Complete and Partial Fault Tolerance of Feedforward Neural Nets *

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

Technical Report No. TR-92-CSE-26 †
July 1992, last revised April 1993

ABSTRACT

A method is proposed to estimate the fault tolerance of feedforward Artificial Neural Nets (ANNs) 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 ANNs by replicating the hidden units. It exploits the *intrinsic* weighted summation operation performed by the processing units in order to overcome faults. It is simple, robust and is 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, irrespective 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 ANNs.

Although the redundancy needed for complete fault tolerance is substantial, the results do show that ANNs 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 to 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 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.

---

* This work was supported in part by NSF Grant MIP 90-13013.

† Abbreviated version of this report is in print in the *IEEE Transactions on Neural Nets* (to appear in 1994).

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. Fault tolerance of ANNs is often taken for granted or treated as a subsidiary issue [1, 2]. Investigations by Carter et. al. [1, 3] indicated that ANNs are not always fault tolerant and demonstrated the need to quantitatively evaluate their robustness. Several expositions have addressed various aspects of fault tolerance of ANNs [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 re-learning. The issue of on-line or operational fault tolerance is not considered there. Fault tolerance of Hopfield type ANNs for optimization problems was investigated in [6]. 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 [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 multi layer version, which had a single hidden layer consisting of 5 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, it 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 of graceful degradation for continuous valued ANNs. 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 0 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. They however did not try to evaluate how much redundancy is necessary to achieve a given level of fault tolerance or how to make a give 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 0 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 in order to overcome faults. Based on this scheme, metrics were devised to measure the fault tolerance of ANNs 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 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 0 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 TMR (triplication and majority vote) is a better way of achieving complete fault tolerance for most ANNs. 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 presented in Section V. The results show that the actual redundancy needed to synthesize completely fault tolerant nets 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 VI 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

$$o_i = f(resultant\_input_i) \qquad \text{where} \qquad f(x) = \frac{a}{1+e^{-x}} - b \qquad \text{and}$$

$$resultant\_input_i \quad = \quad \sum_{j=1}^{N_i} w_{ij} o_j - bias_i \tag{1}$$

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 ANNs can be broadly classified as (i) On-line or Operational fault tolerance, and (ii) Fault-tolerance through re-learning. The first refers to the ability to function in the presence of faults without any re-learning 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 re-allocating 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, re-learning, or re-configuration 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 ANNs, 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 0 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 0. 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 0 corresponds to a disconnection of that link. Setting a bias to 0, on the other hand, has no physical significance. Bias faults are therefore limited to stuck–at $\pm W$. During testing, an output is classified as follows:

$$output = \begin{cases} \text{“}\mathbf{1}\text{”} & \text{if} \quad resultant\_input = net\_input - bias > 0 \\ \text{“}\mathbf{0}\text{”} & \text{if} \quad resultant\_input = net\_input - bias < 0 \end{cases} \tag{2}$$

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 [eq. (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, ANNs 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. TMR (Triple Modular Redundancy) and n-MR schemes with majority voters have been widely used for reliability enhancement in digital systems. Extension of these schemes to ANNs is the most obvious way to render them fault tolerant. In a TMR scheme for ANNs, 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 very large. Also, majority voting on analog signals is likely to be considerably harder than voting on digital signals. Hence, the voter itself could be much more complex than the majority voters in digital systems. Nevertheless, it is a very robust, well proven technique, applicable to any net.

ANNs, however, are quite different from digital systems. Typically, they are comprised 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 ANNs and biological systems along with the highly parallel mode of operation suggests that ANNs could be *inherently* fault tolerant. Any such *intrinsic* fault tolerance in ANNs 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 in order to achieve fault tolerance. It exploits the intrinsic weighted summation operation performed by the units in order 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 equations (1) and (2) it is easy to verify that the output changes its logical level only if the $resultant\_input$ changes its sign. 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 outputs (logical) as the original net. This process of replication is illustrated in Figure 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

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

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 ANNs. 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 $\frac{1}{g}$. 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 in image processing instead of computing the mean values.

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 ANNs 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, irrespective 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 "**0**" 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 atleast 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 0 fault on any of the links feeding any output unit causes the unit to produce a classification error; then, at least 2 extra replications, or equivalently, 3 or more groups of hidden units are required to achieve complete fault tolerance for all single faults.*

Proof : 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 this 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 $1 \le k \le P$. From the perspective of the ith output unit, each of the P patterns is thus mapped onto a point in the N_i dimensional space, where the jth coordinate of a point is the output of the jth hidden unit that feeds the ith output unit. The output unit can be thought to implement a hyperplane in an $(N_i + 1)$ dimensional space, such that some of the points are on its positive side and some on its negative side. A point is on the positive (negative) side of a hyperplane implemented by an output unit if its output for that pattern is positive (negative). The total input to unit i, denoted by $resultant\_input_i$, for the kth pattern is then

$$resultant\_input_i = \sum_{j=1}^{N_i} w_{ij} x_j^k \ - bias_i \tag{4}$$

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 sub–cases: (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,

$$\sum_{j=1}^{N_i} w_{ij} x_j^k = w_{i1} x_1^k + rest > 0 \qquad \text{where} \qquad rest = \sum_{j=2}^{N_i} w_{ij} x_j^k \ - bias_i \qquad \text{whereas,} \tag{5}$$

$$0 \cdot x_1^k + rest < 0 \qquad \text{upon a stuck-at 0 fault on } w_{i1} \tag{6}$$

The above equations imply that

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

$$\text{Setting} \qquad rest = -rest^{'} \quad \text{and} \quad w_{i1} = w_{i1}^{'} \tag{8}$$

the following equations are obtained:

$$w_{i1}^{'} x_1^k - rest^{'} > 0 \qquad : \text{correct operation} \tag{9}$$

$$0 - rest^{'} = -rest^{'} < 0 \qquad : \text{incorrect operation when } w_{i1} \text{ gets stuck-at } 0 \tag{10}$$

$$\text{where,} \qquad rest^{'} > 0 \ ; \ w_{i1}^{'} > 0 \tag{11}$$

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

$$resultant\_input_i = -w'_{i1}x_1^k - rest' < 0 \tag{12}$$

If n additional replications are to restore the original polarity,

$$\frac{|-w'_{i1}x_1^k - rest'|}{w'_{i1}x_1^k - rest'} = \frac{w'_{i1}x_1^k + rest'}{w'_{i1}x_1^k - rest'} < n \qquad \text{or, on rearrangement,}$$

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

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

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

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

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

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

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

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

$$rest' - 0 = rest' > 0 \qquad : \text{incorrect operation when } w'_{i1} \text{ gets stuck-at } 0 \tag{18}$$

$$\text{where,} \quad rest' > 0 \ ; \ w'_{i1} > 0 \tag{19}$$

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

$$resultant\_input_i = rest' + w'_{i1}x_1^k > 0 \tag{20}$$

If n additional replications are to restore the original polarity,

$$\frac{rest' + w'_{i1}x_1^k}{|rest' - w'_{i1}x_1^k|} = \frac{w'_{i1}x_1^k + rest'}{w'_{i1}x_1^k - rest'} < n \qquad \text{or, on rearrangement,}$$

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

which is identical to equation (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 2 cases (A) $w_{i1} > 0$, and (B) $w_{i1} < 0$. (under the assumption that w_{i1} is non-zero to begin with)

(A) $w_{i1} > 0$: This is further classified into two sub–cases.
(i) The logical level switches from "**1**" to "**0**" : Here, we get

$$w_{i1} x_1^k + rest > 0 \quad : \text{correct operation} \tag{22}$$

$$0 \cdot x_1^k + rest = rest < 0 \quad : \text{when } w_{i1} \text{ gets stuck-at } 0 \tag{23}$$

Since $w_{i1} > 0$ the above equations imply that $x_1^k > 0$. After Setting $rest = -rest'$ and $w_{i1} = w'_{i1}$, the resulting equations are identical to (9) and (10) of case (I–A) above. The remaining steps and equations of the proof are identical to the corresponding case (I–A) above, i.e., consider a fault in which the weight changes its sign and impose the restriction that n additional groups should be able to restore the correct polarity. The resulting equations are identical to equations (12) to (14) above.

(ii) The logical level switches from "**0**" to "**1**" : In this case,

$$w_{i1} x_1^k + rest < 0 \quad : \text{correct operation} \tag{24}$$

$$0 \cdot x_1^k + rest = rest > 0 \quad : \text{when } w_{i1} \text{ gets stuck-at } 0 \tag{25}$$

Since $w_{i1} > 0$, the above equations imply that $x_1^k < 0$. After Setting $rest = rest'$; $w_{i1} = w'_{i1}$ and $x_1^k = -\alpha_1^k$, the resulting equations are identical in form to (17) and (18). The rest of the proof is identical to the corresponding case (I–B) above.

(B) Now consider $w_{i1} < 0$:
(i) If the polarity switches from "**1**" to "**0**" , one obtains

$$w_{i1} x_1^k + rest > 0 \quad : \text{correct operation} \tag{26}$$

$$0 \cdot x_1^k + rest = rest < 0 \quad : \text{when } w_{i1} \text{ gets stuck-at } 0 \tag{27}$$

Since $w_{i1} < 0$ the above equations imply that $x_1^k < 0$. Set

$$rest = -rest' \quad , \quad w_{i1} = -w_{i1}' \quad \text{and} \quad x_1^k = -\alpha_1^k$$

where $\quad rest' > 0 \; ; \; \alpha_1^k > 0 \;$ and $\; w_{i1}' > 0 \quad$ to obtain (28)

$$(-w_{i1}') \cdot (-\alpha_1^k) - rest' = w_{i1}'\alpha_1^k - rest' > 0 \quad : \text{correct operation} \tag{29}$$

$$0 \cdot \alpha_1^k - rest' = -rest' < 0 \quad : \text{when } w_{i1} \text{ gets stuck-at } 0 \tag{30}$$

These equations are identical in form to equations (9) and (10) above. From here on, the steps of the proof are identical to case (I–A) above.

(ii) Finally, if the polarity changes from "**0**" to "**1**" upon a fault, the proof is almost identical to that of case (II-A-ii) above. **Q. E. D.**

Note that the above Theorem is not restricted only to sigmoidal activation functions. It can be extended to incorporate any activation function as long as the function is monotonic.

This result suggests that the conventional TMR scheme is as good or better than the replication of hidden units. While this might be true for most ANNs, there are a few cases when the replication of hidden units needs less overall redundancy. Note that a conventional TMR scheme would need to replicate the output units as well, besides adding the majority voters. If the number of output units is comparable to or higher than the number of hidden units, then the above scheme of triplicating the hidden units needs less overall redundancy, because the replication of output units and the majority voters are avoided. It is not uncommon for a net to have the number output units comparable to or higher than the number of hidden units. The inputs and outputs of an ANN are not free to be chosen but are a part of the task specification. For example, a local encoding can be used to represent the classification outputs (a local encoding means that only one unit corresponding to the category of the exemplar is "on" at a time, all the other units are "off"). In such cases, the number of output units can be significant, depending upon the number of output classes.

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 better for most ANNs.

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 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. In fact the *attainable* or *feasible* lower bound is often higher than the above, as demonstrated by analytical and simulation results in the next section.

V Analytical and Simulation Results on Canonical and Benchmark Problems

We have applied the above process of replications to several problems. Our results indicate that a large amount of redundancy is usually required to achieve complete fault tolerance. For specific problems, the minimum redundancy sufficient to synthesize completely fault tolerant nets has been derived analytically. We begin with the analytical results for the Encoding and XOR problems [16, chapter 8]. Encoding nets are illustrated in Figure 2. Topology of an $n - m - n$ encoding net is shown in Figure 2-a. Here, $n - m - n$ refers to an $n \times n$ encoding problem with m hidden units. There are n input and n output units and m hidden units in the single hidden layer. There is full connectivity between the layers and there are no layer skipping connections. Usually $m < n$ and the hidden units are supposed to develop compact representations or encodings for the inputs and reproduce them at the output layer. Figure 2-b shows a 2-1-2 encoding net and one set of weights and biases (out of the several possible). Further details can be found in [16, chapter 8].

The replication procedure above needs minimal or near-minimal initial net to begin with. We therefore present the following result that shows the minimum number of hidden units needed for solving an $n \times n$ encoding problem.

Theorem 2 : *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.*

The proof is not directly related to fault tolerance issues and has been omitted for the sake of brevity. The proof and other details can be found in [23].

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

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

The proof is included in the appendix.

(ii) The above bounds on the number of groups hold for any $n - 2 - n$ *encoding problem or its complement.*

The proof is very similar to that of part *(i)* above. It is, however, extremely lengthy, does not provide much additional insight and is therefore excluded.

Corollary : *The above bounds on the number of groups hold for the XOR problem.*

The proof is almost identical to that of Theorem 3 and has therefore been omitted.

Note that the proofs for these theorems are *constructive* and these lower bounds are *attainable* unlike the general result derived in the previous section, which says nothing about whether that bound is attainable. These results give both *necessary* and *sufficient* conditions, while the general lower bound of the previous section establishes a *necessary* condition. These results are stronger for yet another reason; they are true irrespective of whether any of the output links is essential. They show a minimum sized net for the encoding and XOR problems, and prove that the minimum sized net *always* needs 4(3) groups to achieve complete fault tolerance for asymmetric(symmetric) sigmoidal units. These results clearly indicate that the *attainable lower bound* is dependent on the specific problem, as well as the type of units and topology.

Extensive simulations on the above and several other problems such as binary addition, TC problem, multi input parity [16, 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. Similar observations were reported in [9] and [13]. Clearly, 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 [24].

The Two Spirals Classification Benchmark [20, 24, 25] : The task is to learn to discriminate between two sets of training points which lie on two distinct spirals in the x-y plane. The two spirals are illustrated in Figure 3. 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 (or -1.0 or -0.5) for one set and 1.0 (or 0.5) for the other set.

The Sonar Benchmark [24, 26] : This is the data set used by Gorman and Sejnowski in their study of the classification of sonar signals using a neural network [26]. 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 about the generation of the data sets can be found in [24, 26].

The Vowel Benchmark [24, 27] : 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 the training phase. We also tried different output encodings. In the localized output encoding, there are 11 output units and exactly one of the 11 units, the one corresponding to the vowel spoken is "*on* ". Straight binary encoding on the other hand uses 4 units to represent binary numbers from 0 to 10 to indicate the vowel spoken.

The NETTalk Benchmark [24, 28] : This a the well–known data set used by Sejnowski et. al. [28] in their study of speech generation using a neural network. The task is to train a network to produce the proper phonemes, given a string of letters as input. This is an example of an input/output mapping task that exhibits strong global regularities, but also has a large number of more specialized rules and exceptional cases. The data set [24] consists of 1000 words along with a phonetic transcription for each word.

The input to the network is a series of seven consecutive letters from one of the training words. The central letter in this sequence is the "current" one for which the phonemic output is to be produced. Three letters on either side of this central letter provide context that helps to determine the pronunciation. There are a few words in English for which this local seven-letter window is not sufficient to determine the proper output. This makes the problem harder since the net should also be able to discern the broader context if possible. Individual words from the training set are moved through the window so that each letter in the word is seen in the central position. Blanks are added before and after the word as needed. Some words appear more than once in a dictionary, with different pronunciations in each case; only the first pronunciation given for each word was used.

For each of the seven letter positions in the input, the network has a set of 28 input units: one for each of the 26 letters in English, and two for the punctuation characters (dash "–" for continuation and underscore "_" for word boundary). Our goal was to have a reasonably large sized net (large number of units and links and large number of training patterns) for the fault tolerance experiments. The NETTalk benchmark provided us with such large nets. For our purpose, it was sufficient to train the net on the word set. It was not necessary to train the net on continuous text. The sentence termination character, i.e., full stop "·" was therefore not used in the input set. Hence, we have 28 input units instead of 29 (which were used in the original experiments [28]). Thus, there are $28 \times 7 = 196$ input units in all.

The output side of the network uses a distributed representation for the phonemes. There are 21 output units representing various articulatory features such as voicing and vowel height. Each phoneme is represented by a distinct binary vector over this set of 21 units. In addition, there are 5 output units that encode stress and syllable boundaries.

We chose a set of 200 words from the 1000 word list. The total number of I/O patterns generated by the 200 word list was 1114. In the original experiments, the "best guess" error criterion was used. This refers to a phoneme making the smallest angle with the "desired output" vector. Such a criteria is not suitable for our purpose. Also, nets trained on the other benchmarks were not evaluated on the basis of such "angle proximity". Hence we used the same criteria employed for the other benchmarks: every output had to be of the correct logical level. For this problem, an output value was deemed correct if it was within 0.4 of the target. ($+0.1$ to $+0.5$ was considered logical "**1**" while -0.1 to -0.5 was considered logical "**0**"). Note that this is a much stricter criterion. The Cascade-Correlation learning algorithm was used to train the net. It typically uses about 75 units, and roughly 25000 independently adjustable parameters (weights and biases).

For each of these problems, many nets were generated using 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 non uniform distribution of computational load. Few dominant weights are fault tolerance bottlenecks. This happens even if the fan–in is very large. In the Sonar benchmark, for example, the fan–in of the output unit was 63 in some nets while in the NETtalk benchmark, 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 5) to achieve complete fault tolerance. It appears that the theoretical lower bounds derived above are almost impossible to realize in practice. This clearly demonstrates the need to modify the learning procedures to improve the fault tolerance.

VI 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 ANNs. Unlike digital logic, ANNs 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 tasks might not be as rigidly defined as that of a digital system. Partial fault tolerance is therefore more pertinent for ANNs. 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 problems mentioned above. Results for the $3-2-3$ Encoding problem and the first three benchmarks (all except NETTalk) are illustrated in Figures 4 to 7.

(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. Nets trained on all except the NETTalk benchmark happen to be manageably large. An exhaustive test strategy was therefore used to generate the plots in Figures 4 to 7.

To generate these figures, 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 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 × 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 4 to 7. The partial fault tolerance was calculated for increasing number of replications until it reached a satisfactory level (usually 95 % or higher).

Figure 4 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 5 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 6 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 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. 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 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. Actual numerical data showed that complete fault tolerance is *not* achieved even at $x = 6$, i.e., 6 extra replications.

Figure 7 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. This is a straight binary encoding of the outputs. Once again the plots show that the initial partial fault tolerance is excellent: over 90 % of all faults are tolerated without any additional redundancy. Also, the first replication yields maximum enhancement in partial fault tolerance.

It should be noted that a TMR scheme with majority voter achieves complete fault tolerance with much less redundancy (all the plots would reach the value $y = 1$ at $x = 2$, i.e., total number of modules $= 2 + 1 = 3$).

(B) Random Testing : The number of weights in the NETTalk benchmark is too large to permit exhaustive testing. We therefore adopt a more efficient testing strategy described below. 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 ANNs 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 for testing ANNs will have to be devised as ANNs mature from the experimental stage into commercial products.

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 in order to rapidly cover most faults. We extend this idea to the testing of ANNs. 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 the unit. Hence, it is more likely to cause malfunction than a disconnection of a single link. For these reasons, the bias faults should be tested exhaustively.

We applied this method to the spiral, sonar and vowel benchmarks and compared the results with the exhaustive testing scheme. The results are shown in Figures 8 and 9. Figure 8 shows the estimates of fraction of faults tolerated for the two spirals benchmark. The net had approximately 250 independent parameters, including 20 biases. The x coordinates represent 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 due to the different random seed. 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. Similar results were obtained for the vowel benchmark and are therefore omitted.

Figure 9 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 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 plot is shown in Figure 10. 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 good degree of partial fault tolerance to begin with, but a large number of replications are needed for complete fault tolerance, despite the large fan–in.

It is seen that the initial partial fault tolerance of ANNs is very good. Furthermore, the initial partial fault tolerance seems to improve with the net size, i.e. larger nets tolerate a greater fraction of all possible single faults without any additional redundancy. Also, the first replication yields greater improvement in fault tolerance than later, successive replications. This is especially true of the larger, complex benchmark nets. This is a very fortuitous situation. Had it been the other way around, i.e., larger nets were to have smaller partial fault tolerance and the estimates of partial fault tolerance generated by random testing were to deteriorate with net size, a more elaborate strategy would be required. This fact is important and can help decide whether the 3 fold redundancy required by a TMR scheme is worthwhile or the excellent initial partial fault tolerance is sufficient for the intended application. If the 3 fold redundancy is not affordable, the data suggests that one extra replication may be the best compromise, since later successive replications yield smaller enhancements in performance.

VI Conclusion

A method was proposed to estimate fault tolerance of feedforward ANNs and the redundancy required to achieve it. Fault models appropriate for hardware implementations were presented. A procedure was developed to build fault tolerant ANNs 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 irrespective 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 in order 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. The theoretical lower bounds are almost impossible to realize in practice, unless the nets are hand crafted for maximum fault tolerance. 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 ANNs. The symmetric version of the activation function yields higher fault tolerance in addition to faster convergence and utilizes less independent parameters in most cases.

Even though the amount of redundancy needed for complete fault tolerance is prohibitive, the data illustrates that ANNs 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 to later, successive replications.

It is evident that efficient testing strategies must be devised to test ANNs 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 non-uniform 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 in order to develop the specific weights and biases that optimize fault tolerance.

VIII Appendix

Proof of Theorem 3 : This result can be proved using a geometrical construction. The topology of the net for the $2-1-2$ problem is illustrated in Figure 2–b. Without loss of generality, the input patterns are assumed to be {0,1} and {1,0}, respectively. The same patterns should be reproduced at the output. The output of the hidden unit is in $(0,1)$ and can be represented by a point along a line. Choose that line to be the x axis. Then, the outputs of the hidden unit corresponding to each of the 2 input patterns are points between (0,1) on the x axis. This is illustrated by points P_1 and P_2 in Figure 11. Here, the hidden unit must have 2 distinct outputs, one corresponding to each of the input patterns. If not, the output units can not distinguish between the patterns. Denote the two distinct outputs of the hidden unit (corresponding to the 2 input patterns) as o_1 and o_2, respectively, where $o_1 < o_2$. Let the weights from the hidden unit to the output units be w_1 and w_2 and their biases be b_1 and b_2, respectively, as shown in Figure 2-b. Output units 1 and 2 are *on* or *off* when

$$f_i(x) = w_i x - b_i > 0 \quad or \quad f_i(x) = w_i x - b_i < 0 \quad , respectively. \tag{31}$$

In the above equation, $i = 1, 2,$ x is the output of the hidden unit and assumes one of the two values : $x = o_1, o_2$, and f_i represents the *resultant_input* to unit i. The weight and bias of each output unit define a line of the form (31), as illustrated by the lines l_1

and l_2 in Figure 11. These lines correspond to the separation surface implemented by the corresponding output units. On one side of the line, the *resultant_input* (i.e., f_i) is positive and on the other side, it is negative as illustrated by line l_1 in Figure 11. Hence, the reference to a line implies a reference to the corresponding unit. Henceforth, we just use the labels 1 and 2 to refer to the output units as well as the corresponding lines. Similarly, the point labels P_1 and P_2 are also used to refer to input patterns 1 and 2, respectively.

Learning implies finding weights and biases that satisfy the constraints

$$f_1(o_1) > 0 \;\; ; \;\; f_1(o_2) < 0 \;\; ; \;\; f_2(o_1) < 0 \;\; ; \;\; f_2(o_2) > 0 \tag{32}$$

The first two inequalities say that points P_1 and P_2 must be on positive and negative sides of line l_1, because unit 1 should be *on* for pattern 1 and *off* for pattern 2. The interpretation of the last two inequalities is similar. Together, the constraints imply that both lines l_1 and l_2 intersect the x axis between P_1 and P_2 and that one of them has a positive slope and the other has a negative slope. Figure 11 illustrates a case where the points P_1, P_2 and lines l_1, l_2 satisfy the above constraints.

In this representation, the vertical distance P_1B between P_1 and the line l_1 represents the *resultant_input* to output unit 1 for pattern P_1. Similarly, distance P_1A represents the *resultant_input* to unit 2 for pattern P_1. It is useful to think of directed distance from the points P_1, P_2 to lines l_1, l_2. If the direction is upwards, then the corresponding *resultant_input* is positive (i.e., the output of the unit is "**1**"), while a downwards distance implies a negative *resultant_input* ("**0**" output).

For the purpose of further analysis, we only need to concentrate on line l_1 that is represented by

$$y = w_1 x - b_1 \quad ; \quad w_1 > 0 \quad ; \quad b_1 > 0 \tag{33}$$

If a fault changes the value of the weight from w_1 to $-w_1$, the new line (hyperplane) is defined by

$$y = -w_1 x - b_1 \quad \text{which is labeled as} \quad l_3 \quad \text{in the figure.} \tag{34}$$

Due to this fault the resultant input to unit 1 for input pattern P_2 has changed its polarity (sign) and now has the negative value $-w_1 o_2 - b_1$ instead of the old value $w_1 o_2 - b_1$ which is positive. We add n additional hidden units and expect that the correcting influence of these other units (groups) will restore the polarity of the resultant input to its correct value. For this to be possible,

$$\frac{distance[EP_2]}{distance[CP_2]} < n \quad \text{or algebraically} \quad \frac{|-w_1o_2 - b_1|}{w_1o_2 - b_1} < n \quad or \quad \frac{w_1}{b_1} > \frac{n+1}{(n-1)o_2} \tag{35}$$

Next, consider a fault in the first layer which is made of the weights between the input units and the hidden unit. Let it change the value of one of the weights feeding the hidden unit and make it equal to the weight of the other link. In Figure 2-b, this corresponds to changing the weight value $-w_3$ in the first layer to w_4. This causes point P_1 to coincide with point P_2. The resultant input (to unit 1) corresponding to pattern P_1 is now $w_1o_2 - b_1$ instead of the correct value $w_1o_1 - b_1$. It has thus changed the polarity (sign) from negative to positive. If the n additional inputs are to be able to restore the polarity, the following inequality has to be satisfied.

$$\frac{distance[CP_2]}{distance[BP_1]} < n \quad \text{or algebraically} \quad \frac{w_1o_2 - b_1}{|w_1o_1 - b_1|} < n \quad or \quad \frac{w_1}{b_1} < \frac{n+1}{o_2 + no_1} \tag{36}$$

Equations (35) and (36) can be satisfied only if

$$\frac{n+1}{(n-1)o_2} < \frac{n+1}{o_2 + no_1} \quad \text{or} \quad n > \frac{2o_2}{o_2 - o_1} = 2 + \frac{2o_1}{o_2 - o1} \tag{37}$$

We are interested in the smallest integer n for which the above inequality holds. This happens when o_2 is maximized and o_1 is minimized. In the above equation, $o_1, o_2 \in (0,1)$ and can only asymptotically approach 0 and 1, respectively, as the argument of the exponential in (1) approaches $\pm\infty$. For any real implementation, then, the minimum value of n has to be 3. This implies that the total number of groups must be 4 or more. **Q.E.D.**

For good error tolerance, it is desirable to have o_1 and o_2 as far apart as possible. This is true because for any set of weights and biases in the last layer, the vertical distances to the lines from points P_1, P_2 can be maximized by pushing those points as far as possible. The above equations state that fault tolerance is also maximized by maximizing $o_2 - o_1$.

Proof of Corollary 1 :
In this case the outputs are bipolar and $o_1, o_2 \in (-1,1)$. By setting $o_1 = -o_2$ in equation (37), we see that the equation yields $n > 1$. If $n = 1$, the original constraints (35) and (36) can be satisfied only in a a limiting sense. For any real implementation, n must be at least 2 and hence the total number of groups needed must be at least 3. **Q.E.D.**

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] Y. Le Cun, J. S. Denker and S. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems 2* (D. S. Touretzsky, ed.), pp. 598–605, Morgan Kaufman Publishers, 1990.

[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 *Proceedings of Southeastcon*, pp. 328–331, IEEE Computer Society Press, 1989.

[5] T. Petsche and B. W. Dickinson, "Trellis Codes, Receptive Fields, and Fault Tolerant, Self-Repairing Neural Networks," *IEEE Transactions on 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," *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pp. 455–459, Jan 1990.

[7] 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.

[8] L. A. Belfore II and B. W. Johnson, "Analysis of the Faulty Behavior of Synchronous Neural Networks," *IEEE Transactions on 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 *Proceedings of International Conference on Acoustics Speech and Signal Processing (ICASSP), Toronto, Canada*, vol. II, pp. 1053 – 1056, May 1991.

[10] C. H. Sequin and R. D. Clay, "Fault Tolerance in Artificial Neural Networks," in *Proc. of Int. Joint Conf. on Neural Nets (IJCNN), San Diego, CA*, vol. I, pp. I–703 – I–708, Jun. 1990.

[11] 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.

[12] R. D. Clay and C. H. Sequin, "Fault Tolerance Training Improves Generalization and Robustness," in *Proc. of Int. Joint Conf. on Neural Nets (IJCNN), Baltimore, MD*, vol. I, pp. I–769 – I–774, Jun. 1992.

[13] B. E. Segee and M. J. Carter., "Comparative Fault Tolerance of Parallel Distributed Processing Networks (Debunking the Myth of Inherent Fault Tolerance)," Tech. Rep. ECE.IS.92.07, Intelligent Structures Group, Robotics Laboratory, Electrical and Computer Engineering Department, University of New Hampshire, Durham, Feb 1992.

[14] D. S. Phatak and I. Koren, "Fault Tolerance of Feedforward Neural Nets for Classification Tasks," in *Proceedings of International Joint Conference on Neural Nets (IJCNN), Baltimore, MD*, vol. II, pp. II–386 – II–391, Jun. 1992.

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

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

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

[18] Y. le Cun, "A theoretical framework for back-propagation," in *Proceedings of the 1988 Connectionist Models Summer School, San Mateo, CA* (D. Touretzsky, G. Hinton, and T. Sejnowski, ed.), pp. 21–28, Morgan Kaufman Publishers, 1988.

[19] S. E. Fahlman, "Faster Learning Variations on Back Propagation: An Empirical Study," in *Proceedings of the 1988 Connectionist Models Summer School, San Mateo, CA* (D. Touretzsky, G. Hinton, and T. Sejnowski, ed.), Morgan Kaufman Publishers, 1988.

[20] 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.

[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 *Neural Information Processing Systems 2.* (D. S. Touretzsky, ed.), pp. 650–657, Morgan Kaufman, 1990.

[23] 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.

[24] 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 University.

[25] K. J. Lang and M. J. Witbrock, "Learning to Tell Two Spirals Apart," in *Proceedings of the 1988 Connectionist Models Summer School, San Mateo, CA* (D. Touretzsky, G. Hinton, and T. Sejnowski, ed.), Morgan Kaufman Publishers, 1988.

[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] A. J. Robinson and F. Fallside, "A Dynamic Connectionist Model for Phoneme Recognition," in *Proc. of nEuro, June 1988, Paris*, Jun. 1988.

[28] T. J. Sejnowski and C. R. Rosenberg, "Parallel Networks That Learn to Pronounce English Text," *Complex Systems*, vol. 1, pp. 145–168, 1987.

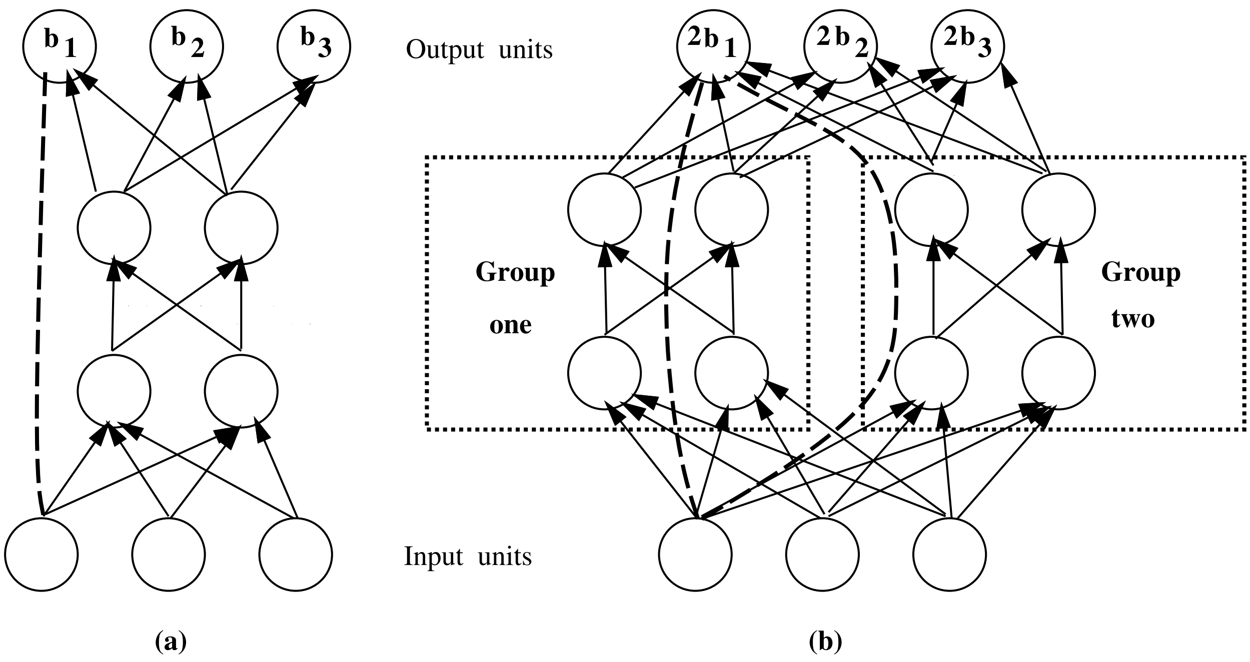

Figure 1 : Replication of Hidden Units.

(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

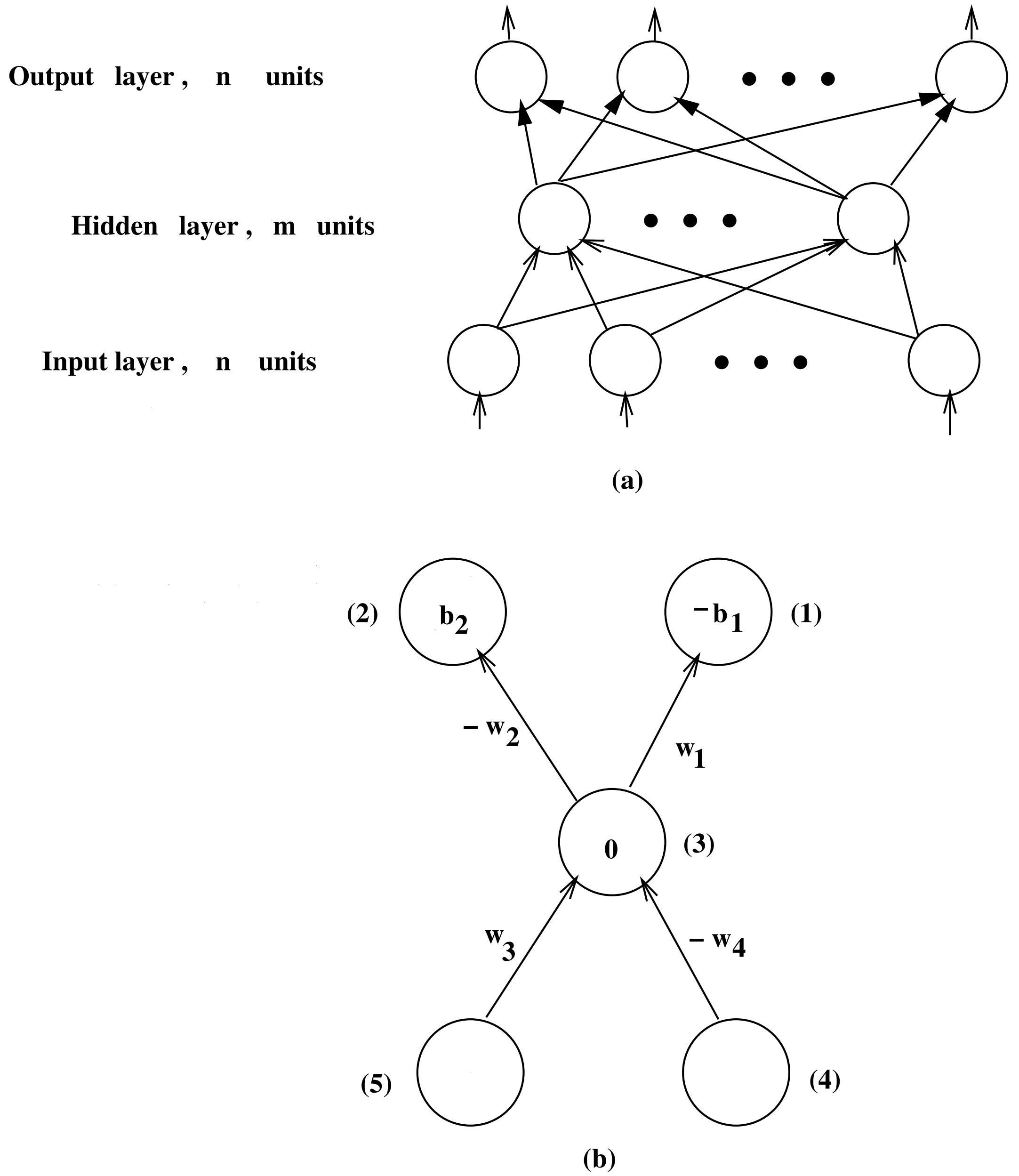

Figure 2 : Encoding nets.

(a) A two–layer $n - m - n$ encoding net
(b) A 2–1–2 net with weights and biases
w_i and $b_i > 0$ for all i. Unit indices are shown in parenthesis.

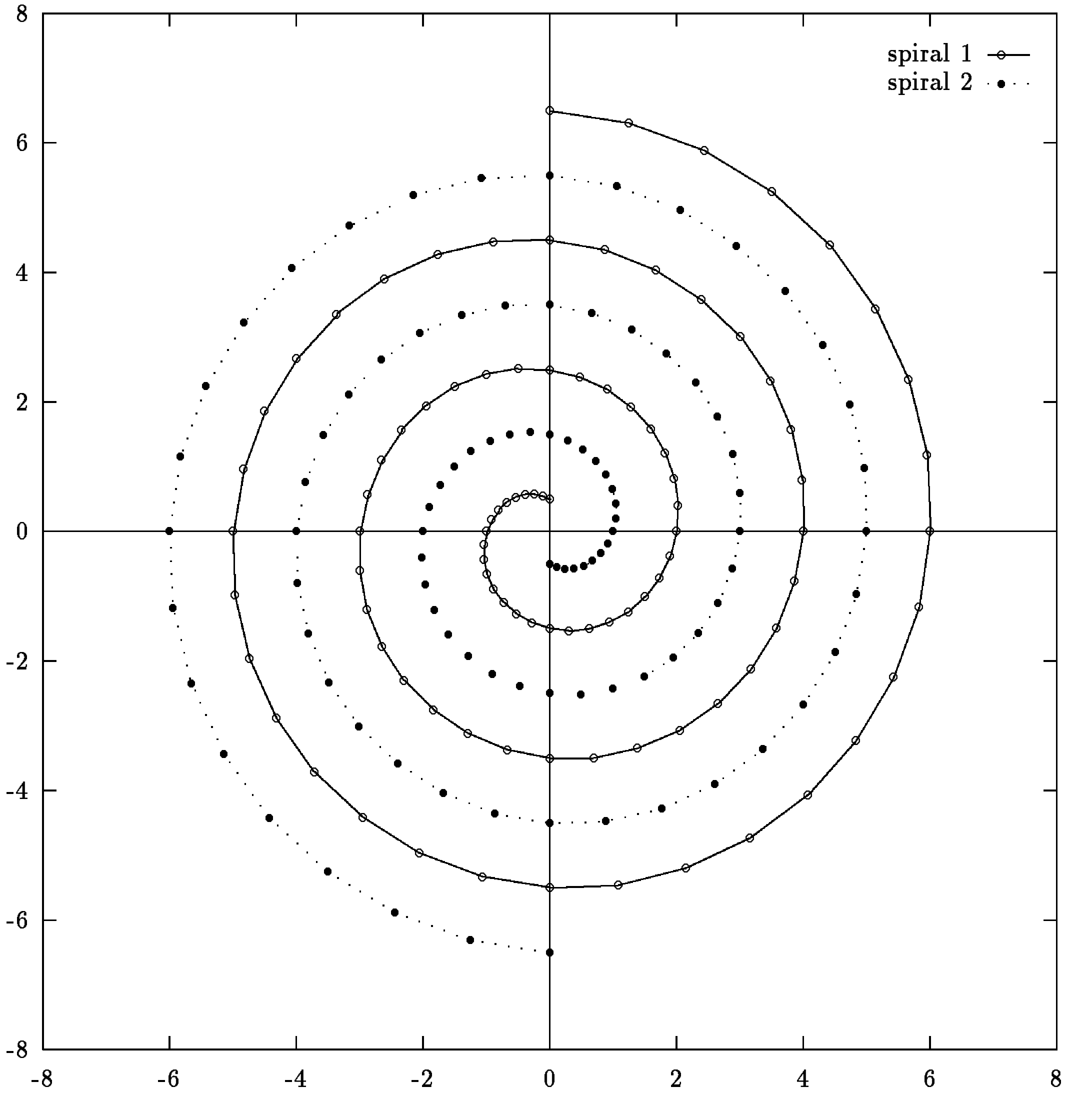

Figure 3 : The Two Spirals Benchmark Data [24].

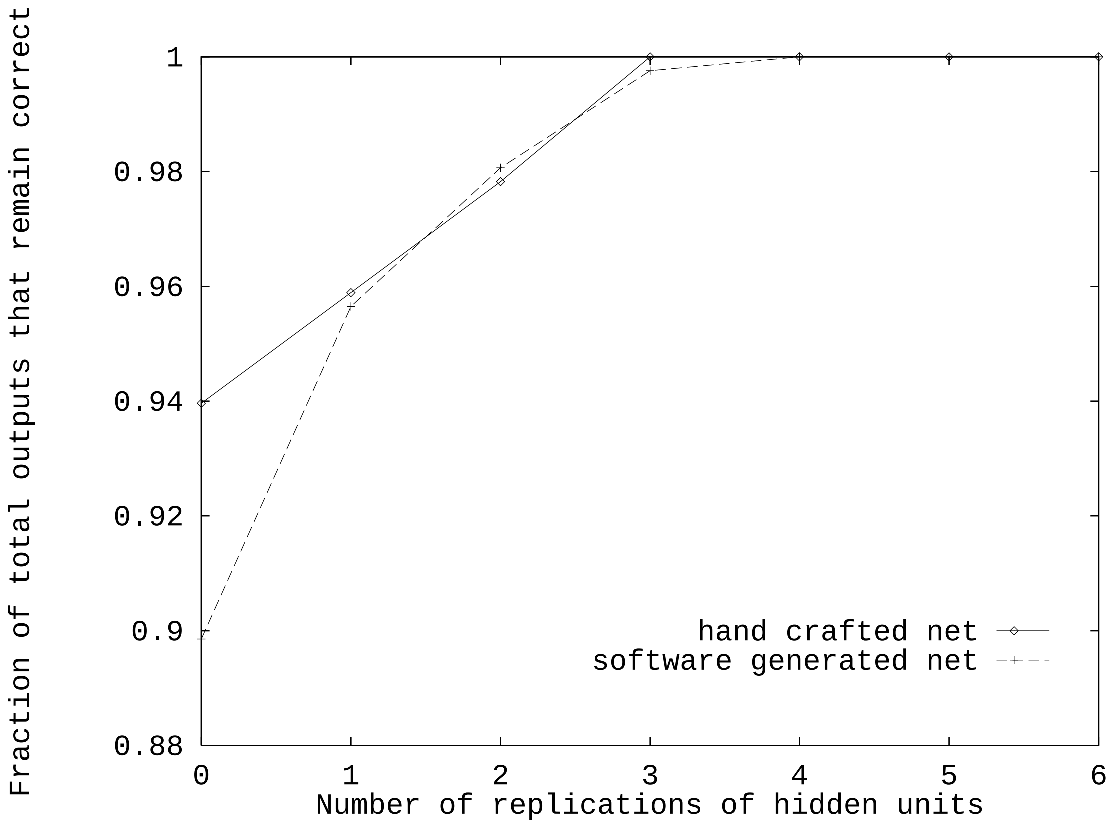

Figure 4 : Comparison of partial fault tolerance of 3-2-3 encoding nets. A hand crafted net and the best (out of 100) back propagation generated net.

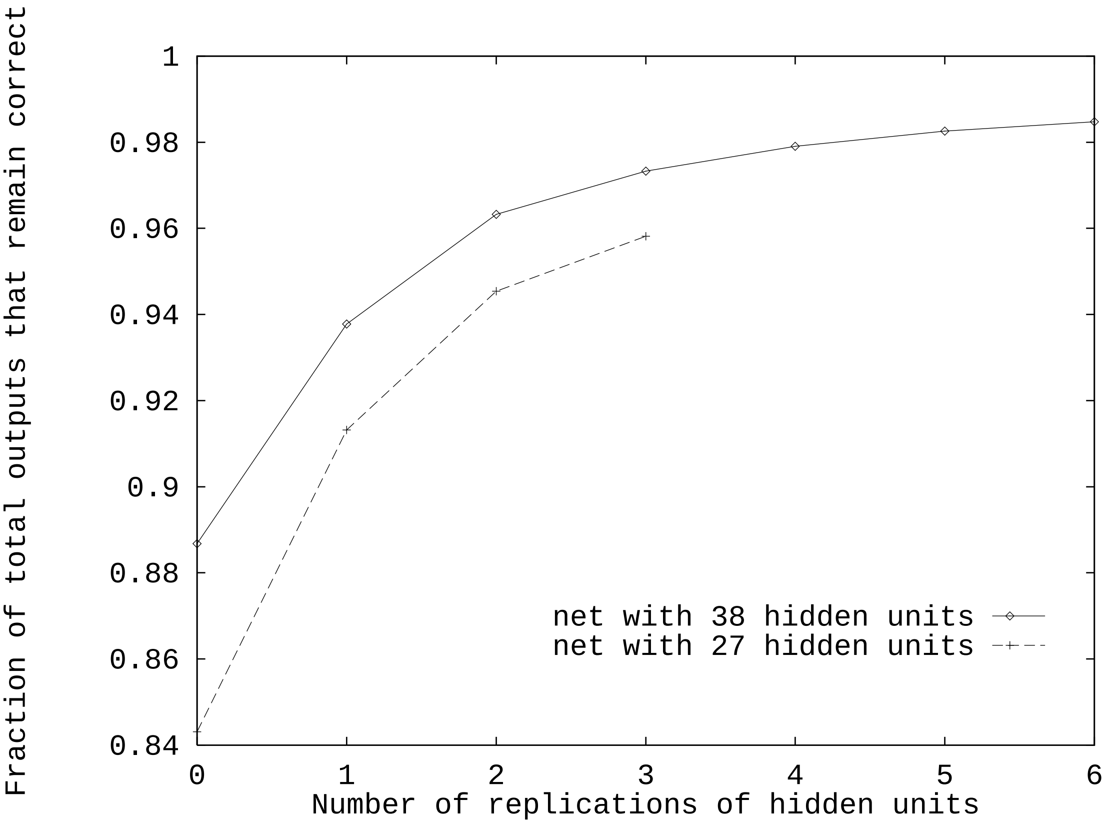

Figure 5 : Partial fault tolerance of nets trained on the **Two Spirals Benchmark** (by the Cascade Correlation algorithm.)
(a) Net with 38 hidden units. (b) Net with 27 hidden units.

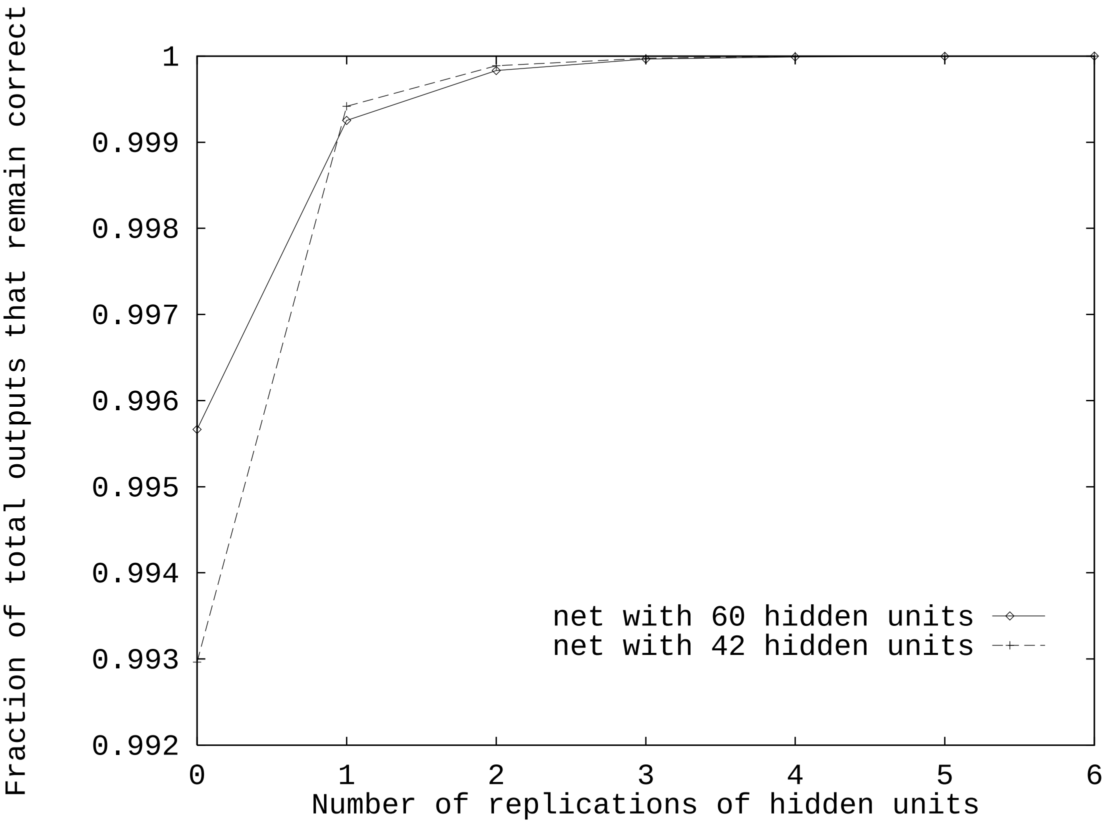

Figure 6 : Partial fault tolerance of nets trained on the **Sonar Benchmark** by Back Propagation. (a) Net with 60 hidden units. (b) Net with 42 hidden units.

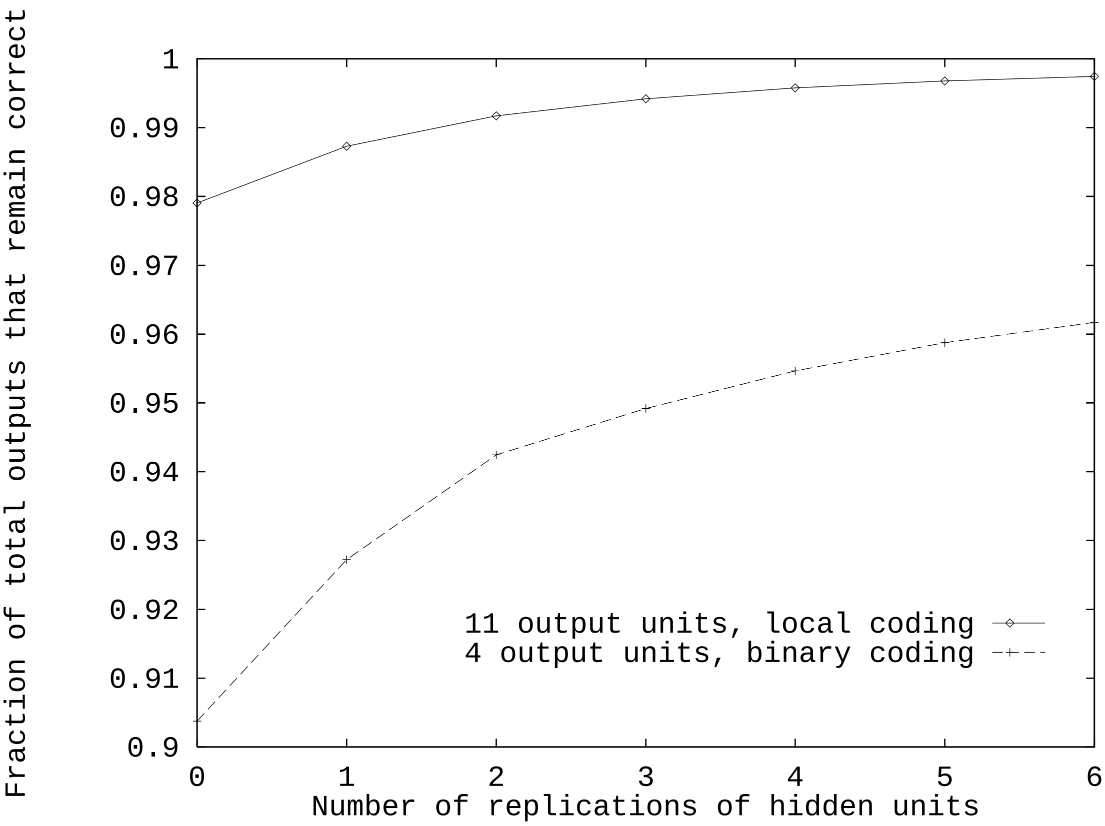

Figure 7 : Partial fault tolerance of nets trained on the **Vowel Benchmark** by the Cascade Correlation algorithm. (a) Net with 1 hidden and 11 output units. One output is "on" for each of the 11 vowels (localized encoding). (b) Net with 3 hidden and 4 output units which produce 4 bit binary numbers from 0 to 10 to indicate the vowel.

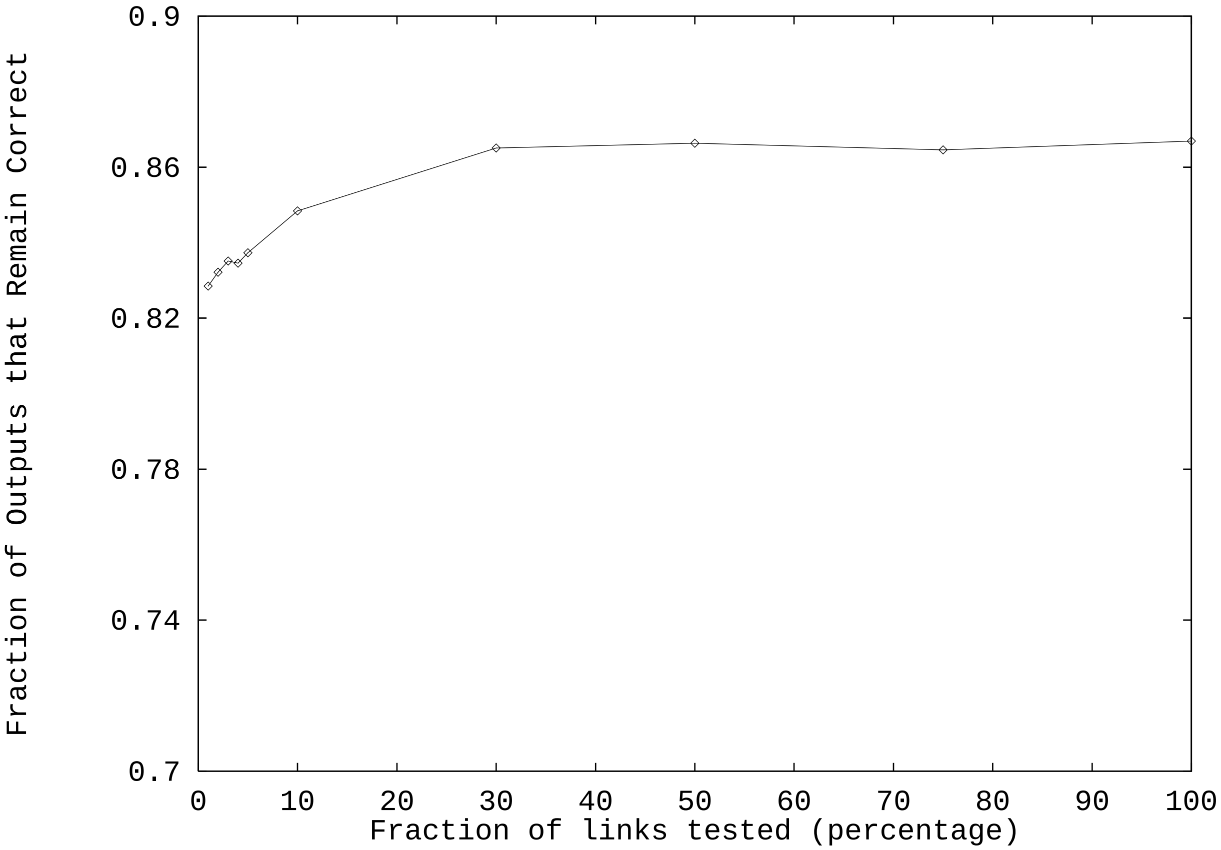

Figure 8 : Comparison of random link testing method with exhaustive testing for a typical net trained on the Two Spirals benchmark by the Cascade Correlation algorithm. All biases are exhaustively tested. A fraction of the total number of links are then chosen at random for weight fault testing. This fraction (the x coordinate) is varied from 1 % to 100 % (exhaustive testing).

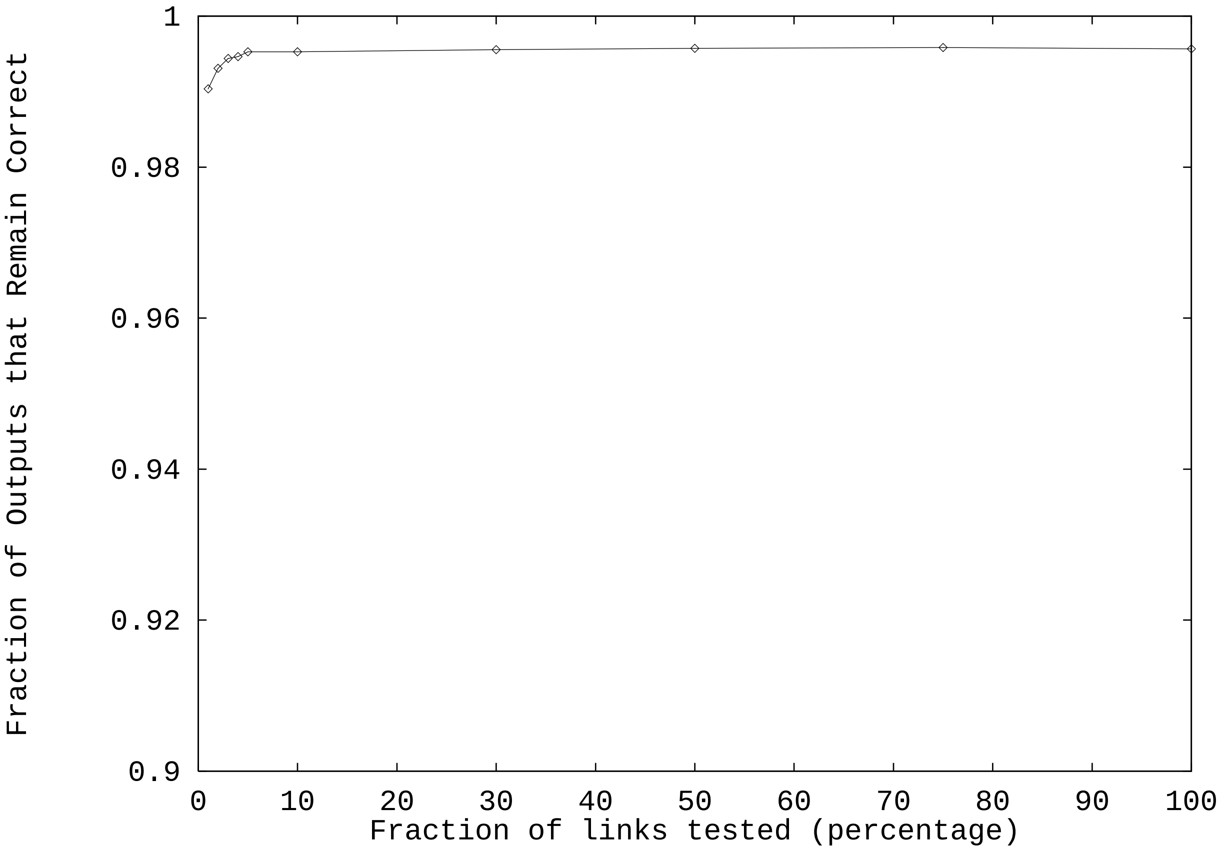

Figure 9 : Comparison of random link testing method with exhaustive testing for a typical net trained on the Sonar benchmark by the Back Propagation algorithm. The point corresponding to 100 % on the x axis is the result generated by exhaustive testing.

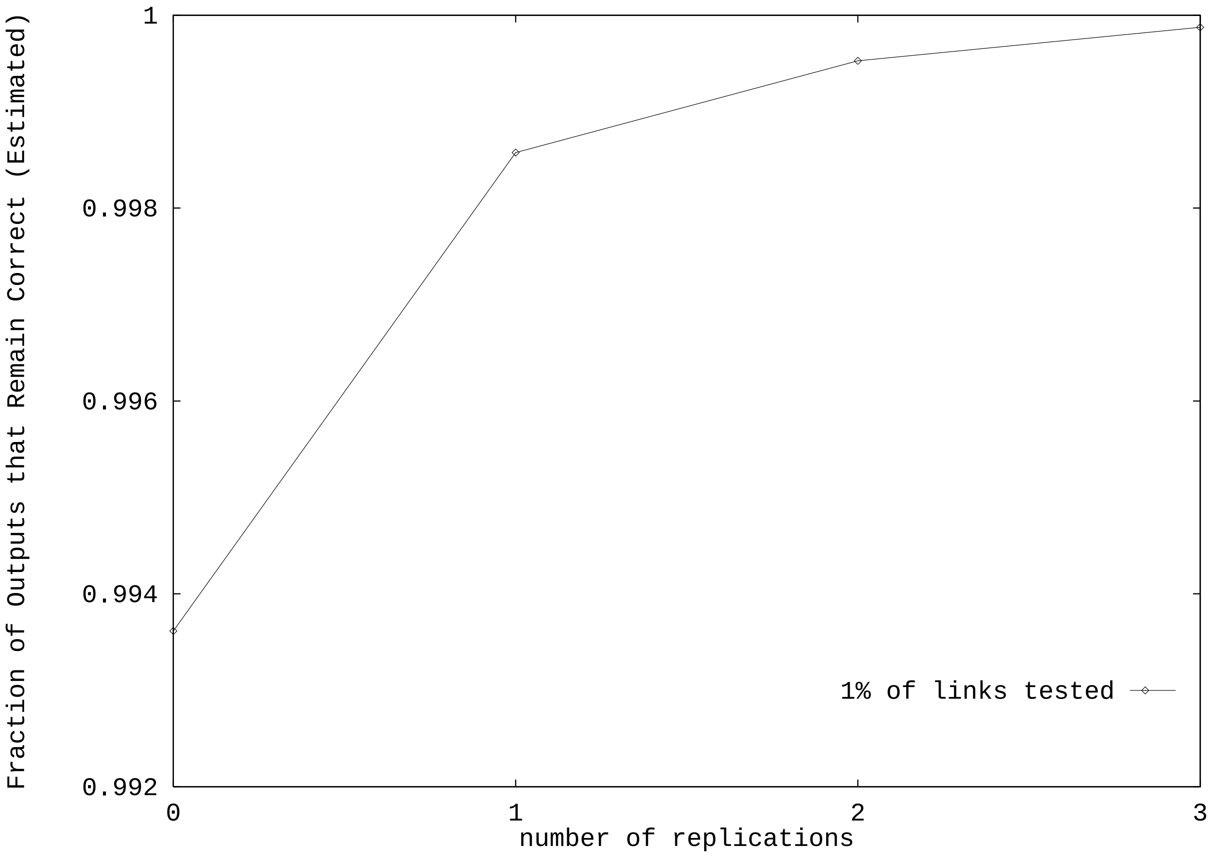

Figure 10 : Partial fault tolerance of a typical net with Cascade Correlation on the NETTalk benchmark. Training set consisted of 200 words which generated 1114 I/O patterns. The net had 24622 independent parameters (weights and biases) including 75 biases. All bias faults were exhaustively tested. 1 % of the links were then randomly tested for weight faults.

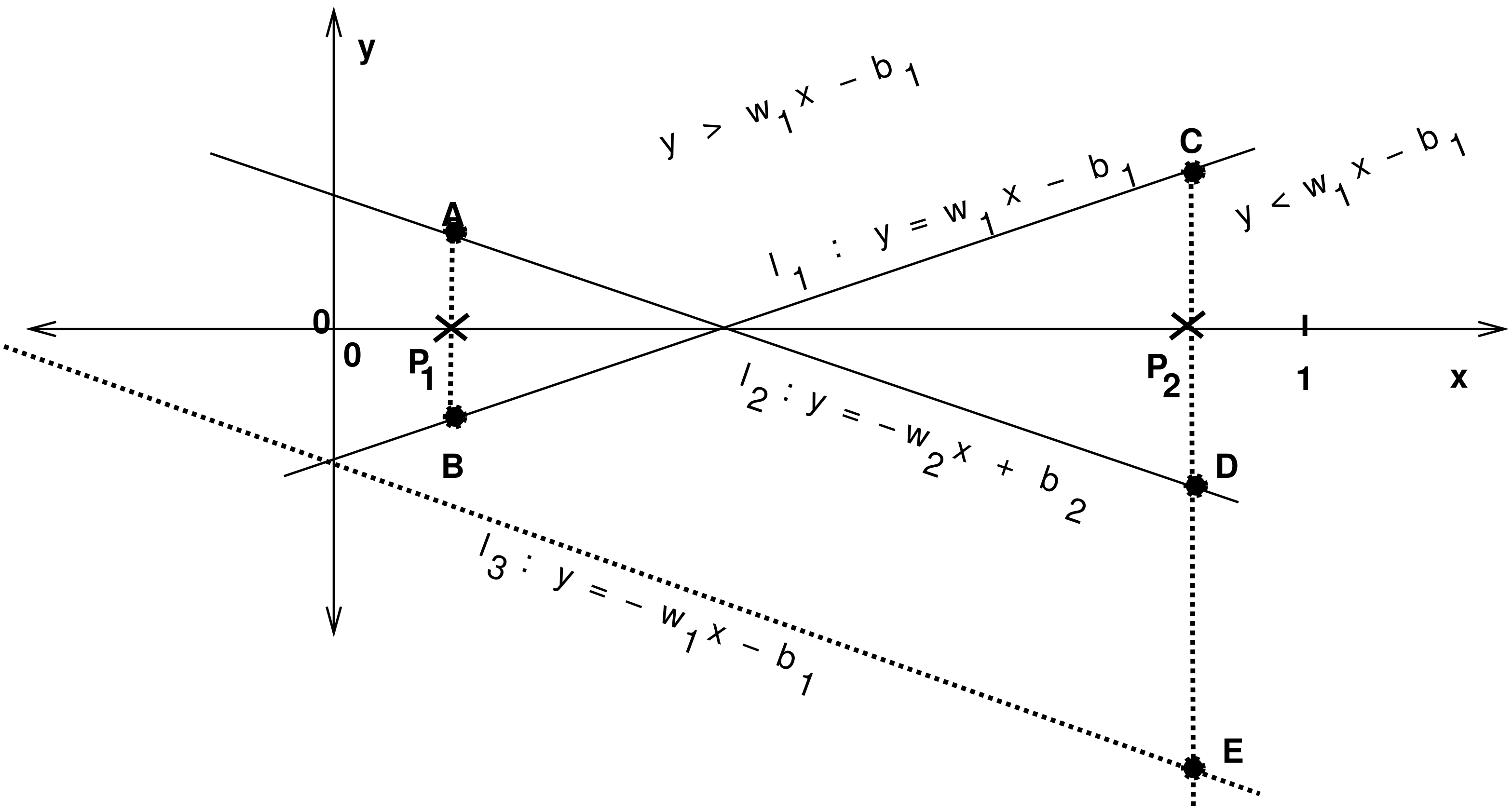

Figure 11 : A geometrical interpretation of the 2–1–2 encoding problem.
please refer to the appendix for details.