Limiting Fault-Induced Output Errors In ANN's.

Reed D. Clay and Carlo H. Séquin
TR-91-025
April, 1991

Abstract

The worst case output errors produced by the failure of a hidden neuron in layered feed-forward ANN's are investigated. These errors can be much worse than simply the loss of the contribution of a neuron whose output goes to zero. A much larger erroneous signal can be produced when the failure sets the value of the hidden neuron to one of the power supply voltages.

A new method is investigated that limits the fractional error in the output signal of a feed-forward net due to such saturated hidden unit faults in analog function approximation tasks. The number of hidden units is significantly increased, and the maximal contribution of each unit is limited to a small fraction of the net output signal. To achieve a large localized output signal, several Gaussian hidden units are moved into the same location in the input domain and the gain of the linear summing output unit is suitably adjusted. Since the contribution of each unit is equal in magnitude, there is only a modest error under any possible failure mode.

LIMITING FAULT-INDUCED OUTPUT ERRORS IN ANN's.

Reed D. Clay and Carlo H. Séquin[1]

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

ABSTRACT

The worst case output errors produced by the failure of a hidden neuron in layered feed-forward ANN's are investigated. These errors can be much worse than simply the loss of the contribution of a neuron whose output goes to zero. A much larger erroneous signal can be produced when the failure sets the value of the hidden neuron to one of the power supply voltages.

A new method is investigated that limits the fractional error in the output signal of a feed-forward net due to such saturated hidden unit faults in analog function approximation tasks. The number of hidden units is significantly increased, and the maximal contribution of each unit is limited to a small fraction of the net output signal. To achieve a large localized output signal, several Gaussian hidden units are moved into the same location in the input domain and the gain of the linear summing output unit is suitably adjusted. Since the contribution of each unit is equal in magnitude, there is only a modest error under any possible failure mode.

1. INTRODUCTION

In related work [1,2], we have investigated many issues of fault tolerance in Artificial Neural Networks (ANN's). A key concept discussed in these papers is that fault tolerance is not inherent in ANN's. While neural nets do exhibit "graceful degradation" in the sense that they do not completely break down in the presence of a single fault (as opposed to standard computers), mechanisms must be added to provide ANN's with the capability to automatically "learn" a task with a certain degree of fault tolerance.

In this paper, we present and examine a technique for achieving some degree of fault tolerance for networks that are trained to approximate and interpolate analog functions. Other people have studied the effect of failures of hidden neurons on the output signal and have tried to minimize this effect [3]. However, they only considered the case where the fault produced an error that resulted in the loss of the contribution of a hidden unit. The case where a failure causes a hidden unit to be stuck at some nonzero value may produce a larger error over a much wider domain. We present a new technique to limit errors for hidden units that fail in arbitrary ways, and we only make the simple assumption that the output error of a neuron is bounded by the most positive and most negative power supply values.

For this study, we concentrate on faults in hidden units. Faults in the input or output units would still be catastrophic. However since the number of hidden units is typically much larger, faults are statistically more likely to occur in these intermediate layers.

[1]The main part of this work was done while the author was on sabbatical at ICSL

2. AN EXAMPLE PROBLEM

We first used the standard "backpropagation" algorithm of [4] to train a small 3 layer network with a single input and a single output unit to learn the simple function shown in Figure 1a. We then tried a variation of this algorithm where the "sigmoidal" hidden units were replaced with "Gaussian" hidden units. The output of a sigmoidal, or logistic, unit is an "s" shaped curved based on the sum of the weighted inputs coming into the unit. The output of a Gaussian unit is a localized "bump" defined by its width, height, and center location. We have found that these localized "basis functions" considerably reduce the training time for a network to learn to approximate analog functions [2].

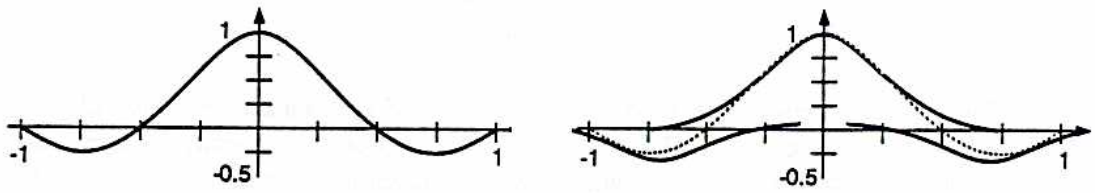


Figure 1: *(a) Target function Sinc(x) = sin(2πx)/2πx for x∈[-1,1] and, (b) its decomposition into a sum of 3 Gaussians.*

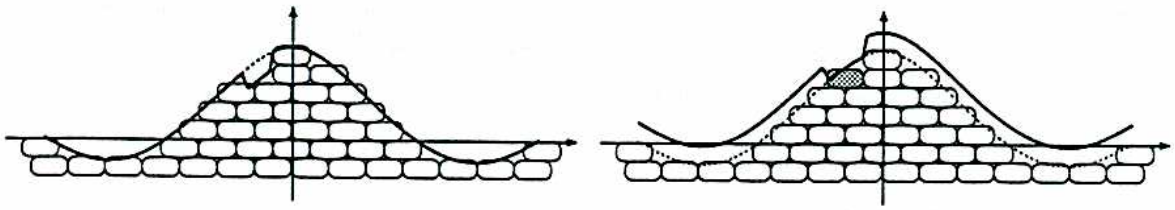
A typical solution to the sinc(x) function with three Gaussian hidden units is shown in Figure 1b. In this tutorial example, the output of the network is the weighted sum of three Gaussian hidden units with the center unit having the largest weight. The maximal error is only 0.04. However, this network is very sensitive to faults. For example, if there was a fault on the weight of the center hidden unit causing it to be stuck at zero, the resulting error around the center of the function would be 1.0 which is 100% of the peak of the output signal. A large percentage error can also result if the center Gaussian gets stuck at a value of +1 over the entire range, or if the weight of one of the side units changes from its current value of -0.2 to +1. In the latter case, a unit that makes only a small contribution to the actual output signal could still produce a much larger error signal if it or its associate weight fails in an unfortunate manner. Our solution is to add many more hidden units that each make a small contribution to the overall output and that also produce limited errors in case they should fail.

3. THE "BRICKS MODEL" SOLUTION

3.1. Basic Idea of the "Bricks Model"

A conceptual way to visualize our approach is shown in Figure 2. The output function to be learned is composed of many small contributions that are of about the same size -- the so called "bricks". A missing brick, i.e., a hidden unit that produces zero output has a very limited effect on the output signal (Fig. 2a), in width as well as in amplitude. On the other hand, a hidden unit that is stuck at a wrong output value can have an effect over the whole domain. It is thus necessary to set the peak output value from the hidden units to be comparable to the maximum value that such a unit can deliver under all failure modes, i.e., it should be close to the power supply voltages. These units are then connected directly to the output unit without any attenuating weights, thus further avoiding the possibility that the error signals of such a unit can be significantly larger than the peak contribution to the desired output signal. With these precautions, a unit stuck at peak output value will have an effect that may

spread over the whole domain but which will be limited in amplitude (Fig. 2b).



(a) *(b)*

Figure 2: *(a) "Bricks" approximation to Sinc(x) with a stuck at 0 fault (= missing brick) and, (b) a stuck at 1 fault that affects the entire domain.*

With the above scheme, the maximum amplitude error made by any single hidden unit failure is defined by the fractional contribution that each of them make with respect to the peak output signal, and that fractional contribution is implicitly defined by the total number of hidden units; the more units we use, the more we can restrict the possible error signals. This fault tolerance scheme is most effective when all units make the same contribution, hence the notion of uniform "bricks" that build up the output function. Since there is no way to influence the height of individual hidden units, the only way to achieve a higher output value is to pile more bricks into the same location. Thus the only individual variable is the centers of all the Gaussian basis functions. A modified version of the backpropagation algorithm adjusts these individual locations to minimize the error in the output. In addition, the algorithm adjusts global gain and bias values in the output unit, as well as a globally defined width for all the "bricks".

3.2. Implementation of the "Bricks Model"

In our exploratory implementation of this technique, the "bricks" are modeled by n-dimensional symmetrical Gaussian "humps". All Gaussians have the same width and height and their centers are defined by n-dimensional points, where n is the number of input units. For reasons discussed above, the Gaussian functions use the full amplitude range available at the level of the hidden units and the overall gain is adjusted in the output unit so that the overall function has the right amplitude. Because of this normalization, the maximal error due to a fault of a hidden unit cannot be larger than the peak signal it produces under normal operation.

The network operates as follows: The n input units are set and each Gaussian calculates its output based on the global width parameter of the Gaussians and on the distance between the individual center of the Gaussians and the input point (in n-dimensional space). The output unit sums the values of all the Gaussians for this location, multiplies this value by the global height parameter, and then adds a bias term to produce the output value of the network.

Mathematically, the network performs the following calculation:

$$F = b + h\sum_{i=1}^{g} G(w,c_i) \quad \text{and} \quad G(w,c_i) = \exp\left[-\frac{1}{2w^2}\sum_{j=1}^{n}(x_j - c_{ij})^2\right]$$

where F is the output of the network, b is the bias term of the output unit, h is the global height parameter, w is the global width parameter, c_{ij} is the j^{th} dimension of the center of the i^{th} Gaussian, g is the number of Gaussian hidden units, and n is the number of input units.

The four parameters are updated so as to minimize the function $E = (T-F)^2/2$, where T is the target value we would like the network to output. We use a backpropagation scheme that approximates a gradient descent. This means that we change each parameter, p, according to $\Delta p = -k\delta E/\delta p$, where k is a small constant typically called the learning rate. The actual update rules are:

$$b = \min(T)$$

$$\Delta h = k(T-F)\frac{1}{g}\sum G(w,c_i)$$

$$\Delta w = k(T-F)h\frac{1}{g}\sum(G(w,c_i)(\sum(x_j-c_{ij})^2/w^3))$$

$$\Delta c_{ij} = k(T-F)h\ G(w,c_i)(x_j-c_{ij})/w^2$$

Note that these equations differ slightly from the classical gradient descent update rules. First, since the height of all the units must be the same, we specify that all hidden units make a positive contribution and we set the bias, b, to the minimum target value that the network has been presented with. We found that with the classical update rules, the height and width parameters changed much faster than the Gaussian center parameters. The reason for this can be understood by noting that while the Δc_{ij} parameter concerns each individual center, the Δh and Δw parameters affect *all* the individual units. So, to keep the change rate of the two global terms in line with those of the center locations, we divide the global changes by g, the total number of Gaussian hidden units.

A slight variation in the above update rules can allow the user to specify the maximum error that can occur due to a single fault. One way to do this is by setting the global height to either a fixed absolute value or to a certain fraction (say 10%) of the observed peak output signal. The global width parameter is then adjusted by the algorithm to match the sum of the volumes of all the "bricks" with the total volume under the goal function.

3.3. Simulations of "Bricks Model"

We demonstrate the operation of the "Bricks" model on 3 simple examples.

3.3.1. Simulation 1: sinc(x) in 1D

In this simulation, the target function is $f = sin(2\pi x)/2\pi x$ for $-1 \le x \le 1$. 40 data points, equally spaced over the domain, are used as a training set. The network has 20 Gaussians, initially spaced uniformly over the input domain. The height, width, bias, and learning rate parameters were set over a range of initial conditions with typical values of 0.0, 0.15, 0.0, and 0.2, respectively. The simulation was run for 400 trials, which corresponds to an average of 10 presentations of the 40 patterns.

The table below compares the "Bricks" model to the standard backpropagation algorithm using Gaussians (i.e. each Gaussian has its own width and height which are updated individually).

Model	*Max Err (0 Faults)*	*Max Amp*	*Max Err (1 Fault)*
Standard	*0.011*	*0.33*	*0.67*
Bricks	*0.023*	*0.16*	*0.18*

The column "Max Err (0 Faults)" shows the largest error to the approximation of the target function by the fault-free network. "Max Amp" is the maximum Gaussian amplitude assumed by one of the

hidden units. "Max Err (1 Fault)" is the sum of the maximum error of the fault-free network plus the largest amplitude error that could occur given a worst case fault in the network. For the "Standard" model we assume, this to be twice the maximum amplitude since, with suitable normalization, the worst error is a change in the sign of this weight. Of course, this error could be much worse if the output signals of the hidden units are using only a fraction of the physically possible signal amplitude range, or if conductance weights are used which, due to a short, could assume a large multiple of their usual value.

For the "Bricks" model, the maximum signal fault is the same as the standard amplitude, as was illustrated in Figure 2 and discussed in Section 3.1.

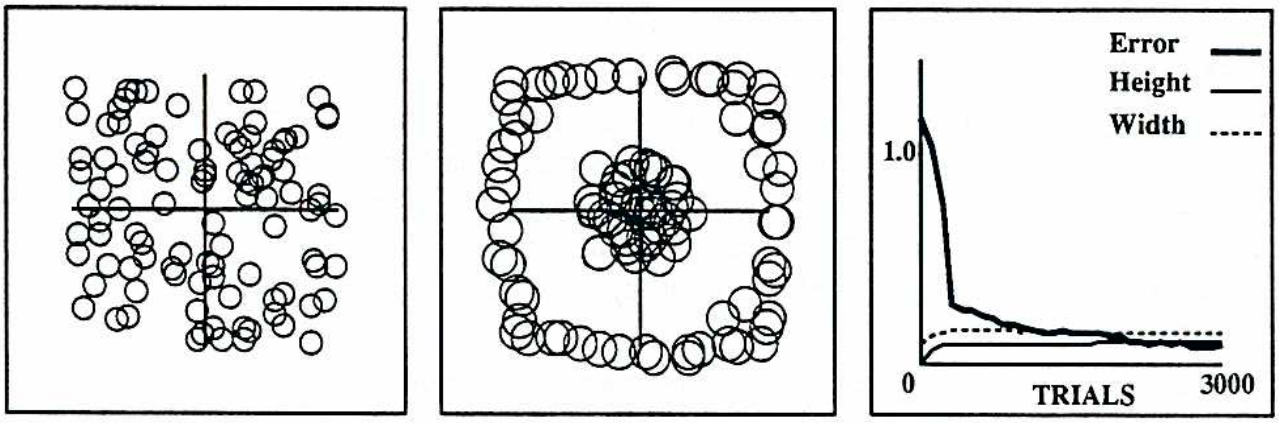
3.3.2. Simulation 2: sinc(r) in 2D

In this simulation, the target function is $f = \sin(2\pi r)/2\pi r$, where $r=\sqrt{x^2+y^2}$ for $-1 \le x,y \le 1$. 400 data points, equally spaced over the domain, are used as a training set. The network has 100 Gaussians, randomly placed in the input domain. The height, width, bias, and learning rate parameters were set over a range of initial conditions with typical values of 0.0, 0.1, 0.0, and 0.4, respectively. The simulation was run for 3000 trials, which corresponds to an average of about 7.5 presentations of the 400 patterns.

Like the table above, the following table shows the comparison of the "Bricks" model to the standard backpropagation algorithm.

Model	*Max Err (0 Faults)*	*Max Amp*	*Max Err (1 Fault)*
Standard	*0.039*	*0.35*	*0.74*
Bricks	*0.084*	*0.10*	*0.18*

Figure 3 shows graphically how the simulation progresses. The initial state of the locations and widths of the Gaussians is shown in Figure 3a, while Figure 3b shows the final state after the simulation. Figure 3c shows how the width, the height, and the maximum error change over the course of the simulation.



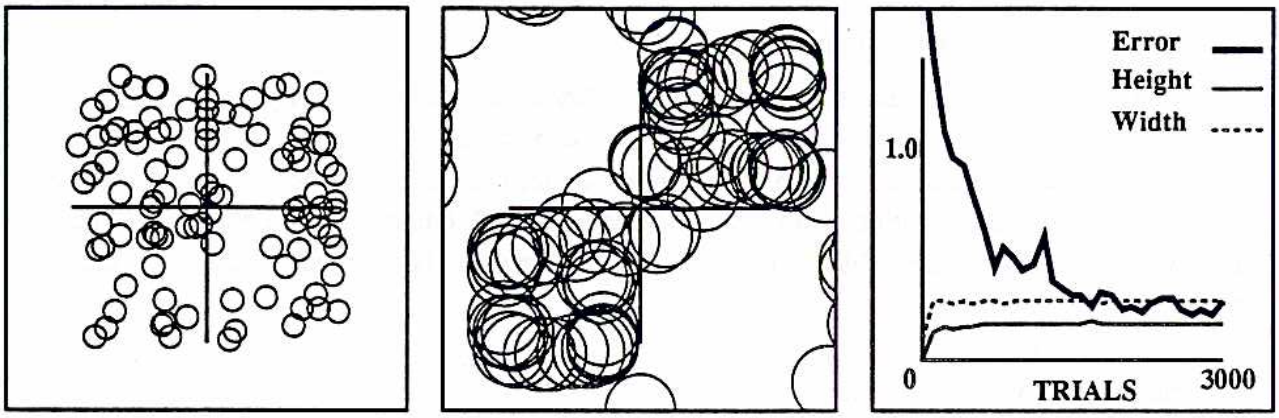
(a) *(b)* *(c)*

Figure 3: *(a) Initial Gaussian centers and widths (circles with 2σ radius), (b) final state, (c) plot of key parameters over time.*

3.3.3. Simulation 3: sign(x*y) in 2D

In this simulation, the target function is the analog version of the exclusive-or problem where $f = sign(xy)$ for $-1 \le x,y \le 1$. The parameters of the simulation are the same as for the 2D sinc function above. Figure 4 shows graphically how the simulation progresses, and the following table summarizes the results in the format of the previous tables.

Model	*Max Err (0 Faults)*	*Max Amp*	*Max Err (1 Fault)*
Standard	*0.07*	*0.75*	*1.57*
Bricks	*0.19*	*0.18*	*0.37*



(a) *(b)* *(c)*

Figure 4: *(a) Initial Gaussian centers and widths (circles with 2σ radius), (b) final state, (c) plot of key parameters over time.*

3.4. Modifications of "Bricks Model"

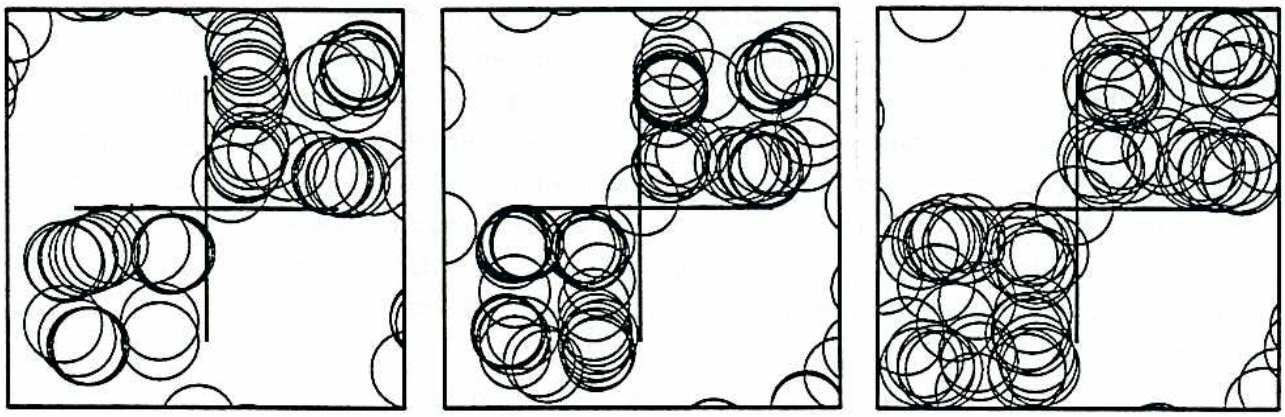
In the following subsections, we examine some modifications to our "Bricks Model" algorithm to see what impact they have on improving the speed or quality of convergence.

3.4.1. Adding random noise to center displacements

One problem with backpropagation is that hidden units can "clump" together. This occurs whenever the parameters of two hidden units coincide by chance. Since they both then "see" the same errors, both of their parameters are changed identically for the remainder of the simulation. This is more of a problem with the "Bricks Model" because, by definition, the height and width parameters are the same for all the hidden units. Thus if the locations of any two Gaussians happen to overlap, they will never separate.

To avoid this problem, we propose adding random noise to the adjustment of the centers of the Gaussians so that the coinciding centers will separate. The effect of noise is illustrated in Figure 5. Figure 5a shows the final state of the Gaussians in a training run without noise as described in Section 3.3.3. Although one cannot identify aligned Gaussians, the seemingly fewer number of circles in this figure as compared to the other two indicates that a significant number of them lie in the same place. Figure 5b shows the results of randomly modifying adjustment steps of the centers by ±20%. Although the overlapping is no longer exact, there is still a significant amount of clumping. Figure 5c shows the results of modifying the center adjustments of the Gaussians by a factor varying from 0.5 to 2.0: this

distribution is much more uniform. The convergence rates in these three cases are about the same, but the overall sum-squared error is better with the added randomness.

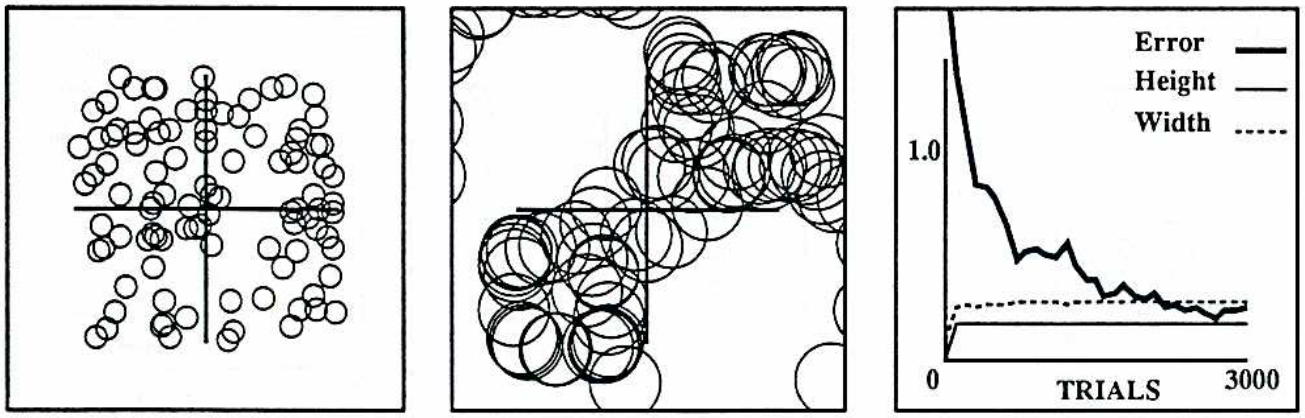


(a) (b) (c)

Figure 5: *(a) Final state with no randomness to center adjustments, (b) ±20% randomness, (c) 50% to 200% randomness.*

3.4.2. Height adjustment as a percentage of maximum height

In another variation to the "Bricks Model", the height of the Gaussians is set to some fixed percentage of the maximum height of the target function since the primary goal is to control the maximum fractional error that the failure of any one unit can cause. This can be done by taking the difference of the maximum and minimum target values observed so far and then multiplying by the desired percentage. Thus a user can specify the percentage of fault tolerance of a network to worst case single errors. An example of this is shown in Figure 6. In this case the percentage error was set to 8.6% to correspond to the resulting fraction observed in Section 3.3.3. This value was chosen such that the final height of the Gaussians in this experiment was the same as the final height of the Gaussians in Figure 4. In this case, since the height parameter reaches its final value very early in both simulations, the results are very much the same.



(a) (b) (c)

Figure 6: *(a) Initial Gaussian centers and widths (circles with 2σ radius), (b) final state, (c) plot of key parameters over time.*

3.4.3. Increasing the Attraction Width of the Gaussians

A problem that occurs with the use of Gaussians in the "Bricks Model" is that the magnitude of the change in the positions of the Gaussians depends on the height of the Gaussian at the current pattern input point. Thus if the data point falls at a point that is two or three sigmas from the (fixed width) Gaussian, the function value of the latter at that point is very small and the change in its center position is negligible. This is illustrated in Figure 7 where one can see that many of the units do not participate in the approximation to the curve since they were too far away to be "pulled in". However, if the "attraction width" is increased by a factor of two, as shown in Figure 8, more of the units are "pulled in". This results in faster convergence to a smaller error than the case in Figure 7. Also, since there are more units, their heights are smaller, and this makes the network illustrated in Figure 8 more tolerant to failures of the hidden units.

Increasing the "attraction width" substantially beyond two times the actual "hump width" does not provide further improvement. On the contrary, some of the units are now pulled towards the centroid of a whole curve segment that is in the influence region of the increased "attraction width"; this tends to pull the units of the curve that is to be approximated to areas where the curve has high curvature.

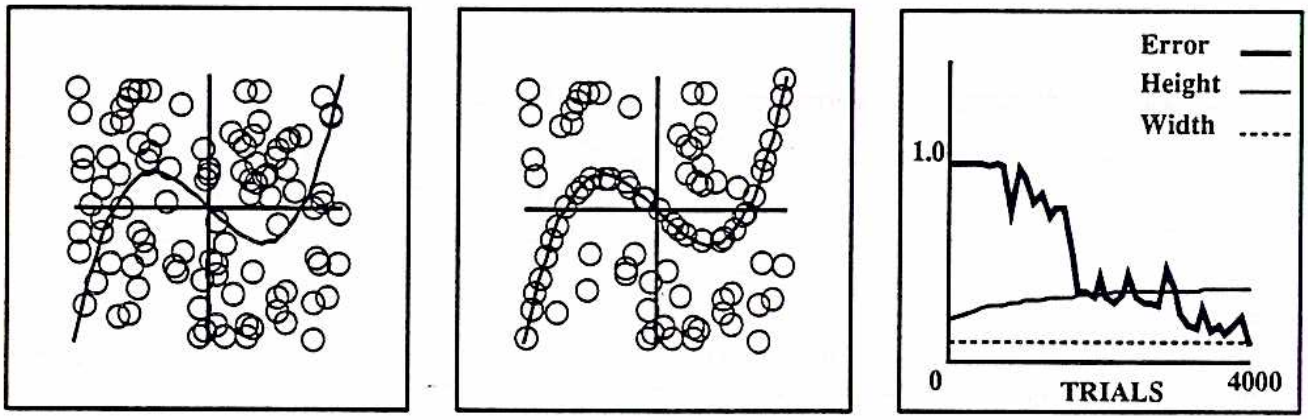


Figure 7: *(a) Initial Gaussian centers and widths, (b) standard algorithm, (c) plot of key parameters over time.*

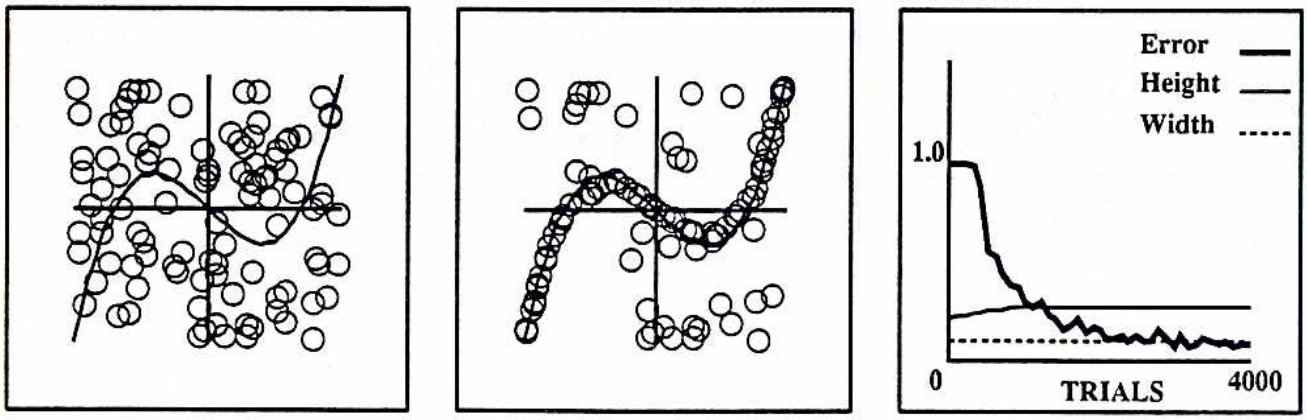


Figure 8: *(a) Initial Gaussian centers and widths, (b) algorithm with 2x effective radius, (c) plot of key parameters over time.*

4. CONCLUSION

In summary, the "Bricks" model shows certain advantages over traditional ways of approximating an analog output function with an artificial neural network. The decomposition of the output signal into many small, equally-sized contributions makes it possible to a achieve a higher degree of fault tolerance, because the worst case error amplitude that any failure of a hidden unit can produce can be limited to some known fixed value. In principle, this value can be made arbitrarily small by increasing the number of hidden units and correspondingly lowering their individual contributions to the output signal. However, this improved fault tolerance must be weighed against the costs of providing more hidden units and the disadvantage of longer training times.

If small output errors on the order of a few percent of the output signal are desired, it will be advantageous to resort to a hierarchical scheme with multiple redundant networks and a supervisory net that disconnects nets producing output signals that deviate too much from the average signal of all the nets [2,5]. However, the scheme presented here may have its niche for limiting output errors to about 10%; it requires only a small change to the implementation and control algorithm of traditional layered feed-forward neural network architectures.

5. Acknowledgements

This work was supported by the Air Force Office of Scientific Research (AFSC/JSEP) under Contract Number F49620-90-C-0029 and NEC.

References

[1] C. H. Séquin and R. D. Clay, "Fault Tolerance in Artificial Neural Networks", Proc. Int. Joint Conf. on Neural Networks, pp. I-703-708, San Diego, CA, June 1990.

[2] C. H. Séquin and R. D. Clay, "Fault Tolerance in Feed-forward Artificial Neural Networks", to appear in "Neural Networks, Concepts, Applications and Implementations", Vol. 4, Antognetti and Milutinovic, eds., Prentice Hall, 1991. Available as Tech Report #90-031 from International Computer Science Institute, Berkeley.

[3] C. Neti, M. H. Schneider, and E. D. Young, "Maximally Fault-Tolerant Neural Networks and Nonlinear Programming", Proc. Int. Joint Conf. on Neural Networks, pp. II-483-496, San Diego, CA, June 1990.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," Parallel Distributed Processing: Explorations in the Microstructure of Cognition, eds. D. E. Rumelhart, J. L. McClelland, and PDP Research Group, Bradford Books, Cambridge, MA, 1986.

[5] W. P. Lincoln and J. Skrzypek, "Synergy of Clustering Multiple Back Propagation Networks," in *Neural Information Processing Systems 2*, ed. D.S. Touretzky, pp. 650-657, Morgan Kaufmann Publishers, San Mateo, 1990.