mathcal{N})$. Traditionally, the resulting form of the cost C involves only the estimate $y(\mathbf{x}_i)$ of the mean of the assumed error model as a function of the input: the variance is assumed to be constant, and constant terms drop out after differentiation. In contrast, we here allow the variance to depend on the input, and explicitly keep these terms in C. Given any network architecture and any error model, the appropriate weight-update equations for gradient decent learning can be derived straightforwardly. In the next section, we illustrate this for the case where we assume that the deviations of the observed value from the mean are Gaussian distributed, and the variance of the Gaussian is given by $v(\mathbf{x})$.

1.2 Gaussian Error Model

Assuming normally distributed errors around $f(\mathbf{x})$ corresponds to a conditional target probability distribution of

$$P(d_i|\mathbf{x}_i, \mathcal{N}) = \frac{1}{\sqrt{2\pi\sigma^2(\mathbf{x}_i)}} \exp\left\{-\frac{[d_i - \mu(\mathbf{x}_i)]^2}{2\sigma^2(\mathbf{x}_i)}\right\} \quad , \tag{1}$$

with $\mu(\mathbf{x}) = f(\mathbf{x})$. Using the network output $y(\mathbf{x}_i) \approx \mu(\mathbf{x}_i)$ to estimate the mean, and $v(\mathbf{x}_i) \approx \sigma^2(\mathbf{x}_i)$ to estimate the variance, we obtain for the monotonically related negative log likelihood,

$$-\ln P(d_i|\mathbf{x}_i, \mathcal{N}) = \frac{2\pi v(\mathbf{x}_i)}{2} + \frac{[d_i - y(\mathbf{x}_i)]^2}{2v(\mathbf{x}_i)} \quad . \tag{2}$$

Dropping constant terms, summation over all patterns i yields the total cost:

$$C = \sum_i \frac{1}{2}\left(\frac{[d_i - y(\mathbf{x}_i)]^2}{v(\mathbf{x}_i)} + \ln[v(\mathbf{x}_i)]\right) \quad . \tag{3}$$

In order to write the explicit weight-update equations, we have to make assumptions about the transfer function of the units in the network. We here choose linear activation function for the y-unit, tanh activation functions for the hidden units, and an exponential activation function for the v-unit. We can then take derivatives of the cost C with respect to the network weights. To update weights connected to the y and v-units we have:

$$\Delta w_{yj} = \eta \frac{1}{v(\mathbf{x}_i)}[d_i - y(\mathbf{x}_i)]h_j(\mathbf{x}_i) \tag{4}$$

$$\Delta w_{vk} = \eta \frac{1}{2v(\mathbf{x}_i)}\left\{[d_i - y(\mathbf{x}_i)]^2 - v(\mathbf{x}_i)\right\} h_k(\mathbf{x}_i) \tag{5}$$

where η is the learning rate. For weights not connected to the output, the weight-update equations are derived by using the chain rule in the same way as in standard backpropagation. Note that Eq. (5) is equivalent to training a separate function-approximation network where the targets for $v(\mathbf{x})$ are the squared errors. Note also that if $v(\mathbf{x}_i)$ is constant, Eqs. (3)–(4) reduce to their familiar forms for standard backpropagation with a sum-squared error cost function.

A variation of $v(\mathbf{x}_i)$ with i implies a different weighting of each pattern. The $1/v(\mathbf{x})$ term in Eqs. (4)–(5) can be interpreted as a form of weighted regression, lowering the effective learning reate in high-noise regions. The effect is that the network tries hard to obtain small errors on those patterns where it can; it tries less hard on the patterns for which the expected error is going to be large anyway.

2 Mechanics

If the weighted regression term is allowed a significant influence early in gradient descent, local minima frequently result: the network consumes all its resources by trying hard to fit the first statistical feature it happens to encounter low errors on, discounting other patterns as being "high-variance." To avoid premature weighting of different patterns (that would be based on inaccurate values of $v(\mathbf{x}_i)$ before $f(\mathbf{x})$ is at least roughly approximated by $y(\mathbf{x})$), we separate training into three phases:

Phase I (Mean): Randomly split the available data into equal halves, sets $\mathcal{A}$ and $\mathcal{B}$. Learn the conditional expectation value $y(\mathbf{x})$ by using set $\mathcal{A}$ as training set. In Phase I we use simple gradient descent on a simple squared-error cost function, i.e., Eqs. (3)–(4) *without* the $1/v(\mathbf{x})$ terms.[1] To guard against overfitting, training is considered complete at the minimum of the squared error on the cross-validation set $\mathcal{B}$, monitored at the end of each complete pass through the data.

[1] Further details are: all inputs are scaled to zero mean and unit variance. All initial weights feeding into hidden units are drawn from a uniform distribution $\in [1/i, -1/i]$ where i is the number of incoming connections. All initial weights feeding into y or v are drawn from a uniform distribution $\in [s/i, -s/i]$ where s is the standard deviation of the (overall) target distribution. No momentum is used, and all weight updates are averaged over the forward passes of 20 patterns.

Phase II (Variance): Attach a layer of hidden units connected to both the inputs and the hidden units of the network from Phase I (see Figure 1). Freeze the weights trained in Phase I, and train the v-unit to predict the squared errors, again using simple gradient descent as in Phase I. The training set for this phase is set $\mathcal{B}$, with set $\mathcal{A}$ used for cross-validation— if set $\mathcal{A}$ were used as the training set in this phase as well, possible overfitting in Phase I could result in seriously underestimating $v(\mathbf{x})$. To avoid this risk, we interchange the data sets. The initial value for the offset β of the v-unit is the natural logarithm of the mean squared error (from Phase I) of set $\mathcal{B}$. Phase II stops when the squared error on set $\mathcal{A}$ levels out or starts to increase.

Phase III (Weighted regression): Re-split the available data into two new halves, $\mathcal{A}'$ and $\mathcal{B}'$. Unfreeze all weights and train all network parameters to minimize the full cost function C using Eqs. (4)–(5) and their chain-rule counterparts on set $\mathcal{A}'$. Training is considered complete when C has reached its minimum on set $\mathcal{B}'$. Note that the $1/v(\mathbf{x})$ terms in Eqs. (4)–(5) implement a form of weighted regression, attenuating the learning in regions where the estimated $v(\mathbf{x})$ is high. The three-phase approach greatly reduces the probability of local minima by giving the composite network a head start on learning the function $f(\mathbf{x})$.

3 Example #1: Computer-Generated Data

To demonstrate the application this method, we construct a one-dimensional example problem where the true $f(\mathbf{x})$ and $\sigma^2(\mathbf{x})$ are known. We take the equation

$$f(x) = \sin(\omega_\alpha x)\sin(\omega_\beta x) = \frac{1}{2}[\cos(\omega_\alpha - \omega_\beta)x - \cos(\omega_\alpha + \omega_\beta)x] \ , \quad \text{with } \omega_\alpha = 2.5 \text{ and } \omega_\beta = 1.5 \qquad (6)$$

over the interval $x \in [0, \pi]$. We generate the target values by adding normally distributed noise $n(x) = N(0, \sigma^2(x))$, where $\sigma^2(x)$ varies according to

$$\sigma^2(x) = 0.01 + 0.25 \times [1 - \sin(\omega_\alpha x)]^2 \quad . \qquad (7)$$

We generate 1000 patterns for training, and an additional 10^5 patterns for evaluation after training.

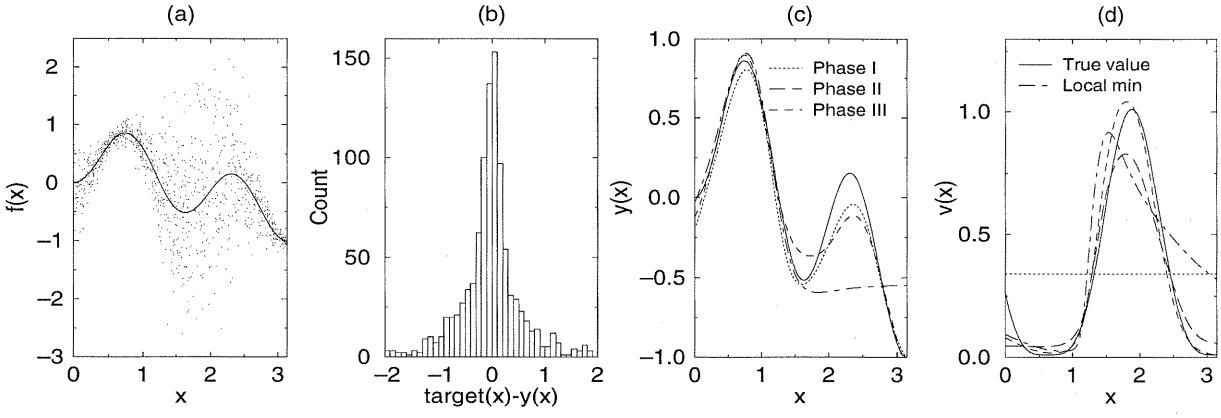


Figure 2: The computer-generated example:
(a) True function $f(x)$ and the 1000 training points;
(b) Histogram of residual errors (end of Phase III);
(c) True $f(x)$ (solid line) and network output $y(x)$ at end of Phase I (dotted) and Phase III; also shown is an example of a local minimum resulting from starting immediately with Phase III.
(d) True $\sigma^2(x)$ and network output $v(x)$ at end of Phase II and Phase III. (Phase I value is mean squared error.) Legend applies to both (c) and (d).

Training follows exactly the three phases described above with the following details: Phase I uses a network with one hidden layer of 10 tanh units and a conservatively small learning rate of $\eta = 10^{-2}$. For Phase II we add an auxiliary layer of 10 tanh hidden units connected to the v-unit (see Figure 1) and use the same η. In Phase III the composite network is trained by gradient descent on Eq. (3) with $\eta = 10^{-4}$, minimizing the full cost (weighted regression). In Figures 2b and 2c we also plot a local minimum resulting from attempting to minimize Eq. (3) directly: the model misspecification is interpreted as a high-noise region and effectively ignored in learning.

Note the kurtosis (tall center and heavy tails) of the distribution of the residuals (Figure 2b), typical for data drawn from at least two normal distributions with the same mean but different variances. Note in Figure 2c how the Phase I estimate is closer to the true function *overall*, but misses the function in the low-noise region (for $0 < x < 1$). In contrast, the Phase III weighted-regression estimate is much closer to $f(x)$ for $0 < x < 1$, but is smoother for $1.5 < x < 2.5$ where the uncertainty is large anyway. Figure 2d shows that $v(x)$ after the final Phase III accurately estimates $\sigma^2(x)$, with an appropriate slight overestimation in the region of model misspecification. Table 1 illustrates the importance of the fine tuning that Phase III performs in minimizing the negative log likelihood. The entries in the table show

that the network is able to predict the expected error for this noisy system very well: the percentage of residual values less than one and two predicted standard deviations is very close to the theoretical and optimally achievable values for this data set.

row		**Train**	$(N = 10^3)$	**Test**	$(N = 10^5)$
		NMSE	cost	NMSE	cost
1	Phase I	0.599	−0.039	0.582	−0.051
2	Phase II	0.599	−0.395	0.582	−0.398
3	Phase III	0.604	−0.498	0.588	−0.513
4	$n(x)$ *(exact additive noise)*	*0.588*	*−0.542*	*0.562*	*−0.581*
		ρ		ρ	
5	$\rho(v^{1/2}(x)$, residual errors)	0.579		0.597	
6	$\rho(\sigma(x)$, *residual errors)*	*0.588*		*0.605*	
		1 std	2 std	1 std	2 std
7	% of errors $< v^{1/2}(x); 2v^{1/2}(x)$	70.2	95.3	69.63	95.16
8	*% of errors* $< \sigma(x); 2\sigma(x)$	*67.7*	*95.2*	*66.76*	*94.67*
9	*(exact Gaussian)*	*68.26*	*95.44*	*68.26*	*95.44*

Table 1: Results of the computer-generated example. The first 3 rows give the normalized mean squared error and the cost (more negative values are better). (NMSE denotes the mean squared error divided by the overall variance of the target). While the error for Phase I and Phase II is identical (frozen weights to y-unit), the cost decreases after Phase II. Row 4 compares to the values of the exact model. Row 5 gives the correlation coefficient between the network's predictions for the standard error (i.e., the square root of the activation of the v-unit) and the actually occurring L1-errors, $|d(x) - y(x)|$. Row 6 gives the value obtained by comparing the value σ (i.e., the ideal model) with the residual errors. Rows 7 and 8 give the percentage of residuals smaller than one and two standard deviations.

4 Example #2: Laser Data Set From the Santa Fe Competition

We now apply our method to a set of observed data, the 1000-point laser intensity series from the Santa Fe competition.[2] Viewing time series prediction as function approximation or nonlinear regression, it would make it easier for the predictor if we had more points that lie on the manifold. (Figure 3 shows the manifold the network is trying to approximate.) In the competition, Sauer (1994) upsampled the 1000 available data points with an FFT method by a factor of 32. This does not change the effective sampling rate, but it "fills in" more points. We use the same upsampling trick (without filtered embedding), and randomly split the resulting 31200 data points into two equal-sized sets, $\mathcal{A}$ and $\mathcal{B}$.

Phase I: We train a simple network with 25 inputs (corresponding to 25 past values), 12 tanh hidden units, and 1 linear output for single-step prediction.[3] (Learning rate $\eta = 10^{-7}$.) At the end of each epoch, we monitor not only the single-step errors for the cross-validation set $\mathcal{B}$, but also something else: in view of the competition task of iteratively predicting 100 values (with error bars), we pick 500 starting positions at random from the original series (from points 26-900). We then compute the 100-point iterated predictions for each of these starting position. Phase I stops when the average error of the 100 predictions following the 500 starting points has reached a minimum. Interestingly, this minimum occurs well before the minimum of single-step prediction error on set $\mathcal{B}$.

Phase II: We freeze the weights from Phase I, and add the 12 tanh hidden units as shown in Figure 1. We train the v-unit on the squared errors from set $\mathcal{B}$ with $\eta = 10^{-9}$. The final weights for Phase II are taken at the location of the error minimum on set $\mathcal{A}$.

Phase III: All weights are unfrozen, and the composite network is trained with $\eta = 10^{-9}$. We monitor both the cost C and the average error over the 500 iterated sequences. The final network is the one at the minimum of the iterated prediction measure.

The competition task for this data set was to submit a 100-point continuation with error bars. None of the 14 submissions estimated the error bars in a principled way. Our method, as discussed so far, does not yet provide uncertainty estimates for iterated predictions either. We now show how our single-step uncertainty estimates can be used for iterated predictions: In addition to the point prediction of the next time step, we make predictions at ± one standard deviation of the most recent value of the input vector. If the distance between either of these two predictions is greater than the single-step uncertainty estimate, this distance is used as the current uncertainty estimate. Because the competition log likelihood measure is highly sensitive to even a single error bar that is too small (Gershenfeld & Weigend, 1994, pp. 64-65), we impose a lower bound of 4.0 on the variance in calculating the iterated uncertainties. Table 2 summarizes the results. We would like to point out that our method of predicting errors could have been followed by any competition participant— the only data we use for model building are precisely the 1000 points that were available during the competition.

[2] The data set and several predictions and characterizations are described in the volume edited by Weigend & Gershenfeld (1994). The data is available by anonymous ftp at `ftp.cs.colorado.edu` in `/pub/Time-Series/SantaFe` as `A.dat`.

[3] These are the same dimensions as Wan (1994) used; we did not try any other architectures.

	Train		Test		
	900-pt single st.	900–1000 iterated	1–100 single st.	1–100 iterated	
	NMSE	NMSE	NMSE	NMSE	NALL
Naive Baseline	—	—	—	**1.001**	**8.50**
Sauer (1994)	—	—	—	**0.080**	**4.84**
Wan (1994)					
Network	0.0004	0.0026	0.0230	**0.055**	—
Competition Entry	—	—	—	**0.028**	**3.48**
This Paper					
Network	0.0010	0.0142	0.0198	**0.096**	—
Competition Conditions	—	—	—	**0.016**	**3.28**

Table 2: Results for the laser data, both for single step ("single st.") and iterated predictions. The "naive baseline" corresponds to simply using the mean of the training data as predictions and the standard deviation of the training data as error bars for every point. Wan's (1994) competition entry consisted of 75 points generated by his network, followed by 25 points (after the predicted collapse) from a similar stretch that he picked from the training data by visual inspection. To facilitate the comparison, after our network indicates a collapse (based on a sudden jump in uncertainty), we also uses the mean of training points 620-700 as the prediction, and the standard deviation of this segment as the expected error after the collapse. Boldface denotes the measures used in the Santa Fe competition.

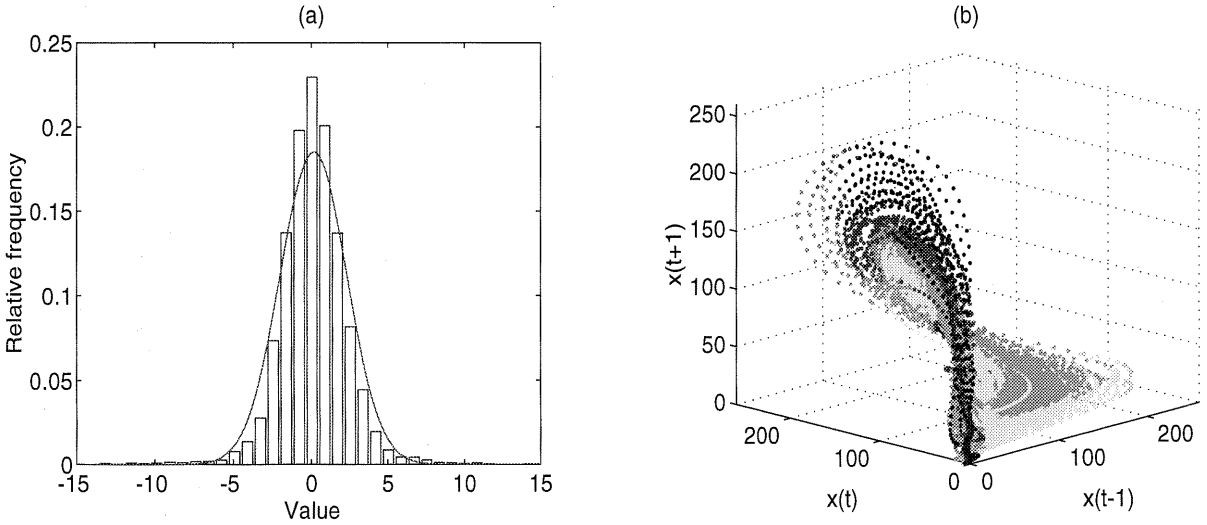


Figure 3: Results for the laser data: (a) Histogram of residual errors. The slight kurtosis suggests non-uniform variance, but less than in Figure 2b. (b) State-space embedding: the grey scale indicates the degree of certainty ($v(\mathbf{x})$ from lightest to darkest, < 3, 3-5, 5-10, > 10). Different regions have clearly different degrees of predictability. This shows that it is a good idea to get *local* estimates of uncertainty!

5 Discussion and Relation to Other Work

In summary, we have combined Sauer's upsampling trick with our method of estimating uncertainties, both for single-step and iterated predictions. The key feature is that we start with the maximum likelihood principle and arrive at local error bars that depend on the location in input space. Comparing our result for the laser benchmark data (shown in Figure 4) with the top two Santa Fe competition entries (in Table 2) demonstrates both the ease with which a simple network approximates the manifold given sufficient data, and the applicability of our method for predicting impredictability.

Our method encompasses most sources of uncertainty, such as uncertainty due to stochasticity (outside shocks, measurement noise), and uncertainty due to the divergence of nearby trajectories (particularly large after the collapses). It indirectly also handles model misspecification by appropriately overestimating the variance. It does not take into account the uncertainty of the predictor due to specific splits of the data into training, cross-validation, and test sets. Weigend & LeBaron (1994) use a bootstrapping method and show on an example (financial data) that this source of uncertainty is significantly more important than model and evaluation uncertainty due to other choices (e.g., initial weights). However, since we here wanted to followed the Santa Fe rules as closely as possible, we did not bootstrap the test set.

When dealing with finite sets of noisy data, overfitting the function is already a serious problem. If we also want to estimate local error bars, not only must we fear overfitting $y(\mathbf{x})$, but we must also be concerned with overfitting $v(\mathbf{x})$. If the standard method of starting with small weights and stopping at the minimum

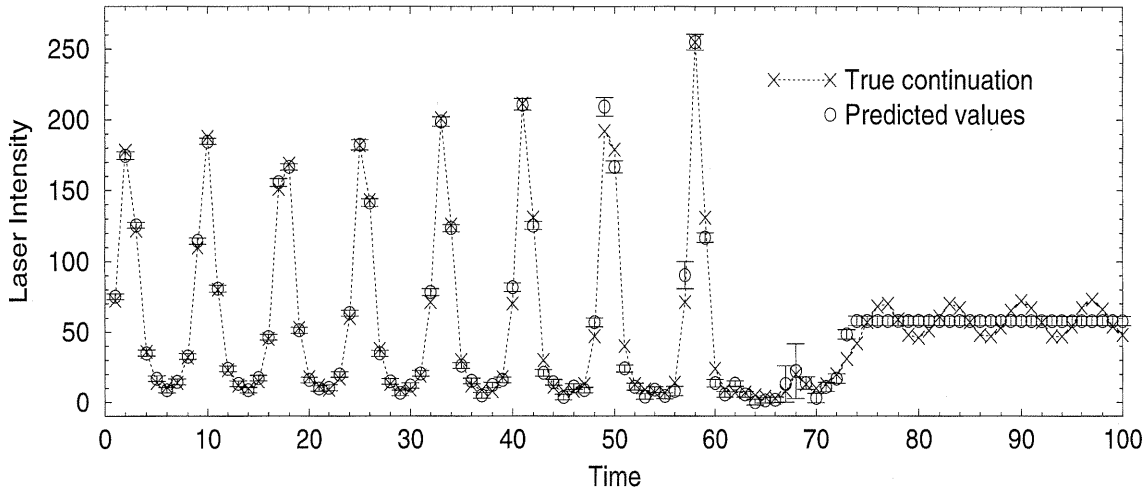


Figure 4: Iterated predictions for the laser data on the continuation set (never used in training). Note the large errors and large error bars in the region of great uncertainty around Time=66. In the region after the collapse, we follow Wan (1994) and use as prediction values the mean of the post-collapse training data (from training points 620-700), and as (global) error bars the standard deviation of the same training region.

of an appropriate cross-validation set does not suffice for a given problem, it is straightforward to employ the usual anti-overfitting weaponry (smooth v as a function of $\mathbf{x}$, pruning, weight-elimination, etc.).

In any example, we have to choose a specific error model. The framework presented here encompasses any distribution with a location parameter (conditional mean) and a scale parameter (local error bar). The framework also carries over from regression to classification where it allows to quantify the amount of uncertainty of a probability estimate (of a pattern belonging to a certain class) by giving the width of that distribution, again depending on the input pattern.

Acknowledgments

This work is supported by the National Science Foundation under Grant No. RIA ECS-9309786 and by a Graduate Fellowship from the Office of Naval Research. We would like to thank Dave Rumelhart, Wray Buntine, Steve Nowlan, and Barak Pearlmutter for discussions.

References

W.L. Buntine and A.S. Weigend. (1991) "Bayesian Backpropagation." *Complex Systems*, **5**: 603–643.

N.A. Gershenfeld and A.S. Weigend. (1994) "The Future of Time Series." In *Time Series Prediction: Forecasting the Future and Understanding the Past*, A.S. Weigend and N.A. Gershenfeld, eds., Addison-Wesley, pp. 1–70.

A.M. Fraser and A. Dimitriadis. (1994) "Forecasting Probability Densities Using Hidden Markov Models with Mixed States." In *Time Series Prediction: Forecasting the Future and Understanding the Past*, A.S. Weigend and N.A. Gershenfeld, eds., Addison-Wesley, pp. 265–282.

D.E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin. (1994) "Backpropagation: The Basic Theory." In *Backpropagation: Theory, Architectures and Applications,* Y. Chauvin and D.E. Rumelhart, eds., Lawrence Erlbaum.

T. Sauer. (1994) "Time Series Prediction by Using Delay Coordinate Embedding." In *Time Series Prediction: Forecasting the Future and Understanding the Past*, A.S. Weigend and N.A. Gershenfeld, eds., Addison-Wesley, pp. 175-193.

A.N. Srivastava and A.S. Weigend. (1994) "Computing the Probability Density in Connectionist Regression." In *Proceedings of the ICANN'94*, to be published.

R. Tibshirani. (1994) "A comparison of some error estimates for neural network models." University of Toronto preprint.

E.A. Wan. (1994) "Time Series Prediction by Using a Connectionist Network with Internal Delay Lines." In *Time Series Prediction: Forecasting the Future and Understanding the Past*, A.S. Weigend and N.A. Gershenfeld, eds., Addison-Wesley, pp. 195-217.

A.S. Weigend and B. LeBaron. (1994) "Evaluating Neural Network Predictors by Bootstrapping." Technical Report CU-CS-725-94 (May 1994), Computer Science Department, University of Colorado.

A.S. Weigend and N.A. Gershenfeld, eds., (1994) *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison-Wesley.