Complete and Partial Fault Tolerance of Feedforward Neural Nets

Dhananjay S. Phatak and Israel Koren, *Fellow, IEEE*

Abstract—**A method is proposed to estimate the fault tolerance of feedforward artificial neural nets (ANN's) and synthesize robust nets. The fault model abstracts a variety of failure modes of hardware implementations to permanent stuck-at type faults of *single* components. A procedure is developed to build fault tolerant ANN's by replicating the hidden units. It exploits the *intrinsic* weighted summation operation performed by the processing units to overcome faults. It is simple, robust, and applicable to any feedforward net. Based on this procedure, metrics are devised to quantify the fault tolerance as a function of redundancy.**

Furthermore, a lower bound on the redundancy required to tolerate all possible single faults is analytically derived. This bound demonstrates that less than triple modular redundancy (TMR) cannot provide complete fault tolerance for all possible single faults. This general result establishes a necessary condition that holds for *all* feedforward nets, regardless of the network topology or the task it is trained on. Analytical as well as extensive simulation results indicate that the actual redundancy needed to synthesize a completely fault tolerant net is specific to the problem at hand and is usually much higher than that dictated by the general lower bound. The data implies that the conventional TMR scheme of triplication and majority vote is the best way to achieve complete fault tolerance in most ANN's.

Although the redundancy needed for complete fault tolerance is substantial, the results do show that ANN's exhibit good partial fault tolerance to begin with (i.e., without any extra redundancy) and degrade gracefully. The first replication is seen to yield maximum enhancement in partial fault tolerance compared with later successive replications. For large nets, exhaustive testing of all possible single faults is prohibitive. Hence the strategy of randomly testing a small fraction of the total number of links is adopted. It yields partial fault tolerance estimates that are very close to those obtained by exhaustive testing. Moreover, when the fraction of links tested is held fixed, the accuracy of the estimate generated by random testing is seen to improve as the net size grows.

I. Introduction

NEURAL computing is rapidly evolving as a viable solution to several problems. For use in applications requiring high reliability, artificial neural nets (ANN's) have to be fault tolerant. The ANN should possess a high degree of fault tolerance to begin with and its performance should degrade gracefully with increasing number of faults. Fault tolerance of ANN's is often taken for granted or treated as a subsidiary issue [1], [2]. Investigations by Carter *et al.* [1], [3] indicated that ANN's are not always fault tolerant and demonstrated the need to quantitatively evaluate their robustness. Several expositions have addressed various aspects of fault tolerance of ANN's [3]–[14].

Simulation results on the XOR problem are reported in [4], but these are quite specific to this one example and the underlying fault model. The emphasis of [5] is on recovery through relearning. The issue of on-line or operational fault tolerance is not considered there. Fault tolerance of Hopfield type ANN's for optimization problems was investigated in [6]. It does not, however, address fault tolerance in trainable ANN's. Belfore *et al.* have developed an analytical technique for estimating the performance of ANN's in presence of faults [7], [8]. They construct a Markov model for the ANN by drawing analogy with magnetic spin systems using statistical mechanics. Emmerson *et al.* studied fault tolerance and redundancy of neural nets for the classification of acoustic data [9]. They found that a single layer perceptron (direct I/O connections without any hidden units) was far less damage resistant than a multilayer version, which had a single hidden layer consisting of five to 25 hidden units. However, increasing the redundancy by increasing the number of hidden units did not yield significant improvement in the performance under faulty conditions. They then performed a singular value decomposition (SVD) analysis of the weight matrix and found that there were no linear dependencies in the hidden units, i.e., none of the units was redundant as per the SVD analysis. They concluded that back propagation training discovers internal representations that appear to be distributed but may not be suited for fault tolerance. Sequin *et al.* [10] set a predetermined number of randomly chosen hidden units stuck at faulty output values during training. The set of units that are assumed to be faulty varies from epoch to epoch during the training. This type of training yields a more fault tolerant net as expected, but it needs longer learning times, may not converge unless the number of faulty units is small, and does not necessarily lead to a better utilization of available resources. Neti *et al.* recently reported numerical results on the synthesis of fault tolerant nets [11]. They add constraints to ensure fault tolerance and look for a solution to the constrained optimization problem. This approach is analytically well founded, but it is also very computation intensive, and more experiments are needed to draw any general conclusions about the resulting fault tolerance. Training procedures that attempt to increase fault tolerance also appear to enhance generalization capability [11], [12]. Segee and Carter [13] have developed a measure

Manuscript received August 1, 1992; revised April 4 and October 11, 1993. This work was supported in part by the National Science Foundation, Grant MIP 90-13013.

D. S. Phatak was with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, he is now with the Department of Electrical Engineering, State University of New York, Binghamton, NY 13902-6000 USA.

I. Koren is with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003 USA.

IEEE Log Number 9214290.

of graceful degradation for continuous valued ANN's. They performed an extensive study of the fault tolerance capabilities of multilayer perceptrons and Gaussian radial basis function networks by fault injection experiments and found that setting a few units stuck at zero during training yielded a substantial enhancement in fault tolerance of radial basis nets but only a marginal improvement in the fault tolerance of multilayer perceptrons. However, they did not try to evaluate how much redundancy is necessary to achieve a given level of fault tolerance or how to make a given net fault tolerant.

The issue of redundancy required to achieve fault tolerance has not yet been adequately addressed. Also, the fault models usually adopted are very restrictive. Most researchers consider only a disconnection of links or units which usually amounts to stuck-at zero type permanent faults. While this may be sufficient to model a burnt out link or open circuit type fault, it may not appropriately model a short to power supply, which can be of dual polarity in an analog implementation. We extend the fault model to allow permanent stuck-at $\pm W$ type faults on a single component (weight/bias). This considerably enlarges the number of actual faults that can be abstracted by the stuck-at model. We have proposed a scheme of replication of hidden units to render a feedforward net fault tolerant [14]. It exploits the intrinsic weighted summation operation performed by the units to overcome faults. Based on this scheme, metrics were devised to measure the fault tolerance of ANN's as a function of redundancy [14]. Our results indicated that a significant amount of redundancy is needed to achieve complete fault tolerance, despite the somewhat restrictive assumption of single faults. Based on analytical results for a few canonical problems and extensive simulations on several benchmarks, we conjectured a lower bound on the amount of redundancy needed to achieve complete fault tolerance in *any* net [14]. In this paper, we analytically prove a modified form of that conjecture and obtain a lower bound on the redundancy that holds for any net. It demonstrates that less than triple modular redundancy (TMR) cannot provide complete tolerance for all possible single faults, if any of the fan-in links feeding any output unit is essential (a link is defined to be essential if the disconnection of that link, or equivalently, a stuck-at zero fault on the associated weight, causes the net to malfunction). Furthermore, analytical as well as simulation results indicate that the actual amount of redundancy needed to synthesize a completely fault tolerant net is usually much higher than that dictated by the general lower bound. The implication is that the conventional strategy of triplication and majority vote is a better way of achieving complete fault tolerance for most ANN's. Our approach also reveals a simple way to improve the fault tolerance of any net to a desired level by adding redundancy.

The next section describes the topology and fault model. Section III presents a general procedure to build a fault tolerant net by replication of hidden units and compares and contrasts it with the conventional TMR/n-MR schemes. Based on this procedure, a lower bound on the redundancy needed to achieve complete fault tolerance is derived in Section IV. Analytical and simulation results obtained by applying the replication scheme to several canonical and benchmark problems are then summarized. The results show that the redundancy required for complete fault tolerance is usually very high. It is extremely hard if not impossible to realize the above lower bound on redundancy in practice. If less redundancy is provided, only partial fault tolerance is feasible. Section V considers this issue of partial fault tolerance. A simple but accurate testing strategy is proposed for large nets, where exhaustive testing is not feasible. Conclusions and future extensions are presented in the last section.

II. Topology and Fault Model

A. Topology

We consider feedforward networks of sigmoidal units trained with supervised learning algorithms. The inputs and weights (includes biases unless mentioned otherwise) are real valued and can take any value in $(-\infty, +\infty)$. The output of the ith unit is given by

$$o_i = f(\textit{resultant\_input}_i)$$

where

$$f(x) = \frac{a}{1 + e^{-x}} - b$$

and

$$\textit{resultant\_input}_i = \sum_{j=1}^{N_i} w_{ij} o_j - \textit{bias}_i. \tag{1}$$

Here N_i is the total number of units that feed the ith unit, w_{ij} is the weight of the link from unit j to unit i, o_j is the output of the jth unit, and a, b are constants. In the analysis and simulations, we have used $(a = 2, b = 1)$ giving symmetric output in $(-1, 1)$; $(a = 1, b = 0.5)$ giving symmetric output in $(-0.5, +0.5)$; and $(a = 1, b = 0)$ with asymmetric output in (0, 1). We concentrate only on classification tasks. Here, N distinct output classes can be encoded into $\lceil \log_2 N \rceil$ (or more if desired) bits. The output units produce the bit pattern which encodes the exemplar's category. Thus the output values can be separated into two distinct logical classes *viz.* "0" and "1." This does not mean that real valued units are being (mis)used just to implement a switching function, because the inputs and weights of the ANN are real valued. Typically, training stops when the outputs are within 5% of the target. Besides the back propagation learning algorithm and its variants [16]–[19], we have also considered the cascade correlation learning algorithm [20]. These two algorithms often lead to very diverse architectures.

B. Fault Model

The fault tolerance properties of ANN's can be broadly classified as 1) on-line or operational fault tolerance and 2) fault-tolerance through relearning. The first refers to the ability to function in the presence of faults without any relearning or any correcting action whatsoever. This is possible only if the net has built-in redundancy. The second is the ability to recover from faults by reallocating the tasks performed by the faulty elements to the good ones. On-line fault tolerance is attractive

because a net with this property can function correctly even in the presence of faults, as long as the fault tolerance capacity is not exceeded. No diagnostics, relearning, or reconfiguration is needed. Thus fault detection and location can be totally avoided. In this paper, we consider only the on-line fault tolerance and evaluate the redundancy needed to achieve it.

Hardware can fail in a number of complex ways. This is especially true of ANN's, since these can be implemented in many radically different ways such as analog VLSI or as virtual nets simulated on a multiprocessor network, etc. It is extremely difficult to account for all types of physical faults. However, permanent physical faults must manifest themselves via their effect on the network parameters. We therefore model faults at a more abstract level: as changes in the weight/bias values. Weights are set stuck at zero and $\pm W$, where W =maximum of {maximum magnitude among all the parameter values developed during learning; value needed to drive a unit into saturation}.

These stuck-at faults on weights and biases model a wide range of physical faults. For instance, in a digital implementation, a stuck-at fault affecting the sign bit of a weight might cause its value to change from $+W$ to $-W$ or vice versa and is covered by this fault model. In an analog VLSI implementation, on the other hand, an open circuit leading to a disconnection of a link could be modeled by setting its weight stuck at zero. If a link got shorted to power supply (which could be of a dual polarity $\pm V_{dd}$), the fault could be abstracted by setting the associated weight stuck at $\pm W$. If the weights are implemented as resistors, their values may degrade over time due to aging. A stuck-at $\pm W$ fault on the weight can appropriately model even the worst case of such a degradation due to aging. A change in the offset voltage of a transconductance amplifier will affect its output. This might be modeled as a change in the bias of the corresponding unit. Setting a bias to $+W$ and $-W$ results in a unit's output stuck at "high" or "low" and models the failure of a unit. Note that setting the weight associated with a link to zero corresponds to a disconnection of that link. Setting a bias to zero, on the other hand, has no physical significance. Bias faults are therefore limited to being stuck at $\pm W$. During testing, an output is classified as follows

$$\textit{output} = \begin{cases} 1 & \text{if } \textit{resultant\_input} = \textit{net\_input} - \textit{bias} > 0 \\ 0 & \text{if } \textit{resultant\_input} = \textit{net\_input} - \textit{bias} < 0 \end{cases}. \quad (2)$$

An output is considered wrong only if it switches the logical level. This happens if the *resultant_input* to an output-layer unit switches its sign [(1), (2)].

We conclude this section with a few more comments about the fault model. First, the model essentially investigates the fault tolerance at the algorithm level, independent of the underlying implementation or peculiarities of physical faults. It abstracts and simplifies physical failures into stuck-at faults affecting single components. It is certainly true that the physical faults are much more complex and lead to multiple component failures more often than not. Before modeling the complex physical faults in all their detail, however, it is more appropriate to treat them as equivalent to (arbitrarily) large changes in single parameter values and test whether the model/algorithm can still yield correct results. Such an abstraction to stuck-at faults has been very widely used in testing of digital circuits and has proved to be sufficient to model majority of physical faults. We have extended it to cover continuous valued parameters so that it can model many physical faults of interest.

Second, it turns out that even with this simplified fault model, ANN's need large redundancy (TMR or higher) to tolerate all possible single faults (this is shown in the following sections). If the simplest of faults require no less than TMR to achieve complete tolerance, multiple and complex faults must require much higher redundancy. The unwieldy and cumbersome general analysis of multiple faults is therefore unnecessary. Finally, we would like to point out that this model is more general than the fault models usually adopted in the available literature. Most researchers consider only stuck-at zero faults which are inadequate to cover many faults of interest as shown above.

III. A General Procedure to Build a Fault Tolerant Net

A simple way to achieve fault tolerance is through redundancy. Triple modular redundancy (TMR) and n-MR schemes with majority voters have been widely used for reliability enhancement in digital systems. Extension of these schemes to ANN's is the most obvious way to render them fault tolerant. In a TMR scheme for ANN's, the whole net would be triplicated and a majority voter would be added for every triplet of output units. This way the number of majority voters equals the number of output units in the original net and could be large. Although a gate that performs majority voting on analog signals is not that complex, it is not as simple as a digital majority voter. A majority gate, in effect, evaluates the *median* of all the input values. It is well known from signal processing literature that the median value is far more robust than the mean when the inputs are noisy or faulty. Thus TMR is a very robust and well proven technique, applicable to any net.

ANN's, however, are quite different from digital systems. Typically, they are composed of a densely interconnected network of processing units or nodes. The units themselves may not be very complex, but a large number of them independently operate in parallel at any given time. The gross similarity between ANN's and biological systems along with the highly parallel mode of operation suggests that ANN's could be *inherently* fault tolerant. Any such *intrinsic* fault tolerance in ANN's must be quantified vis-a-vis the redundancy required to achieve it. This is one of our main goals. To this end, we have developed a scheme of replicating the hidden units to achieve fault tolerance. It exploits the intrinsic weighted summation operation performed by the units to overcome faults. It is quite simple, is applicable to any net, and can be used as a yardstick to measure the redundancy needed to achieve fault tolerance.

From (1) and (2) it is easy to verify that the output changes its logical level only if the *resultant_input* changes its sign.

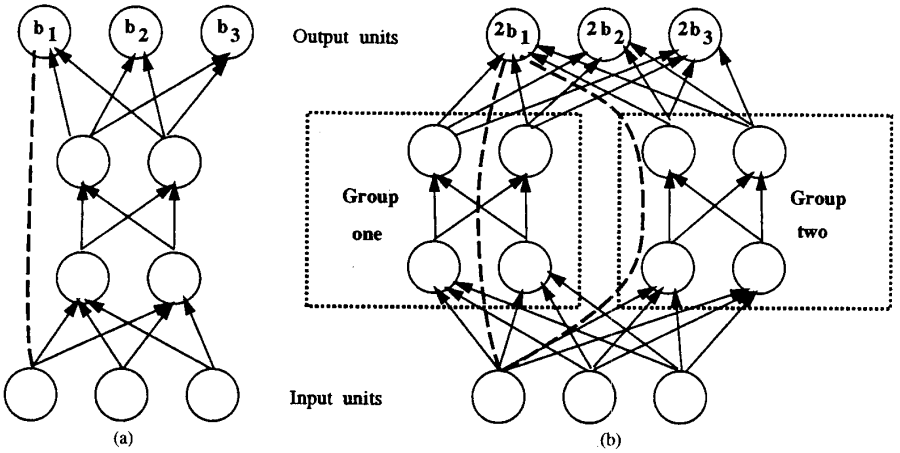

Fig. 1. Replication of hidden units. (a) Original net. (b) One extra group of hidden units. Biases of output units (shown inside the circles representing the output units) are also scaled by 2.0.

Thus ensuring that the *resultant_input* (to every output unit) is of the correct sign for every training sample ensures that the output is correct as well. The output retains its logical value, even if the *resultant_input* is multiplied by any positive number. Thus, if all the hidden units of a net are replicated g times, and the biases of the output units are scaled by g, the resulting net yields the same classification (logical) outputs as the original net. This process of replication is illustrated in Fig. 1. Each replication is referred to as a "group" or a "module." Here, the number of input and output units is unchanged. Only the hidden units and links that directly feed the output units are replicated and the biases of the output units are scaled accordingly. The single fault assumption implies that the contribution (to *resultant_input* of an output unit) of at most one group can be incorrect. Each correctly working group contributes an input of the right polarity (sign). Hence, a sufficient number of these correcting influences can overcome the erroneous contribution of one faulty group by restoring the correct polarity of *resultant_input*. This can be used to build a net that tolerates all single failures as summarized by the following steps:

Step 1) Start with a net that learns the given input/output pattern mapping. It is preferable to have a minimal or near minimal net.

Step 2) Find the number of groups needed to correct each fault as follows: For each component stuck-at a faulty weight value, calculate the *resultant_input* for each output unit that malfunctions. Call it I_f. If the desired *resultant_input* is I_d and each correctly working group contributes I_s, then the number of additional groups needed to restore the polarity of *resultant_input* and the total number of groups are, respectively

$$\left|\frac{(I_d - I_f)}{I_s}\right| \quad \text{and} \quad \left|\frac{(I_d - I_f)}{I_s}\right| + 1. \tag{3}$$

Step 3) The maximum among all the values found in Step 2 is the required number of replications.

Step 4) If the number of groups needed is g, scale (multiply) the bias of each output unit by g.

The number of replications needed depends on the initial "seed" group which should be minimal or near minimal.

Even though this method appears no different from the conventional TMR or n-MR schemes, it is distinct in many ways. In particular, the input and output units are *not* replicated as mentioned above. Only the hidden units and all the connections feeding the output units are replicated and the biases of the output units are scaled accordingly. Moreover, there is no majority voter to explicitly mask out the faults. Rather, the fault tolerance is achieved through the weighted summation process itself, which is an *intrinsic* characteristic of the ANN model. Fault tolerance achieved this way is therefore the same as intrinsic fault tolerance arising due to the high connectivity and other attributes of ANN's. It is this feature which is the most important from our perspective.

This brute force method appears to be very expensive in terms of the number of units and links needed. However, it has some distinct advantages. Izui *et al.* [21] have shown that the rate of convergence during learning as well as the solution accuracy improves with the number of replications. Similar observations have been made in [22] where it is inferred that clustering of multiple nets improves not only the fault tolerance but also the performance. This method can be extended for nets with continuous valued outputs by scaling the *resultant_input* to each output unit by the factor $1/g$. In fact such a scaling amounts to the evaluation of algebraic mean value of the contributions of each of the g groups. Unfortunately, the required redundancy turns out to be too large to be practical. This is not too surprising: it is well known that the mean value is far less efficient at suppressing/filtering out faults than other measures such as the median. A single bad sample can significantly corrupt the mean, but the median can remain unaffected. Hence, median filters are more widely used instead of computing the mean values in applications such as image processing.

In summary, the above procedure gives a simple way of evaluating fault tolerance as a function of the number of replications, i.e., redundancy (this is further elaborated in the section on partial fault tolerance). It therefore serves

to measure the fault tolerance of ANN's as a function of redundancy. This is the main motivation behind the replication scheme.

IV. A Lower Bound on Redundancy

This section establishes a lower bound on the redundancy necessary to tolerate all possible single faults. It is based on the replication procedure described above. It holds for *all* feedforward nets, regardless of the topology or the specific task at hand.

In the following, a link is defined to be essential if the disconnection of that link (i.e., a stuck-at zero fault on the associated weight) causes the net to malfunction. A malfunction refers to a classification error which means that the output unit at the receiving end of the disconnected link produces output of wrong logical value for at least one I/O pattern.

Theorem 1: In a feedforward net consisting of sigmoidal units, if any of the links feeding any output unit is essential, or in other words, if a stuck-at zero fault on any of the links feeding any output unit causes the unit to produce a classification error, then at least two extra replications, or equivalently, three or more groups of hidden units, are required to achieve complete fault tolerance for all single faults.

The proof is included in the Appendix.

This result suggests that the conventional TMR scheme is as good or better than the replication of hidden units. Note that a conventional TMR scheme would need to replicate the output units as well, besides adding the majority voters. Even if the number of output units (and hence the number of voters to be added) is comparable to or higher than the number of hidden units, the TMR scheme is not that expensive in terms of area overhead (as compared with the replication of hidden units) because the units and the voters tend to take a small area. In a neural net it is the interconnections in between the units that take the most area, and the interconnections have to be replicated in both the schemes. In a majority of nets though, the number of outputs is much smaller than the number of hidden units. Furthermore, the lower bound established by the above theorem is not *attainable* in most cases. Hence, a conventional TMR scheme is as good or better for all feedforward ANN's.

Theorem 1 establishes a *necessary* condition on the amount of redundancy needed, when at least one output link is essential. Note that it does not include *sufficiency* conditions, i.e., this lower bound may not be *attainable*. In other words, three groups are necessary but might not be sufficient to tolerate all single failures. The number of groups that are sufficient to achieve complete fault tolerance depends on the specific problem at hand, the topology of the net, and the type of units employed. Also, Theorem 1 says nothing about the case when none of the links feeding the output units is essential. In such a case (when none of the output unit links is essential) fewer groups might render the net completely fault tolerant. If any of the output unit fan-in links is essential, however, the above theorem holds. It shows that large redundancy is needed even if only single failures are considered. Finally, we would like to point out that Theorem 1 is not restricted only to sigmoidal activation functions. It can be extended to incorporate any activation function as long as the function is monotonic.

In fact, the *attainable* or *feasible* lower bound is often higher than the above. This is corroborated by a substantial number of analytical and simulation results. We have applied the above process of replications to several problems. The minimum redundancy sufficient to synthesize completely fault tolerant nets has been analytically derived for some canonical problems [14], [23], [24]. In [24] it is proved that a single hidden unit can solve only the 2×2 encoding problem, while only two hidden units are sufficient to solve any $n \times n$ size problem. Thus the minimum sized net or the seed-net for the replication procedure is known a priori. This makes it possible to derive an attainable lower bound on the amount of redundancy needed for complete fault tolerance. For the encoding and XOR problems, it has been proved [14], [23] that the minimum number of groups of hidden units sufficient for tolerating all single faults is 4 if the sigmoid is asymmetric (i.e., the sigmoid output is in [0, 1]). The corresponding minimum number of groups is 3 if the sigmoid is symmetric (i.e., the sigmoid output is in $[-1, 1]$) [14], [23]. These results are stronger (than the general lower bound proved above) because they are true regardless of whether any of the output links is essential. These results clearly indicate that the attainable lower bound is dependent on the specific problem, as well as the type of units used and the topology of the net.

Extensive simulation data on realistic benchmark problems which are well suited for ANN's also corroborates the above fact, i.e., the actual redundancy required for synthesizing completely fault tolerant nets is much higher than that dictated by the general lower bound. We have run a large number of simulations on all the benchmarks from the CMU database [25] and some from other sources. For each of these problems, many nets were generated using the cascade correlation and/or back propagation algorithms. Simulations showed that the amount of redundancy required to achieve complete fault tolerance is usually extremely high (more than 6 replications). The results also reveal a nonuniform distribution of computational load. Few dominant weights are fault tolerance bottlenecks, while many others can be dispensed without significantly degrading the performance. This happens even if the fan-in is very large. In the Sonar benchmark [23], [25], [26], for example, the fan-in of the output unit was 63 in some nets. In the NETtalk benchmark [23], [25], [27], the fan-in of the output units was as high as 271. Then it might appear that each individual link would not be so critical. Despite the large fan-in, however, the nets still need a large number of replications (no less than five) to achieve complete fault tolerance. It appears that the theoretical lower bounds derived above are almost impossible to realize in practice. Merely providing a large number of hidden units is therefore insufficient. Similar observations were reported in [9] and [13]. This is not too surprising, since the training procedure is not geared toward fault tolerance. It must be modified to equitably distribute the computation among all units.

All this data clearly demonstrates that the conventional TMR scheme of triplication and majority vote is the best way to achieve fault tolerance for most practical problems. This is

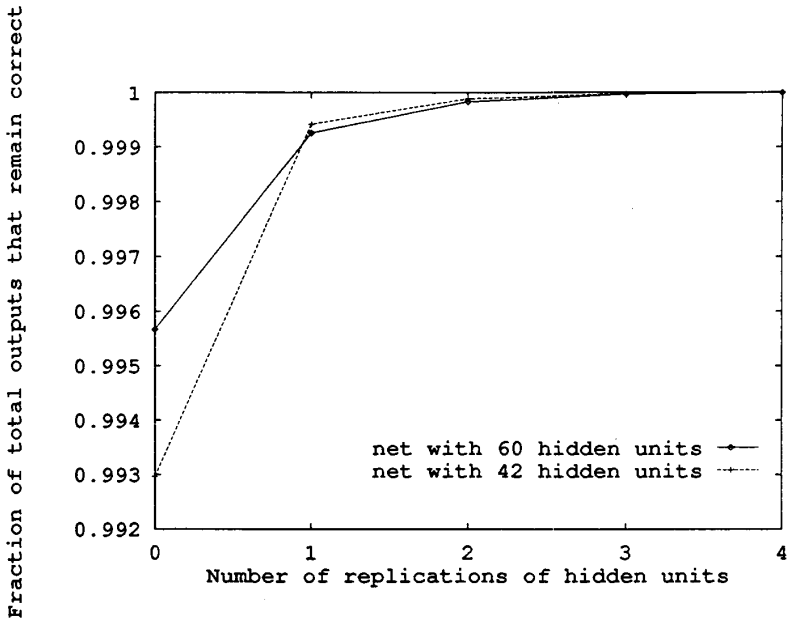

Fig. 2. Partial fault tolerance of nets trained on the sonar benchmark [25], [26] by back propagation. —◊—: Net with 60 hidden units. +···: Net with 42 hidden units.

consistent with the fact that a replication of modules is better than replicating individual elements, which is well established for digital systems. Neural nets are different, however, and it is a bit surprising that the conventional TMR strategy that works so well for digital systems also happens to be the best one for most ANN's. The following qualitative argument might provide more insight into why this happens. In Section III it was pointed out that the evaluation of the mean value, or performing a weighted algebraic sum (which is what a unit in the ANN does), is not an efficient way of masking faults. Instead, the extraction of the median value is a far more efficient way of masking faults. A scheme based on majority voting essentially performs such a median extraction and hence is more efficient.

V. Partial Fault Tolerance

The amount of redundancy needed for complete fault tolerance is prohibitive as shown above. If less redundancy is provided, fewer faults are tolerated. This directly leads to the notion of partial fault tolerance which is particularly important for ANN's. Unlike digital logic, ANN's are expected to perform better on larger, random problems. The specification of such tasks might be incomplete or might allow a few erroneous outputs. In some classification tasks, for instance, a "closest match" with one of the output classes is an acceptable criteria, rather than insisting on an exact match. In such cases, there can be a multitude of outputs that satisfy the closest match criteria. Thus an ANN's task might not be as rigidly defined as that of a digital system. Partial fault tolerance is therefore more pertinent for ANN's. Equally desirable is the ability to degrade gracefully.

A simple metric to quantify the partial fault tolerance is to count the number of faults tolerated as a function of the number of replications. We have applied this metric to all the canonical and benchmark problems mentioned above. For the sake of brevity, we illustrate the results for only two of the benchmarks (Sonar and NETTalk benchmarks) in Figs. 2 and 5, respectively.

A. Exhaustive Testing

The correct value of the fraction of all possible single faults that can be tolerated must be obtained through an exhaustive testing of all possible single faults, one at a time. Such a scheme is feasible only for small or moderately large nets. The Sonar nets happen to be manageably large. An exhaustive test strategy was therefore used to generate the plots in Fig. 2.

Details of the Sonar benchmark can be found in [25], [26]. The problem specification has 60 inputs and one output. For this problem, all the nets used to generate the plots shown in Fig. 2 were generated by the back propagation learning algorithm, had one layer of hidden units, and employed the asymmetric sigmoid. Nets were also generated by the cascade correlation, as mentioned above. This algorithm was used with both asymmetric and symmetric sigmoid activation functions. The partial fault tolerance data for these cascade correlation generated nets shows identical trends and is therefore excluded from the figure for the sake of clarity.

To generate the fraction values, each weight was in turn set at $+W$, 0, and $-W$, where W is the maximum magnitude. This corresponds to testing every possible fault for every weight, one at a time. Each bias was set at $\pm W$. Setting a bias to zero has no physical significance. In contrast, it is essential

to set a weight to zero to model the disconnection of a link. For *each* of the above settings (or faults), all the I/O patterns were tested. All the outputs that went wrong (switched logical levels from "0" to "1" or vice versa) were counted. This sum was then normalized by the following factor

(number of output units × number of I/O patterns ×total number of faults).

Note that

the total number of faults
= (the number of links present in the net
× the types of faults simulated per link
+ the number of biases present in the net
× the types of faults simulated per bias).

The resultant number represents the fraction of outputs that went wrong. Subtracting it from 1.0 yields the fraction of outputs that were correct, which is the partial fault tolerance metric mentioned above. This fraction is the y coordinate in the plots shown in Fig. 2. The partial fault tolerance was calculated for increasing number of replications until it reached a satisfactory level (usually 95% or higher), or up to 4 replications if the initial partial fault tolerance (without any extra replications) itself was higher than 95%.

The plots show that the partial fault tolerance is very good to begin with: more than 99% of all possible single faults are tolerated without any additional redundancy. Moreover, the first replication seems to yield the maximum enhancement in partial fault tolerance. Later successive replications yield lesser enhancements. It should be noted that the plots *appear* to reach the value 1.0 or complete fault tolerance at $x = 4$ in this figure. This is only an illusion, due to the limited resolution of the plotter. Numerical calculations showed that complete fault tolerance is *not* achieved even at x=6, i.e., 6 extra replications. It is clear that a TMR scheme with majority voter achieves complete fault tolerance with much less redundancy (the plot would reach the value $y = 1$ at $x = 2$, i.e., total number of modules= $2 + 1 = 3$).

B. Random Testing

For large nets such as those trained on the NETTalk benchmark (please refer to [25], [27] for further details about this benchmark), the exhaustive test strategy would take prohibitively large time. We therefore adopt a more efficient testing strategy described below. In our simulations [23], we used a net having 196 inputs, 26 outputs, and a list of 200 words, which generated 1114 I/O patterns. All the units had a symmetric sigmoidal activation function with outputs in $[-0.5, +0.5]$. The cascade correlation learning algorithm was used to train the nets. It typically uses about 75 units, and roughly 25000 independently adjustable parameters (weights and biases).

This number of weights and biases is too large to permit exhaustive testing. A more efficient testing strategy must be adopted. An analogy with testing of conventional digital circuits is in order here. Testing of digital logic is a full fledged area that has evolved over the past 25 years or so. It is well known that exhaustive testing of even the simplest systems, i.e., combinatorial logic circuits (which are, in some sense, equivalent to feedforward nets while sequential circuits are like ANN's with feedback connections) is prohibitive although the number of logic gates is relatively small. All kinds of elaborate techniques such as partitioning, testing with random input vectors, modular testing, built-in self-testing, etc., have to be employed to avoid exhaustive testing and still achieve acceptable fault coverage. The same situation arises in neural nets as well. Exhaustive testing becomes infeasible as the net size grows. Efficient strategies have to be devised for testing the hardware in large and complex ANN's.

In digital systems, testing with random input vectors proved to be quite effective and is widely used as a first step of many state-of-the-art testing algorithms to rapidly cover most faults. We extend this idea to the testing of ANN's. The bias faults are exhaustively tested. A fraction of the total number of links are then randomly chosen to be tested for weight faults. In large nets, the number of biases is usually a very small fraction of the total number of parameters. In the NETTalk benchmark, for example, out of the 25000 odd parameters, only 75 are biases. Moreover, a bias fault is equivalent to an *output* fault and thus models the failure of a unit. Hence it is more likely to cause malfunction than the disconnection of a single link. For these reasons, the bias faults should be tested exhaustively.

We applied this method to the Two Spirals Classification [20], [25], [28] and the Sonar and Vowel [25], [29] benchmarks and compared the results with the exhaustive testing scheme. The results are shown in Figs. 3 and 4. Fig. 3 shows the estimates of the fraction of faults tolerated for the Two Spirals benchmark. The net had approximately 250 independent parameters, including 20 biases. The x coordinate represents the fraction of links that were actually tested for weight faults. For each of the x values, several trials with different random seeds were run. Each trial used a different set of links for fault testing, due to the different random seed. The total number of links tested, however, was held fixed, corresponding to the fraction denoted by the x coordinate. *Averages* of the values generated in these trials are plotted as the y coordinates in the graph. It was found that 5 trials (per x coordinate value) were sufficient to yield a standard deviation less than 0.009 and a 95% confidence interval smaller than 2%. It is seen that the estimates are within 5% of the exhaustive-test generated value, even if only 1% of the links are actually tested for weight faults. The estimate improves slowly as more and more links are actually tested. Analogous results were obtained for the Vowel benchmark and are therefore omitted.

Fig. 4 shows a similar plot for the Sonar net. Here the number of independent parameters is significantly higher: nearly 3700, including 60 biases. In this case, the estimates are within 0.5% of the exhaustive test generated value when only 1% of the links are actually tested. Testing several other intermediate sized nets showed that the larger the number of parameters, the more accurate is the estimate of outputs that remain correct when the fraction of links actually tested for faults is held fixed. Thus, for larger nets, a fairly accurate

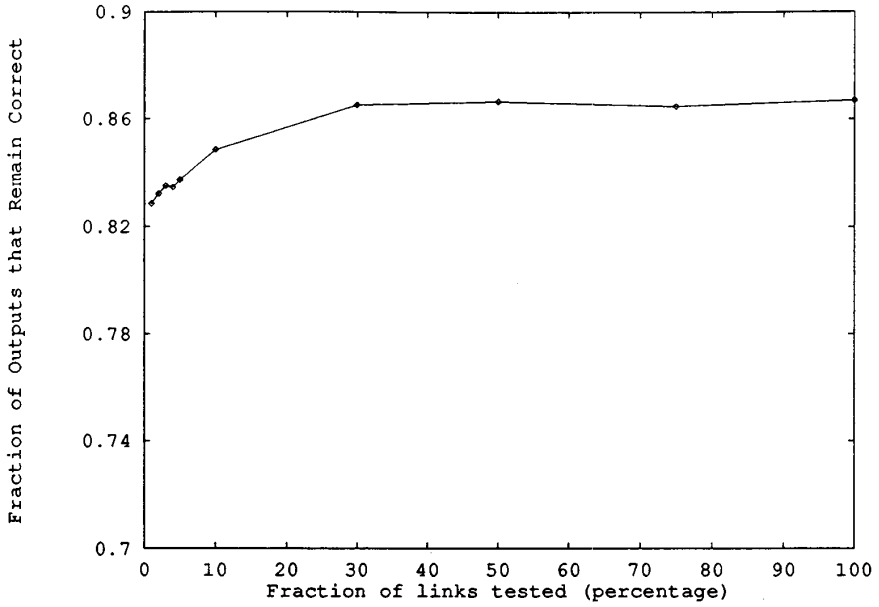

Fig. 3. Comparison of random link testing method with exhaustive testing for a typical net trained on the two spirals benchmark [20], [25], [28] by the cascade correlation algorithm. All biases are exhaustively tested. A fraction of the total number of links are then chosen at random for weight fault testing. This fraction (the x coordinate) is varied from 1% to 100% (exhaustive testing).

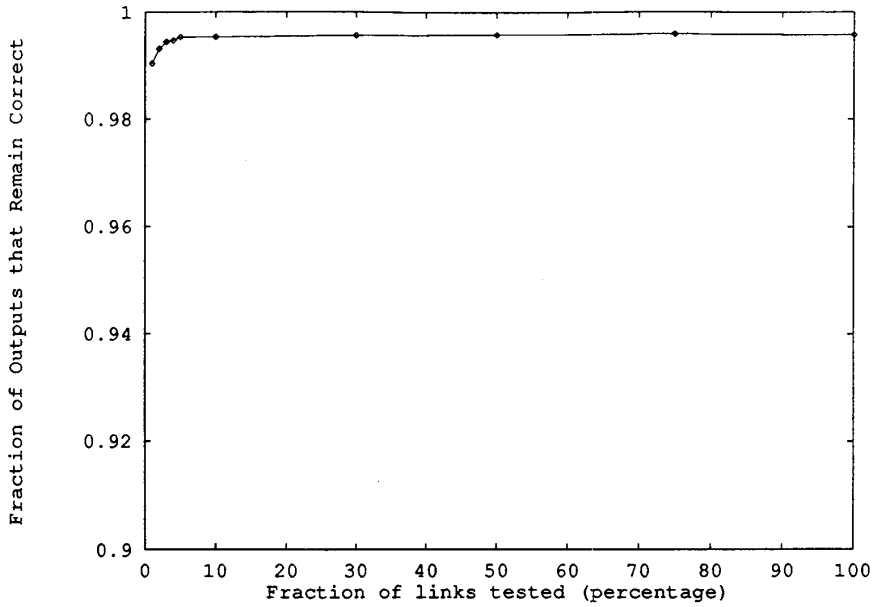

Fig. 4. Comparison of random link testing method with exhaustive testing for a typical net trained on the sonar benchmark by the back propagation algorithm. The point corresponding to 100% on the x axis is the result generated by exhaustive testing.

estimate can be obtained by testing a small fraction (1%) of links for weight faults.

For the NETTalk Benchmark, we therefore tested only 1% of the links. The resulting plot is shown in Fig. 5. Despite the big difference in scale (number of parameters, fan-in, training patterns, etc.), this plot exhibits features similar to the other benchmarks: the net possesses a very good degree of partial fault tolerance to begin with: about 99.4% of all possible single faults are tolerated without any additional redundancy. Also, the first replication is seen to yield the most performance improvement. However, a large number of replications are needed for complete fault tolerance, despite the large fan-in.

All the other nets trained on other benchmarks also showed identical trends. It is seen that the initial partial fault tolerance

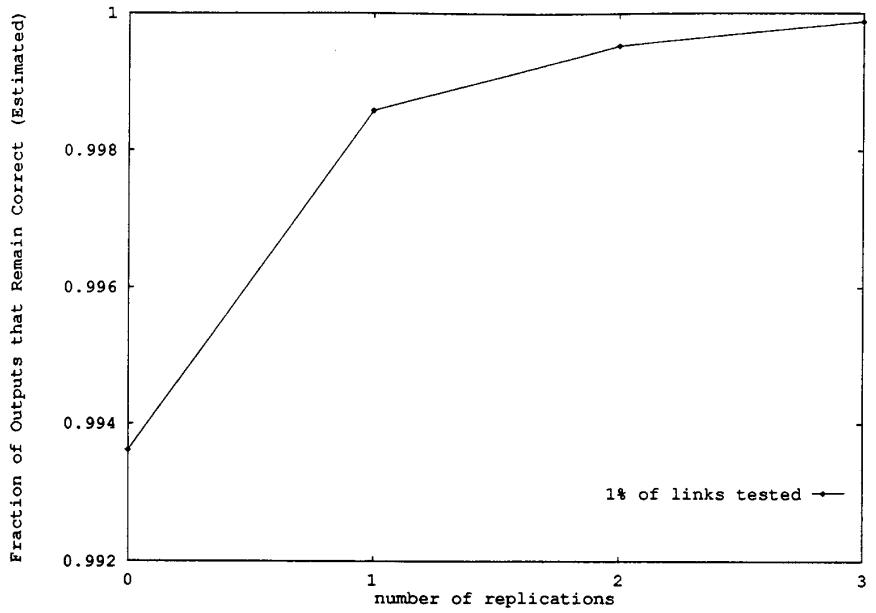

Fig. 5. Partial fault tolerance of a typical net with cascade correlation on the NETTalk benchmark [25], [27]. Training set consisted of 200 words which generated 1114 I/O patterns. The net had 24622 independent parameters (weights and biases) including 75 biases. All bias faults were exhaustively tested. 1% of the links were then randomly tested for weight faults.

of ANN's is very good. Moreover, the first replication yields greater improvement in fault tolerance than later, successive replications. This is especially true of the larger complex benchmark nets. It suggests that one extra replication may be the best compromise, since the number of replications necessary for complete tolerance is prohibitively large. In fact, if more than one extra replication is to be employed, it is better to use a conventional TMR scheme with a majority voter to achieve complete fault tolerance.

VI. Conclusion

A method was proposed to estimate fault tolerance of feedforward ANN's and the redundancy required to achieve it. Fault models appropriate for hardware implementations were presented. A procedure was developed to build fault tolerant ANN's by replicating the hidden units. It relies on the intrinsic sum-of-products operation performed by the units to overcome faults. Based on this procedure, metrics were devised to measure fault tolerance as a function of redundancy. A lower bound on the redundancy required to achieve complete fault tolerance was analytically derived. This general result holds regardless of the topology or the specifics of the underlying problem. It shows that if any of the links feeding the output units is essential, then the ANN needs triple modular or higher redundancy to achieve complete fault tolerance.

Analytical and simulation results based on the proposed metrics show that the minimum number of groups *sufficient* for achieving complete fault tolerance is usually much higher than the minimum number *necessary* which was established by the general lower bound. Substantial amount of simulation data also indicates that the actual redundancy needed for a realizable net is very high. An important implication is that the conventional TMR scheme of triplication and majority voting is the best way to achieve complete fault tolerance for most ANN's.

Even though the amount of redundancy needed for complete fault tolerance is prohibitive, the data illustrates that ANN's do possess good partial fault tolerance to begin with (without any extra redundancy). It can be further enhanced by adding moderate amounts of redundancy. In particular, the first extra replication yields the maximum improvement in fault tolerance as compared with later successive replications.

It is evident that efficient testing strategies must be devised to test ANN's as they grow larger. A simple random testing strategy was proposed for large nets where exhaustive testing is prohibitive. This testing method is seen to yield estimates that are very close to the exhaustive-test generated values. Our results demonstrate that currently used learning algorithms develop nonuniform weights and biases with a few that are critical and many others that are insignificant. Merely providing extra units is therefore insufficient. Future extensions should include modifications of the learning algorithms to develop the specific weights and biases that optimize fault tolerance.

Appendix
Proof of Theorem 1

In the following it is assumed that there are M output units and P I/O or pattern pairs in the training set. Without loss of generality, assume that the ith output unit generates erroneous logical value for the kth I/O pattern. Let the fan-in of the ith output unit be N_i. The outputs of the hidden units (that feed

the ith output unit under consideration) for the kth pattern are denoted by $x_1^k, x_2^k, \cdots x_{N_i}^k$ respectively, where $1 \leq k \leq P$. The total input to unit i, denoted by *resultant_input*$_i$, for the kth pattern is then

$$\textit{resultant\_input}_i = \sum_{j=1}^{N_i} w_{ij} x_j^k - \textit{bias}_i. \tag{4}$$

The output of the ith unit is "1" if *resultant_input*$_i > 0$ and it is "0" if *resultant_input*$_i < 0$. Without loss of generality, assume that the output of the ith unit goes wrong, i.e., switches its logical level when the weight of the link connecting it to hidden unit 1 is set to zero (i.e., upon a stuck-at 0 fault on this link), for the kth pattern.

I) First consider the case when hidden unit 1 has asymmetric sigmoidal output, i.e., $x_1^k \in (0, 1)$. There are two subcases: A) The correct output has a logical level "1" that switches to an incorrect logical level "0" and B) vice versa, i.e., a correct output of logical level "0" switches to an incorrect logical level "1" upon the fault (i.e., upon setting $w_{i1} = 0$).

A) Level switch from "1" to "0" upon fault: In this case, for correct operation

$$\sum_{j=1}^{N_i} w_{ij} x_j^k = w_{i1} x_1^k + \textit{rest} > 0$$

where

$$\textit{rest} = \sum_{j=2}^{N_i} w_{ij} x_j^k - \textit{bias}_i \tag{5}$$

whereas

$$0 \cdot x_1^k + \textit{rest} < 0 \tag{6}$$

upon a stuck-at 0 fault on w_{i1}.

The above equations imply that

$$\textit{rest} < 0 \quad \text{and} \quad w_{i1} > 0. \tag{7}$$

Setting

$$\textit{rest} = -\textit{rest}' \quad \text{and} \quad w_{i1} = w'_{i1} \tag{8}$$

the following equations are obtained

$$w'_{i1} x_1^k - \textit{rest}' > 0 \qquad \text{correct operation} \tag{9}$$

$$0 - \text{rest}' = -\text{rest}' < 0$$

incorrect operation when w_{i1} gets stuck-at zero (10)

where

$$\textit{rest}' > 0; \qquad w'_{i1} > 0. \tag{11}$$

Now consider a fault where w'_{i1} changes its sign and gets stuck at $-w'_{i1}$. In that case the resultant input to the ith output unit (for pattern k) is

$$\textit{resultant\_input}_i = -w'_{i1} x_1^k - \textit{rest}' < 0. \tag{12}$$

If n additional replications are to restore the original polarity, then

$$\frac{|-w'_{i1} x_1^k - \textit{rest}'|}{w'_{i1} x_1^k - \textit{rest}'} = \frac{w'_{i1} x_1^k - \textit{rest}'}{w'_{i1} x_1^k - \textit{rest}'} < n$$

or, on rearrangement

$$n > 1 + \frac{2 \cdot \textit{rest}'}{w'_{i1} x_1^k - \textit{rest}}. \tag{13}$$

Since $\textit{rest}' > 0$, and $w'_{i1} x_1^k - \textit{rest}' > 0$, the above equation implies that

$$n > 1 + \epsilon$$

with

$$\epsilon > 0 \quad \text{or } n \geq 2, \text{ i.e., the number of groups is } n + 1 \geq 3. \tag{14}$$

B) If the logical level switches from "0" to "1," the derivation remains almost identical. In particular, the inequalities in (5) and (6) get reversed. Equations (7) and (8) now become

$$\textit{rest} > 0 \quad \text{and} \quad w_{i1} < 0 \tag{15}$$

$$\textit{rest=rest}' \quad \text{and} \quad w_{i1} = -w'_{i1}. \tag{16}$$

The following equations are obtained instead of (9), (10), (11)

$$\textit{rest}' - w'_{i1} x_1^k < 0: \qquad \text{correct operation} \tag{17}$$

$$\textit{rest}' - 0 = \textit{rest}' > 0:$$

incorrect operation when w'_{i1} gets stuck at zero (18)

where

$$\textit{rest}' > 0; \qquad w'_{i1} > 0. \tag{19}$$

Now consider a fault where w'_{i1} changes its sign and gets stuck at $-w'_{i1}$. In that case the resultant input to the ith output unit (for pattern k) is

$$\textit{resultant\_input}_i = \textit{rest}' + w'_{i1} x_1^k > 0. \tag{20}$$

If n additional replications are to restore the original polarity, then

$$\frac{\textit{rest}' + w'_{i1} x_1^k}{|\textit{rest}' - w'_{i1} x_1^k|} = \frac{w'_{i1} x_1^k + \textit{rest}'}{w'_{i1} x_1^k - \textit{rest}'} < n$$

or, on rearrangement

$$n > 1 + \frac{2 \cdot \textit{rest}'}{w'_{i1} x_1^k - \textit{rest}'} \tag{21}$$

which is identical to (13). Q.E.D.

II) Now consider the case when hidden unit 1 has symmetric sigmoidal output, i.e., $x_1^k \in (-1, 1)$. The proof is almost identical to the above proof. We consider two cases: A) $w_{i1} > 0$, and B) $w_{i1} < 0$. (under the assumption that w_{i1} is nonzero to begin with). Each is further classified into two subcases.

1) The logical level switches from "1" to "0" and
2) vice versa, i.e., the logical level switches from "0" to "1."

The proofs for each of these cases are almost identical to either case (IA) or (IB) above. In particular, the proof for case (IIAi, i.e., $w_{i1} > 0$ and level switches from "1" to "0") is almost identical to that of case (IA). Proofs for cases (IIA-2), (IIB-1), and (IIB-2) are similar to cases (IB), (IA), and (IB), respectively. The details for these cases are therefore omitted for the sake of brevity and can be found in [23]. Q.E.D.

REFERENCES

[1] M. J. Carter, "The illusion of fault tolerance in neural nets for pattern recognition and signal processing," in *Proc. Technical Session on Fault Tolerant Integrated Systems,* Univ. New Hampshire, Durham, 1988.

[2] Y. Le Cun, J. S. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2,* D. S. Touretzsky, Ed. Morgan Kaufman, 1990. pp. 598–605.

[3] M. J. Carter, F. J. Rudolph, and A. J. Nucci, "Operational fault tolerance of CMAC networks," in *Advances in Neural Information Processing Systems 2,* D. S. Touretzsky, Ed. Morgan Kaufman, 1990.

[4] T. R. Damarla and P. K. Bhagat, "Fault tolerance of neural networks," in *Proc. Southeastcon.* New York; IEEE Computer Society Press, 1989, pp. 328–331.

[5] T. Petsche and B. W. Dickinson, "Trellis codes, receptive fields, and fault tolerant, self-repairing neural networks," *IEEE Trans. Neural Networks,* vol. 1, pp. 154–166, Jun. 1990.

[6] P. W. Protzel, D. L. Palumbo, and M. K. Arras, "Performance and fault tolerance of neural networks for optimization," in *Proc. Int. Joint Conf. Neural Networks (IJCNN),* Jan 1990, pp. 455–459.

[7] L. A. Belfore II and B. W. Johnson, "The fault tolerance characteristics of neural networks," *Int. J. Neural Networks Res. Appl.,* pp. 24–41, Jan 1989.

[8] ——, "Analysis of the faulty behavior of synchronous neural networks," *IEEE Trans. Computers,* pp. 1424–1428, Dec. 1991.

[9] M. D. Emmerson, R. I. Damper, *et al.,* "Fault tolerance and redundancy of neural nets for the classification of acoustic data," in *Proc. Int. Conf. Acoustics Speech Signal Processing (ICASSP),* vol. 2, Toronto, Canada, May 1991, pp. 1053–1056.

[10] C. H. Sequin and R. D. Clay, "Fault tolerance in artificial neural networks," in *Proc. Int. Joint Conf. on Neural Nets (IJCNN),* vol. 1, San Diego, CA, June 1990, pp. I-703–I-708.

[11] C. Neti, M. H. Schneider, and E. D. Young, "Maximally fault tolerant neural networks," *IEEE Trans. Neural Networks,* vol. 3, pp. 14–23, Jan. 1992.

[12] R. D. Clay and C. H. Sequin, "Fault tolerance training improves generalization and robustness," in *Proc. Int. Joint Conf. Neural Nets (IJCNN),* vol. 1, Baltimore, MD, June 1992, pp. I-769–I-774.

[13] B. E. Segee and M. J. Carter, "Comparative fault tolerance of parallel distributed processing networks (debunking the myth of inherent fault tolerance)," Intelligent Structures Group, Robotics Lab., Electrical and Computer Engineering Dept., Univ. of New Hampshire, Durham, Tech. Rept. ECE.IS.92.07, Feb. 1992.

[14] D. S. Phatak and I. Koren, "Fault tolerance of feedforward neural nets for classification tasks," in *Proc. Int. Joint Conf. Neural Nets (IJCNN),* vol. 2, Baltimore, MD, June 1992. pp. II-386–II-391.

[15] A. E. Bryson and Y. C. Ho, *Applied Optimal Control.* Hemisphere Publ. Corp., 1975.

[16] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing,* vol. 1: *Foundations.* Cambridge, MA: MIT Press, 1986.

[17] Y. le Cun, "Une Procedure d'appresntissage pour reseau a sequil assymetrique (a learning procedure for asymmetric threshold network)," in *Proc. Cognitiva,* Paris, France, 1985, pp. 599–606.

[18] ——, "A theoretical framework for back-propagation," in D. Touretzsky, G. Hinton, and T. Sejnowski, (Ed.), *Proc. 1988 Connectionist Models Summer School,* San Mateo, CA, 1988, pp. 21–28.

[19] S. E. Fahlman, "Faster learning variations on back propagation: An empirical study," in D. Touretzsky, G. Hinton, and T. Sejnowski (Ed.), *Proc. 1988 Connectionist Models Summer School,* San Mateo, CA.

[20] S. E. Fahlman and C. Lebiere, "The cascade correlation learning architecture," in D. S. Touretzsky (Ed.), *Neural Information Processing Systems 2.* Morgan Kaufman, 1990, pp. 524–532.

[21] Y. Izui and A. Pentland, "Analysis of neural networks with redundancy," *Neural Computation,* vol. 2, pp. 226–238, 1990.

[22] W. P. Lincoln and J. Skrzypek, "Synergy of clustering multiple back propagation networks," in D. S. Touretzsky (Ed.), *Neural Information Processing Systems 2.* Morgan Kaufman, 1990, pp. 650–657.

[23] D. S. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets," Tech. Rep. TR-92-CSE-26, Electrical and Computer Engineering Dept., Univ. of Massachusetts, Amherst, July 1992.

[24] D. S. Phatak, H. Choi, and I. Koren, "Construction of minimal $n-2-n$ encoders for any n," *Neural Computation,* vol. 5, pp. 783–794, Sept. 1993.

[25] S. E. Fahlman *et al.,* "Neural nets learning algorithms and benchmarks database," maintained by S. E. Fahlman *et al.* at the Computer Science Dept., Carnegie Mellon Univ.

[26] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks,* pp. 75–89, 1988.

[27] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Syst.,* vol. 1, pp. 145–168, 1987.

[28] K. J. Lang and M. J. Witbrock, "Learning to tell two spirals apart," in D. Touretzsky, G. Hinton, and T. Sejnowski (Ed.), *Proc. 1988 Connectionist Models Summer School,* San Mateo, CA, 1988.

[29] A. J. Robinson and F. Fallside, "A dynamic connectionist model for phoneme recognition," in *Proc. nEuro,* Paris, June 1988.

Dhananjay S. Phatak received the B. Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay, in 1985; the M.S. degree in microwave engineering in 1990, and the Ph.D. degree in computer systems engineering in 1993, both from the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst.

He is currently an Assistant Professor of Electrical Engineering at the State University of New York, Binghamton. His current research interests are in theory, applications, and electronic as well as optical implementations of neural networks, digital and analog VLSI design and CAD, fault tolerant computing, computer arithmetic algorithms and their VLSI implementations. His research interests also include microwave and optical integrated circuits.

Dr. Phatak has published papers in various journals and presented papers at conferences related to computer arithmetic algorithms and their implementations.

Israel Koren (S'72–M'76–SM'87–F'91) received the B.Sc., M.Sc., and D.Sc. degrees from the Technion-Israel Institute of Technology, Haifa, in 1967, 1970, and 1975, respectively, all in electrical engineering.

He is currently a Professor of Electrical and Computer Engineering at the University of Massachusetts, Amherst. Previously he was with the Departments of Electrical Engineering and Computer Science at the Technion-Israel Institute of Technology. He also held visiting positions with the University of California at Berkeley, University of Southern California, Los Angeles, and the University of California, Santa Barbara. He has been a Consultant to Intel, Digital Equipment Corp., National Semiconductor, Tolerant Systems, and ELTA. His current research interests are fault-tolerant VLSI architectures, models for yield and performance, floor-planning of VLSI chips, and computer arithmetic.

Dr. Koren has published extensively in the IEEE TRANSACTIONS and has also served as program committee member for numerous conferences. He has edited and coauthored the book *Defect and Fault-Tolerance in VLSI Systems,* vol. 1 (Plenum, 1989). He is the author of the textbook *Computer Arithmetic Algorithms* (Prentice-Hall, 1993).