Fault Tolerance of Feedforward Neural Nets for Classification Tasks *

D. S. Phatak and I. Koren
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA 01003

ABSTRACT

A method is proposed to estimate the fault tolerance of feedforward Artificial Neural Nets (ANNs) and synthesize robust nets. Fault models are presented and a procedure is developed to build fault tolerant ANNs by replicating the hidden units. Based on this procedure, metrics are devised to quantify the fault tolerance as a function of redundancy. A significant amount of redundancy is shown to be necessary to achieve complete fault tolerance even if only single faults are considered. Furthermore, lower bounds on the required redundancy are analytically derived for some canonical problems. Our results indicate that ANNs have good partial fault tolerance and degrade gracefully. In particular, just one extra replication is seen to considerably improve the fault tolerance.

I Introduction

Neural computing is rapidly evolving as a viable solution to several problems. For use in applications requiring high reliability, ANNs 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. Several expositions have addressed various aspects of fault tolerance of ANNs [1]-[7]. Investigations by Carter et. al. [1, 2] demonstrate the need to quantitatively evaluate the fault tolerance of ANNs. Simulation results on the XOR problem are reported in [3], but these are quite specific to this one example and the underlying fault model. Fault tolerance of Hopfield type ANNs for optimization problems was investigated in [4]. It, however, does not address fault tolerance in trainable ANNs. Belfore et. al. have developed an analytical technique for estimating the performance of ANNs in presence of faults [5, 6] They construct a Markov model for the ANN by drawing analogy with magnetic spin systems using statistical mechanics. Neti et. al. recently reported numerical results on the synthesis of fault tolerant nets [7]. They add constraints to ensure fault tolerance and look for a solution to the constrained optimization problem. This approach, however, is very computation intensive and more experiments are needed to draw any general conclusions about fault tolerance.

Most researchers concentrate on performance estimation in the presence of faults for *a given net.* To the best of our knowledge, there has been no attempt to relate the fault tolerance to the redundancy necessary to achieve it. In this paper, we propose simple metrics to measure fault tolerance of ANNs for classification tasks and address the issue of redundancy. Our approach also reveals a simple way to improve the fault tolerance of *any* net to a desired level by adding redundancy.

II Topology and Fault Model

We have considered 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 ith unit is given by

$$o_i = f(resultant\_input_i) \quad \text{where} \quad f(x) = \frac{a}{1+e^{-x}} - b \quad \text{and} \quad resultant\_input_i = \sum_{j=1}^{N_i} w_{ij} o_j - bias_i \qquad (1)$$

Here, N_i is the total number of units that feed the ith unit, w_{ij} is the weight 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 $(-1, 1)$ or ($a = 1$, $b = 0$) 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.

*This work was supported in part by NSF Grant MIP 88-05586

0-7803-0559-0/92 $3.00 © 1992 IEEE

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 Boolean or switching function, because the inputs and weights of the ANN are real valued. Training stops when the outputs are within 5% of the target. For testing purpose, an output is considered wrong if it switches the logical level. This happens if the *resultant_input* to a unit switches its sign [eq. (1)]. Besides the Back Propagation learning algorithm [8] and its variants, we have also considered the Cascade Correlation learning algorithm [9]. These two algorithms often lead to very diverse architectures.

The fault tolerance properties of ANNs can be broadly classified as
(*i*) On-line or Operational fault tolerance, and (*ii*) Fault-tolerance through re-learning.
In this paper, we consider the former. Since adjustable weights are at the heart of any learning process, we concentrate on faults that affect them. These faults are assumed to be of the permanent stuck-at type, i.e., weights are set stuck-at $\pm W$ and 0, where W is the maximum magnitude attained by any of the weights during learning. The former can model a link shorted to voltage supply or ground, while the latter can describe a burnt out or open connection. Setting a bias to "1"or "0"results in a unit's output stuck at high or low and models the failure of a unit. We consider only single weight or bias faults. The general analysis with multiple faults quickly becomes unmanageable. Moreover, analysis of single faults does bring out the general trends and helps identify the properties necessary for fault tolerance.

III A General Procedure to Build a Fault Tolerant Net

A simple way to achieve fault tolerance is through redundancy. TMR (Triple Modular Redundancy) and n-MR schemes with majority voters have been widely used for system reliability enhancement. We have extended this approach to ANNs for classification tasks. 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 eq. (1) it is easy to verify that

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

The output remains at the same 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 outputs as the original net. This process of replication is illustrated in Figure 1. Each replication is referred to as a group or a module. The number of input and output units is unchanged, only the biases of the output units are scaled. Each correctly working group (or module) 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

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

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 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.

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. [10] have shown that the convergence speed during learning and the solution accuracy improve with the number of replications. Similar observations have been made in [11] where it is inferred that clustering of multiple nets improves not only the fault tolerance but

also the performance. The required redundancy turns out to be too large to be practical. However, the above procedure gives a simple way of evaluating fault tolerance as a function of the number of replications i.e. redundancy, as illustrated later in the section on partial fault tolerance. It therefore serves to measure the fault tolerance of the ANN as a function of redundancy. This is the main motivation behind the replication scheme.

IV Analytical and Simulation Results

We have applied the above process of replications to several problems. We first summarize the analytical results proved so far for the Encoding and XOR problems [8, chapter 8]. The proofs involve geometrical constructions and have been omitted for the sake of brevity. The proofs and other details can be found in [12]. Here, $n - m - n$ refers to an $n \times n$ encoding problem with m hidden units.

Theorem 1 : *A single hidden unit can solve only the 2×2 encoding problem, while only 2 hidden units are sufficient to solve any $n \times n$ size problem. The same is true of the complementary encoding problem.*

Thus, the minimum size net is known a-priori even for an arbitrary $n \times n$ encoding problem. This makes it is possible to derive lower bounds the amount of redundancy needed for complete fault tolerance. Lower bounds for the XOR problem can also be derived in an identical manner.

Theorem 2 : *(i) For the 2-1-2 encoding problem, the minimum number of modules needed for tolerating all single faults is 4 if the sigmoid is asymmetric* $[0, 1]$. *The corresponding minimum number of modules is 3 if the sigmoid is symmetric* $[-1, 1]$.

(ii) The above bounds hold for any $n - 2 - n$ encoding problem or its complement.

(iii) The above bounds on the number of modules hold for the XOR problem.

Extensive simulations on the above and several other problems such as binary addition, TC problem, multi input parity [8, chapter 8] indicated that the number of replications needed is large. This is expected, since the training procedure is not geared toward fault tolerance. Learning rate and other parameters of the learning algorithm need to be fine tuned in order to generate nets that meet the above lower bounds. The results show that most of the time, the weights developed during the learning phase are quite non-uniform. A few of those with large magnitude are dominant. Their failure causes the net to malfunction, while many others can be dispensed without significantly degrading the performance. Merely providing a large number of hidden units is therefore insufficient. The training procedure must also be modified to equitably distribute the computation among all units.

All of the above examples are well known canonical problems in the ANN literature. Moreover, some of these are analytically tractable as seen above. These problems, however, are not the best candidates to analyze the fault tolerance of ANNs. They require small nets and digital logic can solve such problems much more efficiently. ANNs are expected to perform better on larger, random problems. Our objective was to look for intrinsic characteristics of ANNs. The above analysis does brings out such a characteristic: a significant amount of redundancy is needed to achieve complete fault tolerance. Next, we consider some realistic benchmark problems well suited for ANNs. The training and testing data and a description of the above benchmarks was obtained from the public database maintained by Fahlman et. al. at CMU [13].

The Two Spirals Classification Benchmark [13]: The task is to learn to discriminate between two sets of training points which lie on two distinct spirals in the x-y plane. These spirals coil three times around the origin and around one another. The training data consists of two sets of points, each with 97 members (three complete revolutions at 32 points per revolution, plus endpoints). Each point is represented by two floating-point numbers, the x and y coordinates that are the inputs, plus an output value of 0.0 for one set, 1.0 for the other set.

The Sonar Benchmark [13]: The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. The data set contains 111 patterns obtained by bouncing sonar signals off a metal cylinder at various angles and under various conditions. Also included are 97 patterns obtained from rocks under similar conditions. Each pattern is a set of 60 numbers in the range 0.0 to 1.0 (inputs). Each number represents the energy within a particular frequency band, integrated over a certain period of time. Further details can be found in [13].

The Vowel Benchmark [13]: The objective is speaker independent recognition of vowels. There are 90 utterances of each of the 11 vowels under consideration. For each utterance, there are 10 floating point values that constitute the inputs to the ANN. The ANN is trained to indicate which of the 11 vowels was spoken.

We looked at 2 cases. In one the 90 utterances are divided into 48 training and 42 testing patterns. In the other all 90 cases were used during training phase. We also tried different output encodings as described later.

For each of these problems, many nets were generated using back propagation and cascade correlation algorithms. Every single net with symmetric (asymmetric) sigmoidal units that we trained so far for every single problem, (benchmark or other) needed more than 3(4) modules for complete fault tolerance. The simulations thus corroborate the analytical results derived above and also reveal a non uniform distribution of computational load. In the Sonar benchmark, the fan–in of the output unit was as high as 63 in some nets. 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 (more than 7) to achieve complete fault tolerance.

From the above analytical and simulation results we conjecture that :
(i) A feedforward net consisting of sigmoidal units, whose outputs can be classified into two distinct logical levels viz. "*on*"or "*off*" is either completely fault tolerant, or
(ii) It needs more than 4(3) modules in case of asymmetric(symmetric) sigmoidal activation functions to achieve complete fault tolerance for all single faults.

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. 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 problems mentioned above. Results for the 3-2-3 Encoding problem and the benchmarks are illustrated in Figures 2 to 5. To generate these figures, each weight was in turn set at $+W$, 0 and $-W$, where W is the maximum magnitude. Each bias was set at $\pm W$. Setting a bias to 0 has no physical significance. In contrast, it is essential to set a weight to zero in order to model the disconnection of a link. For each of the above settings (faults) all the input/output 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 $\times$ number of input/output patterns $\times$ total number of faults). Note that the total number of faults = (the number of links present in the net $\times$ the types of faults simulated per link + the number of biases present in the net $\times$ 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 plotted on the y axes in Figures 2 to 5. The partial fault tolerance was calculated for increasing number of replications until it reached a satisfactory level (usually 95 % or higher).

Figure 2 shows the partial fault tolerance of 3-2-3 encoding nets. One of these was hand crafted to optimize (through heuristics and trial and error) the fault tolerance. It is seen that the hand crafted solution shows better partial fault tolerance without any replications (point indicating 0 replications on the x axis). Thus, it is possible to get better fault tolerance, merely by a proper selection of weights and biases, without any extra hardware investment. The hand crafted solution also achieves complete fault tolerance with less replications, again illustrating that specific weights and biases can improve the fault tolerance.

Figure 3 shows the partial fault tolerance of nets for the Two-Spirals benchmark. These were generated by the cascade correlation algorithm. Out of all the nets (more than 50) we picked the best and the worst performers. It should be noted that the difference in the fraction of faults tolerated is not much, indicating that only a few faults are the fault tolerance bottlenecks in either case, and that most other faults are tolerated. However, it does bring out the possibility of achieving even better performance if the weights can be tailored to yield higher fault tolerance.

Figure 4 shows similar plots for the sonar benchmark. All the nets shown in that figure were generated by the back propagation learning algorithm, had 1 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 the symmetric sigmoid activation functions. The partial fault tolerance data for these cascade correlation nets shows identical trends and is therefore, excluded from the figure for the sake of clarity. It should be noted that the plots of fraction of outputs that remain correct *appear* to reach the value 1.0 from $x = 4$ onwards in this figure. However, this is only an illusion, due to the limited resolution of the plotter.

Figure 5 shows partial fault tolerance of nets trained on the vowel benchmark. All of the i/o patterns were used during training. The figure shows plots for two nets. One had 1 hidden and 11 output units, exactly one of which is "on" for each of the vowels (localized output encoding). The other net had 3 hidden units and 4 outputs units. The output units generated 4 bit binary numbers from 0 to 10 to indicate the one of the 11 vowels which is a straight binary encoding of the outputs.

Though complete fault tolerance needs a large number of modules, just one additional replication significantly improves the performance. This is especially true of the larger nets as illustrated in figures 3,4, and 5 by the benchmark nets.

VI Conclusion

A method was proposed to estimate the redundancy and fault tolerance of feedforward ANNs for classification tasks. Fault models were presented and a procedure was developed to build fault tolerant ANNs by replicating the hidden units. Based on this procedure, a metric was devised to measure fault tolerance as a function of redundancy. The metric was applied to several problems and the results illustrate that a significant amount of redundancy is needed to achieve complete fault tolerance even if only single failures are considered. Lower bounds on the redundancy needed for complete fault tolerance were analytically derived for some canonical problems. The symmetric version of the activation function was shown to yield higher fault tolerance. Results indicate that ANNs exhibit good partial fault tolerance and degrade gracefully. For large, complex problems, an initially good partial fault tolerance was seen to further improve to a satisfactory level with just one additional replication. The results show that currently used learning algorithms develop non-uniform weights and biases with a few that are critical and many others that are insignificant. Merely providing extra units is therefore insufficient. Further extensions of the work include a modification of the learning algorithm in order to yield the specific weights and biases that optimize fault tolerance.

References

[1] M.J. Carter, "The Illusion of Fault Tolerance in Neural Nets for Pattern Recognition and Signal Processing," in *Proceedings of Technical Session on Fault Tolerant Integrated Systems*, University of New Hampshire, Durham, 1988.

[2] 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.

[3] T. R. Damarla and P. K. Bhagat, "Fault Tolerance of Neural Networks," in *Proceedings of Southeastcon*, pp. 328–331, IEEE Computer Society Press, 1989.

[4] P. W. Protzel, D. L. Palumbo and M. K. Arras, "Performance and Fault Tolerance of Neural Networks for Optimization," *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pp. 455–459, Jan 1990.

[5] L. A. Belfore II and B. W. Johnson, "The fault tolerance characteristics of Neural Networks," *International Journal of Neural Networks Research and Applications*, pp. 24–41, Jan 1989.

[6] L. A. Belfore II and B. W. Johnson, "Analysis of the Faulty Behaviour of Synchronous Neural Networks," *IEEE Transactions on Computers*, pp. 1424–1428, Dec 1991.

[7] C. Neti, M. H. Schneider and E. D. Young, "Maximally Fault Tolerant Neural Networks," *IEEE Transactions on Neural Networks*, vol. 3, pp. 14–23, Jan. 1992.

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

[9] S. E. Fahlman and C. Lebiere, "The Cascade Correlation Learning Architecture," in *Neural Information Processing Systems 2* (D. S. Touretzsky, ed.), pp. 524–532, Morgan Kaufman, 1990.

[10] Y. Izui and A. Pentland, "Analysis of Neural Networks with Redundancy," *Neural Computation*, vol. 2, pp. 226–238, 1990.

[11] W. P. Lincoln and J. Skrzypek, "Synergy of Clustering Multiple Back Propagation Networks," in *Neural Information Processing Systems 2.* (D. S. Touretzsky, ed.), pp. 650–657, Morgan Kaufman, 1990.

[12] D. S. Phatak and I. Koren., "A Study of Fault Tolerance Properties of Artificial Neural Nets," tech. rep., Electrical and Computer Engineering Department, University of Massachusetts, Amherst, 1991.

[13] *Neural Nets Learning Algorithms and Benchmarks Database.* maintained by S. E. Fahlman et. al. at the Computer Science Dept., Carnegie Mellon University.

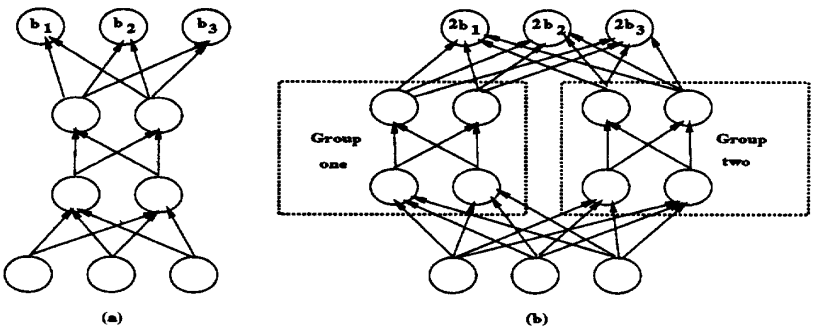

Figure 1: Replication of Hidden Units. (a) Original net. (b) One extra group of hidden units. Biases of Output units are also scaled by 2.0

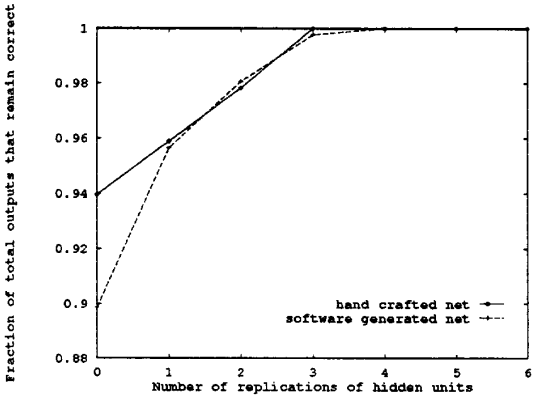

Figure 2 : Comparison of partial fault tolerance of 3-2-3 Encoding nets. A hand crafted net and the best (out of 100) Back Propagation generated net.

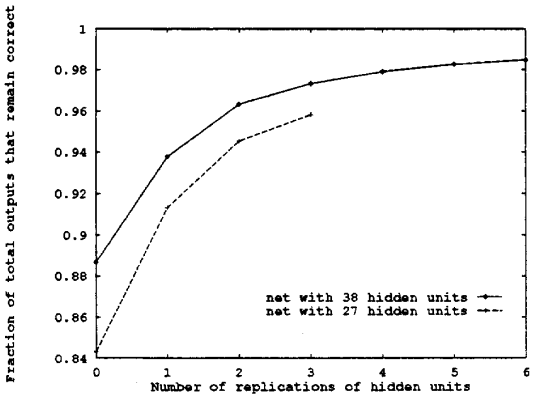

Figure 3 : Partial fault tolerance of nets trained on the **Two Spirals Benchmark [13]** by the Cascade Correlation algorithm. (a) Net with 38 hidden units. (b) Net with 27 hidden units.

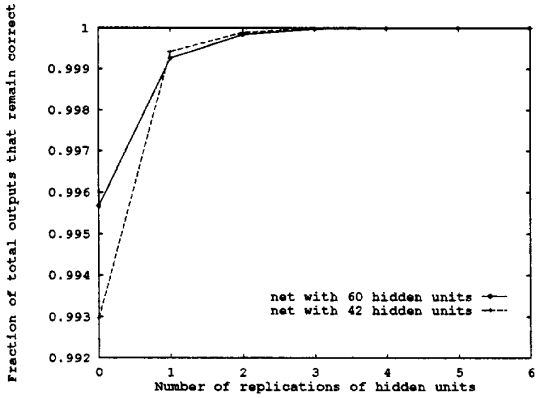

Figure 4 : Partial fault tolerance of nets trained on the **Sonar Benchmark [13]** by Back Propagation. (a) Net with 60 hidden units. (b) Net with 42 hidden units.

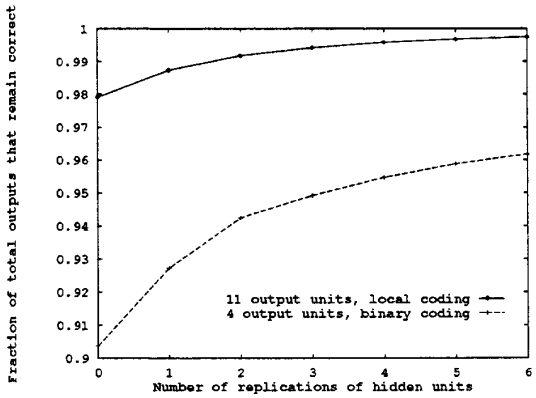

Figure 5 : Partial fault tolerance of nets trained on the **Vowel Benchmark [13]** by Cascade Correlation. (a) Net with 1 hidden and 11 output units. One output is "on" for each of the 11 vowels (localized encoding). (b) Net with 3 hidden and 4 output units which produce 4 bit binary numbers from 0 to 10 to indicate the vowel.