01 Jan 1995

Using Taguchi"s Method of Experimental Design to Control Errors in Layered Perceptrons

William E. Bond
Missouri University of Science and Technology, bondw@mst.edu

Gerald E. Peterson

Daniel C. St. Clair

Stephen R. Aylward

Using Taguchi's Method of Experimental Design to Control Errors in Layered Perceptrons

Gerald E. Peterson, Daniel C. St. Clair, Stephen R. Aylward, and William E. Bond

***Abstract*—A significant problem in the design and construction of an artificial neural network for function approximation is limiting the magnitude and the variance of errors when the network is used in the field. Network errors can occur when the training data does not faithfully represent the required function due to noise or low sampling rates, when the network's flexibility does not match the variability of the data, or when the input data to the resultant network is noisy. This paper reports on several experiments whose purpose was to rank the relative significance of these error sources and thereby find neural network design principles for limiting the magnitude and variance of network errors.**

I. Introduction

Taguchi's method of experimental design is widely used in industry for the purpose of finding those factors in a manufacturing process which are most important in achieving useful goals. Several factors which are related to the goals and are under the user's control are selected. These factors are varied over two or more levels in a systematic way and the results analyzed. The analysis reveals which of the factors are most effective in reaching the goals. Control over achieving the goals will be best obtained by changes in these primary factors. This paper applies these methods to the control of errors in neural networks.

For neural networks, one useful goal is to decrease errors in the final fielded network and another is to decrease the variability of those errors. Factors related to these goals that may be under the network builder's control include the noise in the training data, the density of the training data, the network architecture, the time at which training is stopped, and the noise in the testing data. The result of a Taguchi analysis can help provide answers to the following critical design and construction issues.

- What is the proper density for training samples in the input space?
- When is the best time to stop training to avoid overfitting?
- Which is the best architecture to use? Is it better to use a large architecture and stop training optimally or to use an optimum architecture, which probably will not overfit the data, but may require more time to train?
- If noise is present in the training data, is it best to reduce the amount of noise or gather additional data?
- What is the effect of noise in the testing data on the performance of the network?

This research was undertaken because answers to the above questions would be useful for the application of neural networks to engineering problems, but unavailable, except in bits and pieces, in the literature. We believe this is the first systematic experimental investigation into the causes of errors in neural networks.

Intuitively, the manner in which the above factors could help control network errors is as follows. Decreasing training noise decreases the opportunity for fitting errors in the training data. Increasing training data density increases the amount of flexibility necessary to approximate the noise. An optimum architecture would have exactly that degree of flexibility required to fit the function, but not the noise. Since overfitting occurs in the later stages of training, it is possible to exert some control over it by stopping training when overfitting begins.

From the results of the Taguchi experiments, several conclusions were drawn. They are discussed in detail in Section VI. These conclusions should be considered somewhat tentative because they are based on experiments with only two functions. We believe, however, these conclusions are correct and will withstand further scrutiny. The primary conclusions are the following:

1) For the values of training data noise and data density and for the two architectures which were used, reducing the size of training data noise had more effect on the size and variability of errors than changes in any of the other factors.
2) Training time is significantly reduced for network architectures having two hidden layers when compared with networks having one hidden layer if both networks have a similar number of weights. In general, networks with two hidden layers more closely approximate training samples than those with one hidden layer. Furthermore, if training can be stopped before overfitting starts, then architectures with two hidden layers limit the level and variability of errors better than those with one hidden layer.
3) If error is plotted versus the number of training iterations, then overfitting frequently starts at the bottom of a steep drop in the training error. The possibility of overfitting decreases with increasing density of the

Manuscript received March 15, 1993; revised July 23, 1993 and May 24, 1994.

D. St. Clair is with the University of Missouri-Rolla, St. Louis, MO 63146 USA.

G. Peterson and W. Bond are with McDonnell Douglas Corporation, St. Louis, MO 63146 USA.

S. Aylward is with the Department of Computer Science, University of North Carolina at Chapel Hill, on educational leave from McDonnell Douglas Corporation.

IEEE Log Number 9409400.

training data, increases with the number of hidden layers in the architecture, and increases with increasing training data noise.

4) Increasing training data density will decrease testing error.

Two other observations were made as the research progressed. The first is that a wide variety of training and testing behaviors were observed. For example, cases were found in which the testing error was below the training error for the entire training period. In other cases, no error reduction seemed possible and both error curves were flat. In most cases, even for very large networks, it was not possible to bring the training error to zero—it flattened out at some positive error value.

The second observation is that in some cases, a significant increase in the initial random values of the weights improved convergence dramatically. From a situation in which no convergence was occurring, an increase in the initial values of the weights produced a situation in which convergence was immediate and steep. This was unexpected since many authors [11], [12], [15] suggest starting with small initial weights.

It is intended that this work lay a foundation for building trust in the operation of neural networks, especially in critical applications where the well-being of property or people is at stake. Gaining better control over network errors can contribute to this goal.

In the following sections, the type of neural network used and the manner in which errors are measured are described first. Then an overall description of the experiments is presented. This includes a description of the data, the network architectures, and the experimental method. The two experiments are described in detail, including several graphs which illustrate the generalization behavior, and tables which rank the factors that were effective in reducing errors and their variances. Guidelines for error control which were inferred from the work are presented in Section VI. Finally, some conclusions about the approach and suggestions for further research are given in Section VII.

II. Neural Networks

In this section, notation is fixed and the error measures which were used in the experiments are defined.

A data sample is a set of cases, each case being a pair consisting of an input vector and the corresponding output vector. The sample is drawn from a parent population of all possible input–output pairs. For notation, use $\xi = \{(x_i, y_i)\}_{i=1}^{N}$ for a data sample. The components of x_i and y_i are denoted by $x_i = (x_{i1}, \cdots, x_{iI})$ and $y_i = (y_{i1}, \cdots, y_{iM})$ where I is the number of inputs and M is the number of outputs. The data sample used to train a neural network is called the training sample or the set of training cases.

A feedforward neural network with one hidden layer and one output is given by the expression

$$\mathrm{Out}(x_i) = h\left(w_0 + \sum_{k=1}^{H} w_k g\left(w_{k0} + \sum_{j=1}^{I} w_{kj} x_{ij}\right)\right) \quad (1)$$

the positive integer H is the number of hidden nodes, the activation functions g and h are given, and the weights w_k and w_{kj} are variables which are adjusted in a way such that $f(x_i)$ approximates y_i for every training case (x_i, y_i). A so-called sigmoidal function, such as $s(x) = \tanh\frac{1}{2}x$, is a common choice for each of the activation functions.

The apparent error of a network is the error produced by the training sample. The true error is the error on the parent population. It is tempting to believe that the true error will be close to the apparent error, but experience (see [17]) has shown that the true error is often greater than the apparent error and in common situations the true error can be very significantly greater than the apparent error.

The true error may be estimated (see [17]) by using a separate sample of cases for testing, by using resampling techniques such as cross validation or by algebraic estimates such as the final prediction error of Akaike [1]. Four-fold cross validation is used in the experiments described here. That is, the data sample $\xi = \{(x_i, y_i)\}_{i=1}^{K}$ is partitioned into four equal parts $\xi_1, \xi_2, \xi_3, \xi_4$ and the network is trained four times. During the ℓth training, ξ_ℓ is set aside for testing and $\xi - \xi_\ell$ is used for training. The final error is the average of the errors during the four trials. The error measure used in this report is the average root mean squared error, which is defined as

$$\mathrm{ARMSE} = \frac{1}{K}\sum_{i=1}^{K} |y_i - \mathrm{Out}(x_i)|.$$

Even if the average value of the true error of the network is of an acceptable size, the network may be unusable if the magnitude of the error varies too much. The measure of error variability used here is the variance of the ARMSE over the last half (see Section III-D)) of the training iterations. If T is the number of training iterations, then this variance is given by

$$\sigma^2 = \frac{1}{\frac{T}{2} - 1} \sum_{i=T/2+1}^{T} (\mathrm{ARMSE}_i - \overline{\mathrm{ARMSE}})^2$$

where $\overline{\mathrm{ARMSE}} = \frac{1}{T/2}\sum_{i=T/2+1}^{T} \mathrm{ARMSE}_i$. In most cases, T is a multiple of K.

III. Description of Experiments

The experiments were designed to investigate how to determine the proper density of training samples, when to stop training to reduce overfitting, the best architecture to use, and the effect of noise in the training and testing data. The objective of these experiments was to evaluate theory and practice to produce a set of guidelines which would be useful to practitioners.

Backpropagation neural networks are usually used in one of two situations: to estimate the values of a function or to perform classification tasks. The experiments performed in this research focused on the use of back propagation neural networks to estimate function values. There are a significant number of engineering applications related to this capability.

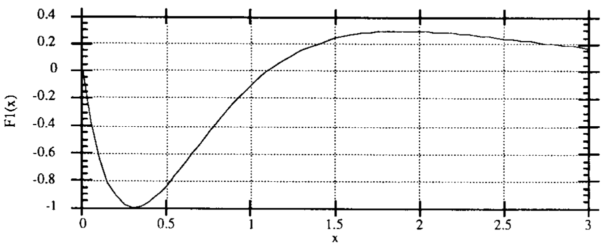


Fig. 1. Graph of $F1$.

A. Functions

Two functions were used. The first was a function of a single variable defined by

$$F1(x) = 4.26(e^{-x} - 4e^{-2x} + 3e^{-3x}), \quad x \in [0, 3]$$

and shown in the graph of Fig. 1. This function was chosen because it has been used as an example to test overfitting and underfitting of spline curves [16] and because the relative simplicity of its graph provided an easy way to observe the effects of the various evaluation criteria.

The second was a function of three variables defined by

$$F2(x_1, x_2, x_3) = x_1 e^{x_2} \cos(x_3), \quad x_1, x_2 \in [-1, 1], \\ x_3 \in [-3, 3].$$

This function models a more complex domain than the first function due to increases in both nonlinearity and dimensionality. The function $F2$ is a shortened form of the function

$$F2'(x_1, \cdots, x_{16}, t) = \sum_{i=1}^{4} x_{4i-3} e^{tx_{4i-2}} \cos(tx_{4i-1} + x_{4i})$$

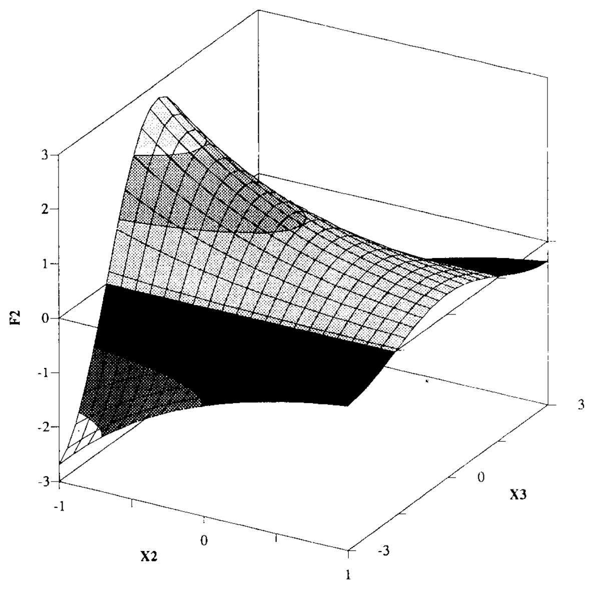
used in modeling damped vibrations such as those found in aircraft components. The shortened form, $F2$, was used to facilitate interpretation of experimental results and to reduce the size of the training sets required for the experiments.[1] Fig. 2 shows graphs of the functions formed from $F2$ by setting a) x_1 to 1 or b) x_2 to 1 and illustrates the nonlinearity involved.

B. Experimental Data

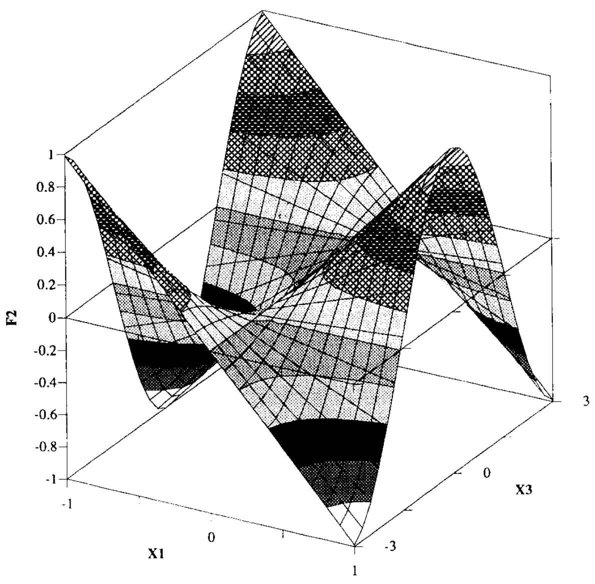
Two factors were considered when generating experimental data for each of the functions. The first was the density of the data and the second was the amount of noise in the data.

Data density is associated with the question of how much data is necessary to train a BP network. Two classes of data density were defined. The first class, DQ1, was chosen to represent data sets containing "relatively few" data points. The second class, DQ2, was chosen to represent "relatively many" data points. Table I shows the number of data points in each set for each function. The number of points in each class was chosen by looking at one or more graphs of the functions and trying to select point densities which were conservative or excessive. This was an iterative process. Points were selected and experiments performed to evaluate the effects. Points were chosen at equally spaced intervals along the respective axes.

[1] Generating data for a function defined on n-space requires points to be generated for an n-dimensional grid. This necessitates extremely large sets of training data to adequately model the domain.



(a)



(b)

Fig. 2. Graph of $F2$.

TABLE I

Data Quantity	$F1$	$F2$
DQ1	68	512
DQ2	268	8000

To determine the effect of noise on network robustness, various levels of noise were added to the training sample. The approach used was to generate data using the following

TABLE II

Noise Class	σ^2 for $F1^*$	σ^2 for $F2^*$
TR0,TE0	0	0
TR1,TE1	0.4	0.1

TABLE III

Architecture	$F1^*$	$F2^*$
$A0$	1-30-1	1-120-1
$A1$	1-10-5-1	1-30-15-1

modified functions

$$F1^*(x) = F1(x) + n(0, \sigma^2)$$
$$F2^*(x_1, x_2, x_3) = F2(x_1, x_2, x_3) + n(0, \sigma^2)$$

where $n(0, \sigma^2)$ denotes a normal random variable with mean 0 and variance σ^2. Similar amounts of noise were added to the testing sample. Table II shows the values of σ^2 used in the experiments. Again, these values were determined by experimenting with various values and observing parts of the graph of each function.

C. Network Architectures

Two basic types of network architectures were considered. The first, denoted as $A0$, will represent a network configuration of the form I-H-O where I is the number of nodes in the input layer, H the number of nodes in the single hidden layer, and O the number of output nodes. The second architecture, denoted as $A1$, represents a network configuration of the form I-$H1$-$H2$-O with two hidden layers of $H1$ and $H2$ nodes, respectively. The specific network architectures for each function are given in Table III.

Theoretical results suggest that any continuous function can be approximated to an arbitrary degree of accuracy using a single hidden layer neural network [10]. This work and the work of other authors [4] suggests that two-hidden-layer networks can often provide solutions with fewer hidden nodes and faster training times. The present paper, however, goes on to evaluate the robustness of each basic network architecture in the presence of noise. Hence, both basic types of architectures were included in these experiments. In the tables of Sections IV and V, a network with two hidden layers is called a pliable architecture, whereas one with one hidden layer is called a stiff architecture.

D. Experimental Method

The Taguchi Method of Experimental Design: To provide scientific discipline to our experimental approach, Taguchi's method of experimental design [13] was used. This method assesses which of several varying factors have the most effect on a desired outcome. Since the goal of this research was to decide which of several factors was most effective in reducing neural network errors and their variances, it seemed appropriate to use the Taguchi approach. In this section the Taguchi method is briefly described.

Suppose an experimental outcome v is a function of several variables $u_1, \cdots, u_n$ whose values can be controlled. Write $v = f(u_1, \cdots, u_n)$. The controlled variables $u_1, \cdots, u_n$ are called factors. The goal is to find those factors which, when changed, will have the most beneficial effect on v. This can be done by varying each factor independently of the others and recording the corresponding change in v or by varying the factors simultaneously in a disciplined way to also determine if the change in v by a particular factor is influenced by the values of the other factors. The Taguchi methods are disciplined ways of varying two or more factors simultaneously.

In one of the simplest of the Taguchi methods, the individual factors are varied between two values each. In a full experimental design, all possible combinations of the values of the factors must be tried. In a fractional design, a subset of the possible value combinations is used.

To conduct the experiments required by a full design using three factors, Table IV is useful. The factors are denoted by X, Y and Z, each with a Lev1 and a Lev2 value. There are eight trials of the experiment, each using the Lev1, Lev2 combination depicted by the white boxes in the X, Y, and Z columns of the row containing the trial number. For example, in Trial 5, the value of X is Lev2, Y is Lev1, and Z is Lev1. The value of the experiment for each trial is recorded in the Value column and copied to each of the white boxes along the row for the trial. Totals and averages are taken of white boxes in each column. The values in the Effect row are found by subtracting the average outcome for Lev1 values from the average outcome for Lev2 values in the X, Y, and Z columns and analogously in the other columns. This provides a numerical value for the average effect on the outcome of moving a factor from its Lev1 value to its Lev2 value. The columns labeled XY, etc. are used for measuring interaction effects. For example if the number in the Effect row of the XY column is large in absolute value, when compared to the Effects in the X, Y, and Z columns, then X and Y may interact in a beneficial or detrimental way. In this case, further analysis should be done to decide the best settings for X and Y (see [13]).

If additional factors are to be considered and the number of experiments kept at eight, other designs may be chosen. For example if there are four factors at two levels each, then the design of Table IV may be modified by replacing the XYZ column with the fourth factor, say W. The eight trials of the table are used with W having the value of the column of last white box in the row of the trial. This is now a fractional design with only eight of the possible 16 experiments being used. The drawback in using this fractional design is that the interaction column headed XY becomes a column representing the interaction effects between X and Y and also the interaction effects between Z and W. Similarly the columns headed XZ and YZ also include the interactions YW and XW, respectively. If the numbers in the Effect row of these columns are small compared to the other numbers, however, then all these interaction effects are small.

TABLE IV

Description of Experiment															
		X		Y		Z		XY		XZ		YZ		XYZ	
Trial	Value	Lev1	Lev2	Lev1	Lev2	Lev1	Lev2								
1			■		■		■	■		■		■			■
2			■		■	■		■			■		■	■	
3			■	■			■		■	■			■	■	
4			■	■		■			■		■	■			■
5		■			■		■		■		■	■		■	
6		■			■	■			■	■			■		■
7		■		■			■	■			■		■		■
8		■		■		■		■		■		■		■	
Total															
Average															
Effect	■														

The Training Process: Backpropagation [15] was used for training with weights being adjusted after each presentation of an input vector. This is called iterative training as opposed to epoch training in which weights are adjusted only after a complete pass through the training sample.

A target value for the ARMSE was set. If this value was achieved at iteration L, then training would continue for another L iterations. A maximum number of iterations was also set which would override the other stopping criterion.

Calculation of Errors and their Variances: To accurately calculate the true error and its variance, four-fold cross validation, as described previously in Section II, was used. Test results were then averaged over the four runs.

The ARMSE training and testing errors were calculated after every 10 epochs. At the end of each 10 epochs of training, the weights were frozen and the training and testing samples were run through the network. The ARMSE was calculated using the formula given in Section II. The graphs of the training and testing errors which are shown later were made from these values.

Error variances were calculated from the ARMSE errors usually over the last half of the training iterations using the formula which is presented in Section II. Normally the errors were no longer decreasing or increasing significantly during the last half of training, so the variance would measure the variability in the errors during the later stages of training, and would approximate the variability present in a fielded network.

Determination of Overfitting: Overfitting occurs when the testing error rises as the training error falls. This indicates that noise in the training sample, rather than the underlying function, is being fit by the network. Obviously, when overfitting occurs, additional training is detrimental rather than beneficial. Training should stop at that iteration when overfitting begins, assuming that this iteration can be determined. Identification of overfitting was made by plotting both training and testing ARMSE on a single graph. As expected, no overfitting was observed when there was no noise in the training data.

IV. $F1$: A Function of One Variable

A. Experimental Results

The single variable function

$$F1^*(x) = 4.26(e^{-x} - 4e^{-2x} + 3e^{-3x}) + n(0, \sigma^2), \quad x \in [0, 3]$$

was studied first. Taguchi tables were constructed to evaluate the effects the different experimental factors had on ARMSE error and its standard deviation. Averages over the four cross validation runs were calculated for both training and testing errors. The Taguchi table for average ARMSE testing error is shown as Table V.

Columns in the table correspond to various combinations of experimental parameters. The columns labeled DQ-TR,TE-A, etc. measure the amount of interdependence between the experimental factors, with two possible interdependencies confounded (see [13]) into each column.

The total ARMSE value for each experimental parameter is shown in the row labeled "Total." Experiment numbers marked with * denote experiments which were terminated when the maximum number of iterations was reached, as opposed to reaching the target training ARMSE.

The row labeled "Effect" can be used to evaluate the results of the experiments. For example, Table V indicates that for the test ARMSE averaged over the four runs of four-fold cross validation, the testing noise level is, by far, the most influential factor since the value of Effect is 0.268 which is more than double the value of any other value of Effect. Therefore, testing data noise is deemed to be a major factor influencing the magnitude of testing ARMSE.

Further evaluation of the Effect values ranks the influences of the experimental factors in the following order: TE, TR, A, and DQ. These values indicate that noise in testing data contributes most to high testing ARMSE values, noise in training data is the next largest contributor, architecture the next, and the amount of data contributes the least. Analysis of

TABLE V

ARMSE (Test) Average															
		Data Quantity		Training Noise		Testing Noise		DQ-TR,TE-A		DQ-TE,TR-A		TR-TE,DQ-A		Architecture	
Trial	Value	DQ1	DQ2	TR0	TR1	TE0	TE1							A0	A1
1	0.0218	0.0218	■	0.0218	■	0.0218	■	■	0.0218	■	0.0218	■	0.0218	0.0218	■
2	0.3514	0.3514	■	0.3514	■	■	0.3514	■	0.3514	0.3514	■	0.3514	■	■	0.3514
3*	0.2317	0.2317	■	■	0.2317	0.2317	■	0.2317	■	■	0.2317	0.2317	■	■	0.2317
4*	0.3596	0.3596	■	■	0.3596	■	0.3596	0.3596	■	0.3596	■	■	0.3596	0.3596	■
5	0.0092	■	0.0092	0.0092	■	0.0092	■	0.0092	■	0.0092	■	■	0.0092	■	0.0092
6	0.3582	■	0.3582	0.3582	■	■	0.3582	0.3582	■	■	0.3582	0.3582	■	0.3582	■
7*	0.1165	■	0.1165	■	0.1165	0.1165	■	■	0.1165	0.1165	■	0.1165	■	0.1165	■
8*	0.3826	■	0.3826	■	0.3826	■	0.3826	■	0.3826	■	0.3826	■	0.3826	■	0.3826
Total	1.8309	0.9644	0.8665	0.7406	1.0904	0.3792	1.4517	0.9587	0.8722	0.8366	0.9943	1.0578	0.7731	0.8561	0.9748
Average	0.2289	0.2411	0.2166	0.1851	0.2726	0.0948	0.3629	0.2397	0.2181	0.2092	0.2486	0.2644	0.1933	0.2140	0.2437
Effect	■	-0.0245		0.0875		0.2681		-0.0216		0.0394		-0.0712		0.0297	

TABLE VI

Low Test Mean	Low Train Mean	Low Test Std Dev	Low Train Std Dev
Low Testing Noise 99 Low Training Noise 95	Low Training Noise 99 High Data Quantity 95 Pliable Architecture 90	Low Training Noise 99 High Testing Noise 95	High Data Quantity Stiff Architecture

TABLE VII

Training Noise	Testing Noise	Average Testing ARMSE
TR0	TE0	0.0155
TR1	TE0	0.1741
TR0	TE1	0.3548
TR1	TE1	0.3711

variance techniques ([14, Chapter 3] or [13, Chapter 8]) using the F-test reveal that testing noise is statistically significant at the 99% level and training noise is significant at the 95% level. Data quantity and architecture are statistically insignificant. The conclusion is that for the settings of the factors which were used, decreasing testing data noise and, less significantly, decreasing training data noise, will be effective in decreasing the testing error; whereas changing the architecture or the data quantity will have little effect. The sign of the Effect values, however, indicate that using a single-hidden-layer architecture or a large data quantity probably would decrease testing errors some.

Table VI summarizes both the training and testing ARMSE errors and their standard deviations for all experiments performed on function $F1^*$. Each column ranks the factors that were found effective in obtaining the goal at the top of that column. For each column, eight experiments were run and a Taguchi table constructed to obtain the results listed. For example, Table V was used to obtain the first column.

B. Discussion of Results

Effect of Noise in Training: Table VII shows the average ARMSE values for each of the possible combinations of train/test noise from Table V. For example, the value in the TR0, TE0 row of Table VII is the average of the values for Trials 1 and 5 in Table V. Evaluation of the tests confirm what one suspects about noise. Noise-free data produced the best results while noise in both training and testing data produced the worst results. In the presence of testing noise, it appears that a slightly lower average ARMSE is obtained with this function by training the model with noise-free data.

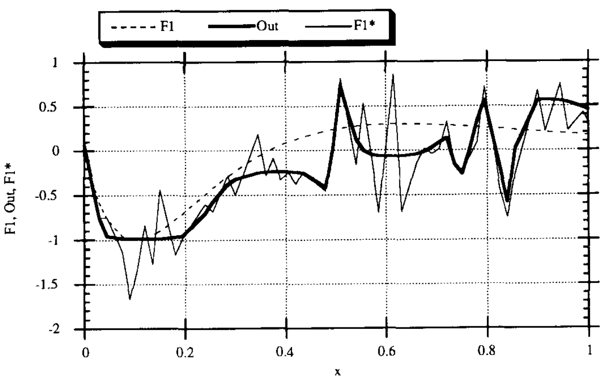


Fig. 3. Plots of $F1$ vs Out vs $F1^*$ generated by a network trained from $F1^*$ with variance of 0.4.

It is important to note that for these experiments, the noise in the training data was quite severe. In fact, the noise in the training data essentially dominated the samples and changed the problem being modeled. Fig. 3 illustrates this concept. Note the graph of the original function, $F1$, in comparison to that of $F1^*$ with $\sigma^2 = 0.4$. Noise has made $F1^*$ a very different function than $F1$! The effect of the trained neural network is to smooth out the results of $F1^*$. This new function, however, is still very different than the original function being modeled.

While an analytical expression for the function of interest is usually unknown, the graphing approach provides a valuable technique for estimating responses of a trained neural network for unseen test data. The approach is to produce a graph

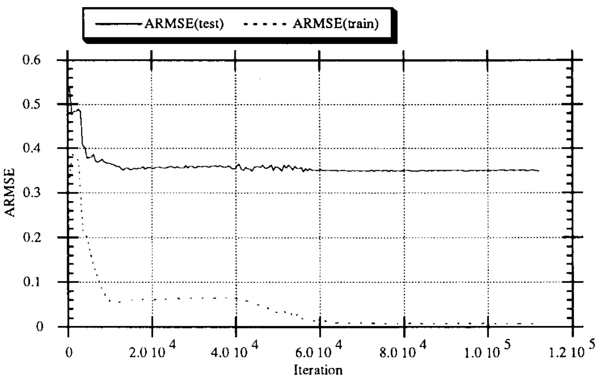


Fig. 4. Results of a DQ1/TR0/TE1/A1 test.

showing the training data and the network output as shown in Fig. 3. If the network does a poor job of fitting the data, as is the case in Fig. 3, it is highly likely that it will not perform well on unseen data. For multivariate functions, sections of the graph in each plane could be reviewed [3].

Effect of Noise in Testing: Similar statements can be made about noise in the testing data. The more noise in the testing data, the less likely the network represents the data being tested. Again, plots of training and testing data along with neural network output provide an indicator of the type of performance that can be expected from the network. Fig. 4 shows the results of one of the DQ1/TR0/TE1/A1 tests.[2] The network learns the training data reasonably well. The noise in the test data causes it to appear to be from a different model than that represented by the training data. The test error remains consistently poor through the training cycle.

Effect of Data Density: Evaluation of Table V indicates that DQ2 data density produces smaller training and testing ARMSE's than DQ1. In addition, the DQ2 values have a smaller standard deviation. This result suggests that increasing training data density decreases testing error.

Effect of Architecture: For this function, the differences in ARMSE averages and standard deviations between the $A0$ and $A1$ architectures is very small for both testing and training data. The single hidden-layer architecture, $A0$, produced a slightly smaller ARMSE average and standard deviation on test data. This is not surprising in view of the simple domain of the function.

The amount of training required for each of the architectures was approximately the same. Four of the eight test sets did not converge. Half of them, however, were architecture $A0$ while the other half were $A1$.[3]

Overfitting: Fig. 5 illustrates a situation in which overfitting occurs. The top graph shows training ARMSE while the bottom shows testing ARMSE. Training ARMSE suddenly increases at about 2.6E6 iterations through the training data. This is possible due to the way in which network training

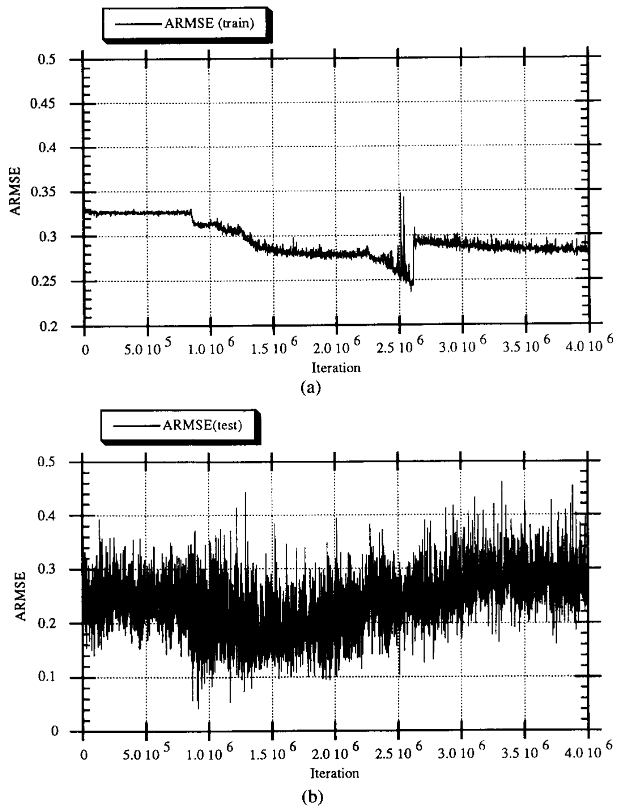


Fig. 5. Overfitting in a DQ1/TR1/TE0/A1 test.

was implemented. After this point, it again begins to decline. Throughout this cycle, the testing ARMSE is steadily increasing. Hence, the network is doing a good job learning the training data even though it is not the correct function.

The differences between noise and overfitting are not always distinct. For example, the phenomenon exhibited in Fig. 4 is not overfitting. The poor test results are caused by noise in the testing data, not the training data. One marked difference between the results is that, in overfitting, testing error continues to increase as training continues. In cases like that shown in Fig. 4, the testing error becomes more and more constant as training continues.

C. An Experiment with Long Training Time

It was decided to train the network which approximates the no-noise version of $F1$ for a very long time to discover changes which may not be observed in the shorter runs. The result is shown in Fig. 6. The network trained for over 13 million iterations, many times the usual run time. The following observations are evident from the graph.

- The training error, although initially above the testing error, eventually drops below, as expected. Generalizing, for a function with no or little noise, if testing error is on average better than training error, then a network may not have trained long enough.

[2] The quadruple represents the combination of experimental factors arranged in the form: Data quantity/level of training noise/level of testing noise/network architecture.

[3] Recall that this lack of training convergence was caused by the excessive noise in the training data.

TABLE VIII

Low Test Mean	Low Train Mean	Low Test Std Dev	Low Train Std Dev
Low Training Noise 95 High Data Quantity	Low Training Noise 99 Pliable Architecture	Low Training Noise 95	Low Training Noise 99 High Data Quantity 95

TABLE IX

Low Test Mean	Low Train Mean	Low Test Std Dev	Low Train Std Dev
Low Training Noise 95 High Data Quantity 90	Low Training Noise 99 Pliable Architecture	Low Training Noise 99	Low Training Noise 95 High Data Quantity 90

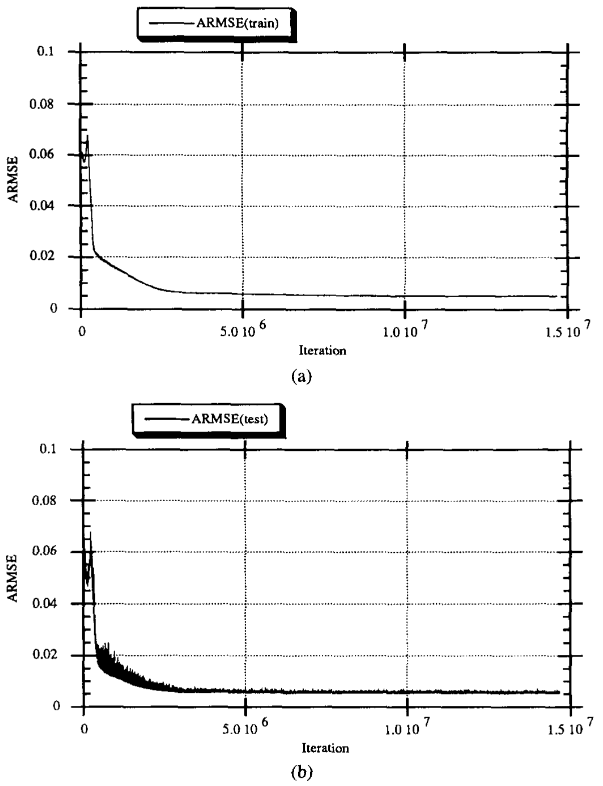


Fig. 6. Result of "infinite" training.

- Testing error variance is high as long as training error is decreasing rapidly as shown in Fig. 6 when the number of iterations is less than 3×10^6. When training error levels out, testing error variance stabilizes at a lower lever.
- The testing error continues to decrease during the entire test. The network finds it difficult to learn the sharp curve in the function, but does continue to slowly do so.

V. $F2$: A Function of Three Variables

A. Experimental Results

The multi-variable function

$$F2'(x_1, \cdots, x_{16}, t) = \sum_{i=1}^{4} x_{4i-3} e^{tx_{4i-2}} \cos(tx_{4i-1} + x_{4i})$$

was selected to evaluate the error characteristics of a neural network under the conditions of a more nonlinear mapping and a higher dimensional input space. Additionally, this function provided a transition of this work to a "real-world" task. As mentioned in Section III-A, this function is used in modeling damped vibrations in aircraft components.

In an effort to more completely understand, visualize, and control the data generated by this function, it was slightly reduced and converted to the form

$$F2^*(x_1, x_2, x_3) = x_1 e^{x_2} \cos(x_3) + n(0, \sigma^2),$$
$$x_1, x_2 \in [-1, 1],\ x_3 \in [-3, 3].$$

As with $F1$, differing data quantities, data noise levels, and network architectures were evaluated using this data. Again cross validation and Taguchi methods were employed.

The following tables summarize the results of the Taguchi experiments on $F2^*$. For this function, several Taguchi tables were generated.

Table VIII was generated from four full three-factor Taguchi tables which were each similar to the one shown in Table V for $F1^*$. For Table VIII, all combinations of data quantity, training noise, and network architecture were evaluated over the last 250 000 iterations of the training process. So, if training to a 0.02 ARMSE error level required 1 million iterations, the training would be continued until 2 million iterations had occurred and the statistics would be collected during the iterations from 1 750 000 to 2 million. It was hoped that this method for collecting statistics would provide a better interpretation of the "in-the-field" performance of the network. It is interesting to note that if those statistics had been collected over the last half of the iterations, as was the case for $F1^*$, Table IX would have resulted. The main difference between these tables is a slightly reduced reliance on training noise in determining training standard deviation and testing mean.

Table X results from the study of all four factors on $F2^*$. The form of the Taguchi table was the same as was used for $F1^*$, however, this table again only tracked the statistics over the last 250 000 iterations. Table XI provides the statistics for the last half of the data. Again, the main difference between these tables is a slightly reduced reliance on training noise in determining training standard deviation.

B. Discussion of Results

Effect of Noise in the Training Data: As in Experiment 1, with $F1^*$, the combination of noise in the training data

TABLE X

Low Test Mean	Low Train Mean	Low Test Std Dev	Low Train Std Dev
Low Testing Noise 95	Low Training Noise 99	Low Training Noise 95	Low Training Noise 99
Low Training Noise 95	Pliable Architecture	High Testing Noise	High Data Quantity
High Data Quantity 90		Pliable Architecture	

TABLE XI

Low Test Mean	Low Train Mean	Low Test Std Dev	Low Train Std Dev
Low Testing Noise 95	Low Training Noise 99	Low Training Noise 99	Low Training Noise 99
Low Training Noise 95	Pliable Architecture 90	High Testing Noise 95	High Data Quantity 95
High Data Quantity 90			

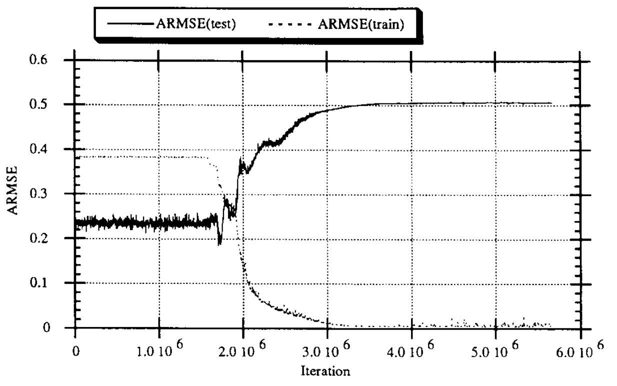


Fig. 7. Training noise variance = 0.2; 125 data points.

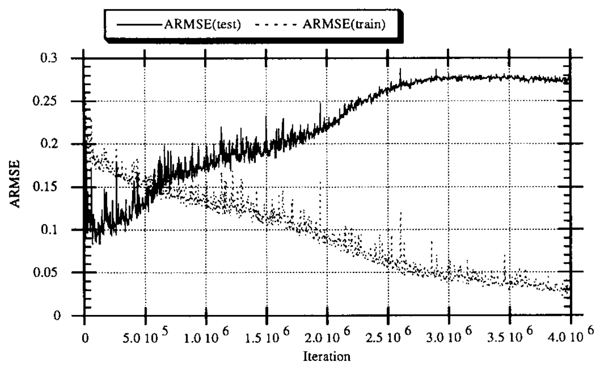


Fig. 8. Training noise variance = 0.2; 512 data points.

and a network with high flexibility resulted in overfitting. Additionally, training noise was the most significant factor in both the final testing error and its variance.

Another significant lesson learned from this data set focuses on the magnitude of the noise. Consider Fig. 7. In this run, the magnitude of the noise which was added to the training data was equivalent to half the magnitude of noise added to the data in Experiment 1. There is no noise in the testing data. Notice how the testing error significantly increases as the training error significantly decreases.

Fig. 8 represents the training and testing performance when the amount of data used for training was increased from 125 data points to 512 data points. Testing performance is now significantly better since the minimum testing error differs from the final testing error by ~0.17, versus a difference of ~0.30 for the testing graph of Fig. 7. The training performance has diminished, however. Overfitting is not as significant. The increase in the number of data points provided additional information about the underlying function and allowed the network to compensate for the noise.

Now consider Fig. 9. Here the noise is an order of magnitude less ($\sigma^2 = 0.1$) and 512 training cases were used. The testing performance has again improved while the training performance has declined. Overfitting is still present, but its effect has decreased to the point where the minimum testing error and the final testing error only differ by ~0.04.

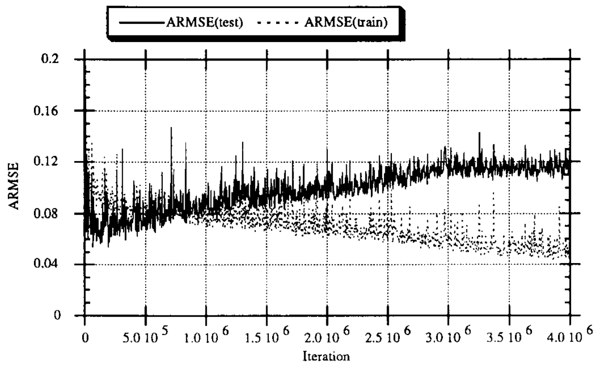


Fig. 9. Training noise variance = 0.1; 512 data points.

It was concluded that the effect of noise in Fig. 7 was too extreme and so severe that the underlying function was no longer a factor in the training data. Testing on a no-noise case produced results which were uncorrelated with training results, as if one function was being used for training and a completely different function was being used for testing. Fig. 8 shows how additional training data offered some compensation for the additional noise, but "acceptable" performance in the presence of overfitting was finally achieved by decreasing the noise level and including additional training points as shown in Fig. 9. A tentative conclusion based on the necessity of reducing the noise level when going from $F1$ to $F2$, is that the amount of noise that can be tolerated is inversely related to the variability of the function being approximated.

Effect of Noise in the Testing Data: These results are similar to those achieved for the first function as would be expected, testing noise is positively correlated with the average testing error, however higher testing noise was slightly negatively correlated with the testing error's variance. Given careful consideration and with the knowledge of hindsight, the common sense nature of these results becomes apparent.

Effect of Data Density: Again, the results are similar to those for $F1^*$. Increased data density offered some compensation for the existence of noise in the training set by producing a lower mean testing error. This fact is key in understanding the cause of overfitting and how it can be reduced.

Effect of Architecture: For this data set, the effect of network architecture was mainly significant in producing a low training mean. The Taguchi tables, however, do not accurately represent the relationship between network architecture and overfitting, since overfitting was not one of the experimental factors.

Overfitting: It is the general conclusion of the $F2^*$ experiments that several factors must exist for overfitting to occur. In turn each of these factors offers a method of reducing overfitting.

First, noise must be present in the training data for overfitting to occur. As stated earlier, overfitting results from learning the nuances of each sample pair which distances it from the underlying function being approximated. If the sample pairs contain no such abnormalities (i.e., no noise), such effects will not occur. In no case and with no amount of extended training did the neural network begin to produce wild interpolations between the noiseless training points. The networks consistently produced approximations which were no more nonlinear than the training data.

Second, for overfitting to occur the network architecture must be overly pliable for the function being learned. As with any polynomial interpolation technique, the degree of the polynomial for which the coefficients are being sought, determines the smoothness/flexibility of the resulting approximation. For neural networks, the number of weights is not the determining factor in the flexibility of a network. The arrangement of the weights, the architecture of the network, is the deciding factor. Drastically differing results were achieved between one and two hidden layer networks even though the number of weights was equivalent in both cases. The two-hidden-layer network was consistently quicker in fitting the training data. In the presence of noise, this implies quicker overfitting.

Third, training data quantity must be low for overfitting to occur. Experiments on this function demonstrated a consistent tie between data quantity and noise levels. As the tables show, increasing the data sampling rate decreased the mean testing noise level. When observing only those cases where noise was present in the training data and the more pliable architecture was used, the effects of data quantity become even more obvious.

Overfitting can be monitored by periodically evaluating a network using testing data during the training process. If

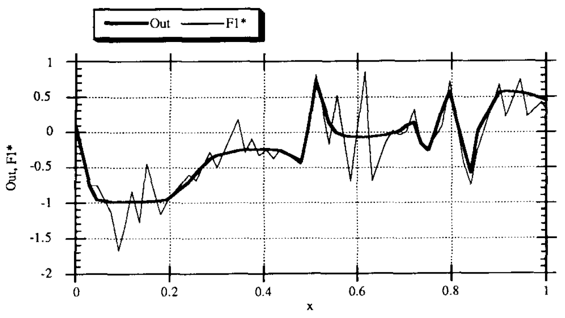


Fig. 10. Two-hidden-layer approximation for $F1^*$ with significant noise and sparse training data.

overfitting occurs, these results provide insight into possible causes and suggest several possible ways to reduce the effect of overfitting. First, reduce training noise if possible. Second, reduce the network architecture. Do not simply reduce the number of weights, but consider a simpler design. Third, increase the number of training samples. Taken together, as shown in Figs. 7–9, the improvement in performance can be phenomenal.

VI. Guidelines for the Design and Construction of Neural Networks

Some guidelines for the design and construction of neural networks which were suggested by the experiments are summarized in the following paragraphs.

1) Use a training sample that is as free of noise as possible. Some researchers have suggested [9] that better generalization will occur if noise is added to the training data. These experiments, however, suggest just the opposite: the larger the amount of noise in the training data, the larger the testing error and its variance.
2) Use dense training data. To decrease errors, dense data is better than sparse data. Denser data means more data, however, and a penalty is paid in training time.
3) For best error control, use networks with two hidden layers. This research focused on network architectures with a single hidden layer or with two hidden layers. Both architectures had nearly the same number of weights. For example, a network with one hidden layer of 120 nodes and another network with two hidden layers of 30 and 15 nodes, respectively, were used in the experiments with $F2$. The primary question was which of these is better for error control. We found that the network with two hidden layers was invariably more flexible than the one with one hidden layer. In general this allowed the two-hidden layer networks to fit the function closer and to give smaller errors.
4) For best smoothness of the approximating function, use networks with one hidden layer or use a dense training sample. Smoothness of the approximating function could be sacrificed when two layers were used. For example, the two-hidden-layer approximating function for $F1^*$ is shown in Fig. 10 and the one-hidden-layer approximating function for the same function is shown in Fig. 11.

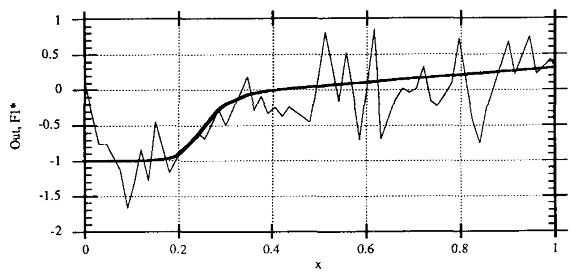


Fig. 11. One-hidden-layer approximation for $F1^*$ with significant noise and sparse training data.

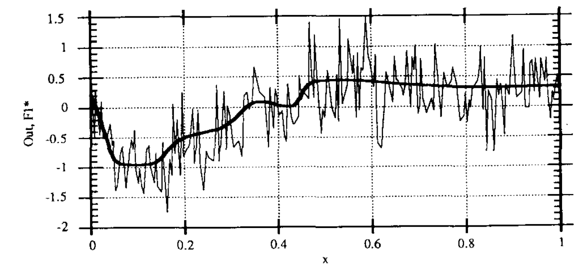


Fig. 12. Two-hidden-layer approximation for $F1^*$ with significant noise and dense training data.

The two-hidden-layer approximation has more abrupt changes and much steeper slopes than the one-hidden-layer approximation.

Examination of graphs indicated that when the density of points in the training sample for the two-hidden-layer approximation was significantly increased, the approximation function became smoother as shown in Fig. 12. Because of the increased number of data points, the network was not flexible enough to move sharply up and down fitting the errors, and, therefore, the best fit was a smooth interpolation through roughly the center of the noise.

5) If convergence is not occurring, start over with different, perhaps much larger, random initial weights. The initial choice of weights is a significant factor in the speed of training and, in some cases, whether satisfactory convergence takes place at all. Consider Figs. 13 and 14. In Fig. 13, initial weights between $-.1$ and $+.1$ were chosen. Note that decreases in the training error did not occur at all. When weights were initialized randomly between -1.0 and $+1.0$, as in Fig. 14, however, convergence was immediate and steep.
6) Stop training when the error on the testing data begins to rise. Overfitting occurred when there was significant noise in the training data, especially if the data set was not very dense. The best results will be obtained if training is stopped just at the point that overfitting begins. To determine this point, it is best to divide the data sample into training and testing portions, perhaps using n-fold cross validation. Then the testing error can be watched and training stopped when it begins to rise.

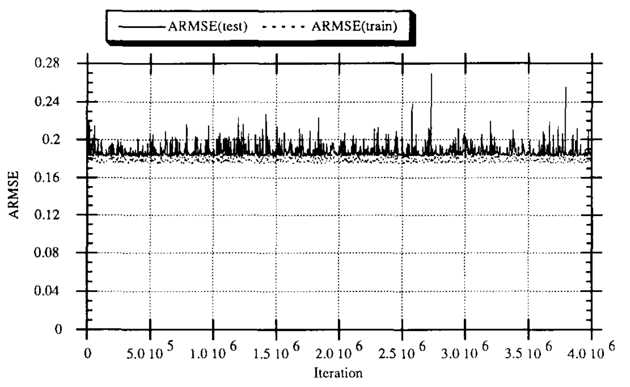


Fig. 13. $F2$ training and testing errors with small initial weights.

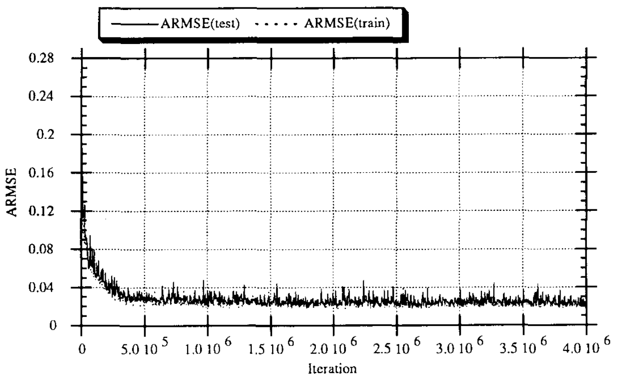


Fig. 14. $F2$ training and testing errors with large initial weights.

7) If there is no testing data, stop training at the bottom of the initial steep drop. If it is not feasible to have both testing and training data, then it may still be possible to estimate when overfitting begins. If there is a steep drop in the training error, then overfitting frequently begins near where the error curve begins to level out. Typical training and testing error curves which illustrate this phenomenon are shown in Fig. 15.
8) To decrease the variance of the testing error, decrease the noise in the training sample. The Taguchi tables indicated that changing the architecture or even the density of the training data had little effect on the variance of the testing error. The only effective way to decrease it, using the factors considered here, is to decrease the noise in the training data. Unfortunately, it may not be easy or even possible to control this noise.

VII. Conclusion

In this section there is a brief description of some overall reactions to the approach taken, and mention of some suggestions for further research.

There was an unexpected wide variation of behaviors in the neural networks used in the experiments. For example, the significant differences between networks with one and two hidden layers was unexpected. It was unexpected to see as much variability in network errors—for example, testing errors

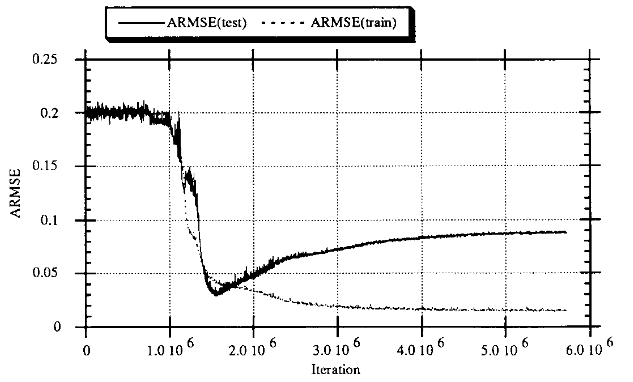


Fig. 15. $F2$ training and testing errors illustrating overfitting beginning near the bottom of a steep drop in the training error.

sometimes below training errors. The profound difference in convergence between networks initialized with small and large weights was a surprise. In some cases, testing errors would begin to rise as if overfitting was occurring, but later would again fall. Thus, the moral: since relatively little is known about the capabilities and characteristics of neural networks, controlled experiments under varying circumstances must be run since the outcome of a single experiment can be misleading.

The Taguchi method of experimental design for controlling experiments was beneficial in pinpointing those features of the design process that would be most helpful in controlling network errors once the levels of the factors had been set. Taguchi, however, does not help with the appropriate levels to use for the factors. A more sophisticated type of experimental design such as response surface methodology [2] may overcome this limitation.

Perhaps the most significant omission in this research is a study of the value of optimal architectures in error control. Methods for finding optimal architectures include adding a term of the form λQ, where Q measures the complexity of the network, to the ARMSE calculation [18], [19], and pruning unnecessary weights from a trained network [8], [7]. Intuitively, a network of optimum size will be just flexible enough to learn the function, but not the errors, so overfitting should not occur and generalization performance should be improved. A Taguchi analysis could consider whether using an optimum architecture is better than using a very flexible architecture and stopping training when overfitting starts, a question which was considered in [5]. Additional studies could analyze the effect of weight decay, by varying between two levels of λ in the complexity term λQ.

REFERENCES

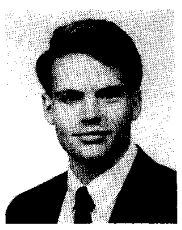
[1] H. Akaike, "Statistical predictor identification," *Ann. Inst. Statist. Math.*, vol. 22, pp. 203–217, 1970.
[2] G. E. P. Box and N. R. Draper, *Empirical Model-Building and Response Surfaces.* New York: Wiley, 1987.
[3] John M. Chambers, William S. Cleveland, Beat Kleiner, and Paul A. Tukey, *Graphical Methods for Data Analysis* Pacific Grove, CA: Wadsworth and Brooks/Cole, 1983.
[4] Daniel L. Chester, "Why two hidden layers are better than one," in *Proc. Int. Joint Conf. Neural Networks*, vol. 1, 1990, pp. 265–268.
[5] W. Finnoff, F. Hergert, and H. G. Zimmermann, "Extended regularization methods for nonconvergent model selection," *Advances in Neural Information Processing V*, C. L. Giles, S. J. Hanson, and J. D. Cowan, Eds. San Mateo, CA: Morgan Kaufmann, pp. 228–235, 1993.
[6] Stuart Geman, Elie Bienenstock, and René Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, pp. 1–58, 1992.
[7] Masafumi Hagiwara, "Novel backpropagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection," in *Proc. Int. Joint Conf. Neural Networks*, vol. I, 1990, pp. 625–630.
[8] Babak Hassibi and David G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," *Neural Inform. Processing Syst.*, vol. 4, 1992.
[9] Lasse Holmström, "Using additive noise in backpropagation training," *IEEE Trans. Neural Networks*, vol. 3, no. 1, pp. 24–38, Jan. 1992.
[10] Kurt Hornik, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.
[11] Kevin Knight, "Connectionist ideas and algorithms," *Commun. ACM*, vol. 33, no. 11, pp. 59–74, Nov. 1990.
[12] Richard P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, pp. 4–22, Apr. 1987.
[13] Robert H. Lochner and Joseph E. Matar, *Designing for Quality, An Introduction to the Best of Taguchi and Western Methods of Statistical Experimental Design* Milwaukee, WS: ASQC Quality Press, 1990.
[14] Phillip J. Ross, *Taguchi Techniques for Quality Engineering, Loss Function, Orthogonal Experiments, Parameter and Tolerance Design.* New York: McGraw-Hill, 1988.
[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, vol. 1. Cambridge, MA: MIT Press, 1986.
[16] Grace Wahba, *Spline Models for Observational Data.* Society Industrial Appl. Math., 1990.
[17] Sholom Weiss and Casimir Kulikowski, *Computer Systems that Learn.* San Mateo, CA: Morgan Kaufmann, 1991.
[18] Andreas S. Weigand, David E. Rumelhart, and Bernardo A. Huberman, "Backpropagation, weight-elimination and time series prediction," in *Proc. 1990 Connectionist Models Summer School*, 1990, pp. 105–116.
[19] ——, "Generalization by weight-elimination applied to currency exchange rate prediction," in *Proc. Int. Joint Conf. Neural Networks*, vol. I, 1991, pp. 837–841.

Gerald E. Peterson received the B.S., M.A., and Ph.D. degrees in mathematics from the University of Utah, Salt Lake City, in 1961, 1963, and 1965, respectively.

He was an Associate Professor in the Department of Mathematics and Computer Science at the University of Missouri-St. Louis from 1975 to 1983. From 1983 to 1986 he was Professor of Computer Science at Southern Illinois University at Edwardsville. In 1986 he joined the McDonnell Douglas Corporation and is currently a Senior Principal Technical Specialist. His research interests include online identification of aerodynamic coefficients using memory-based and neural techniques, and techniques for improving the accuracy of derivatives of neural networks. He is the author of a two-volume IEEE tutorial, *Object Oriented Computing*.

Daniel C. St. Clair received the B.S. degree in mathematics from Culver-Stockton College, Canton, MO, in 1965, the M.S. degree in mathematics from the College of William and Mary, Williamsburg, VA, in 1969, and the Ph.D. degree in mathematics with emphasis in computer science from the University of Missouri-Rolla, St. Louis, in 1975.

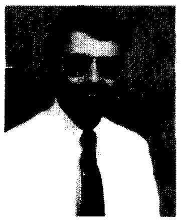
He is Professor of Computer Science at the University of Missouri-Rolla's Engineering Education Center in St. Louis. His research interests include the development of intelligent systems through machine learning and neural network algorithms.



Stephen R. Aylward received the B.S. degree in computer science from Purdue University, West Lafayette, IN, in 1988, and the M.S. degree in computer science from the Georgia Institute of Technology, Atlanta, in 1989.

He worked in the Neural Network Support Laboratory at the McDonnell Douglas Corporation in St. Louis from 1989 to 1993. He is currently a Ph.D. student in the Department of Computer Science at the University of North Carolina at Chapel Hill as part of an educational leave of absence from McDonnell Douglas. His research interests include image processing, segmentation, and object recognition for medical imaging systems.

Mr. Aylward is President of the Triangle Area Neural Network Society.



William E. Bond received the B.S., M.Eng., and Ph.D. degrees in structural engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1975, 1976, and 1980, respectively.

He has worked for McDonnell Douglas since 1979 and on artificial intelligence applications since 1986. He is a Senior Principal Technical Specialist in the New Aircraft and Missile Products division of McDonnell Douglas Aerospace. His work focuses on the development and application of techniques in machine learning and neural networks to aircraft subsystems.