Fault and Failure Tolerance

Jack D. Cowan
Departments of Mathematics and Neurology,
The University of Chicago
Chicago, IL 60637

July 28, 1995

1 Introduction and Early Work

One of the many remarkable properties of the brain is the degree to which it is fault and failure tolerant. In many cases even the loss of substantial amounts of brain cells or tissue does not totally abolish brain function, a property known as “graceful degradation”. It is therefore not surprising that the problems of constructing fault and failure tolerant neural networks have been studied almost since the earliest days of Neural Networks. Von Neumann’s 1952 CalTech lectures entitled “Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components” (von Neumann 1956), set the stage for the early investigations. Von Neumann used redundancy techniques derived from coding theory to provide error detection and correction in neural networks composed of components which malfunctioned from time to time with probability $\epsilon$. By using $N$ components everywhere instead of 1, and the bundle encoding scheme in which activation of more than $(1 - \Delta)N$ lines signals ‘1’ and less than $\Delta N$ signals ‘0’, together with ‘restoring organs’ which mapped bundle activity levels towards either ‘0’ or ‘1’, he was able to achieve arbitrary reliable computing.
More precisely, let $\rho(N)$ be the probability of network malfunction, $\epsilon = 0.005$ and $\Delta = 0.07$. Then

$$\rho(N) \sim \frac{6.4}{\sqrt{N}} 10^{-0.0086N}$$

(1)

This result is somewhat disappointing in that $\rho(N) \leq \epsilon$ if and only if $N \geq 2000$. Thus high levels of redundancy are required to obtained low $\rho(N)$. However for $N \geq 20,000$ the result is insensitive to variations in $\epsilon$ and $\Delta$, provided they are small enough.

Von Neumann’s result is not mathematically very rigorous: the action of the restoring organ depends upon the use of random permutations of bundle lines, and it is not clear how to achieve such permutations [Pippenger 1990]. Recent work along these lines has produced the following rigorous results. Consider the sum (modulo 2) of a function of $n$ arguments computed by a network of $O(n)$ reliable gates. Such a function can be computed by not less than $o(n \log n)$ unreliable gates, as can the $n$-argument functions AND and OR [Dobrushin and Ortyukov 1977]. Similarly, consider the parallel computation of a set of $k$ functions, each of which is the sum (modulo 2) of some of their $n$ arguments. Pippenger [1985] showed that for $n = k$ the minimum possible number of unreliable components required to compute reliably such a set is $O(n^2 / \log n)$. In all these cases then, the ratio of the numbers of reliable to unreliable gates required to achieve reliability varies from $O(1/N)$ in von Neumann’s construction, to at best $O(\log N / N)$ in Pippenger’s.

All these results are at variance with what Information Theory tells us about

---

1We use the ‘big-Ω’ and ‘little-ο’ notation of asymptotic analysis, $f = \Omega(g)$ means $\frac{f}{g}$ is bounded, whereas $f = o(g)$ means $\frac{f}{g} \to 0$ in some limit.
the use of coding techniques for reliable transmission of messages through noisy channels. As is well-known Shannon showed that the noise in a channel determines a maximum information rate $R$ (in bits of information per coded message symbol) at which information in the form of coded messages may be transmitted and decoded without error at the receiver. This maximum rate is known as the channel capacity $C$ and is a function of the noise power in the channel. Thus if the channel noise is additive Gaussian, $C = \log_2 \sqrt{1 + 2E_s/N_o}$ where $E_s$ is the coded message energy per bit, and $N_o$ is the noise energy per bit. Shannon’s famous noisy channel coding theorem then says that coded messages may be transmitted through such a channel and decoded without error if and only if $R \leq C$. In the additive Gaussian case, if $E_s \ll N_o$, we have the result that $R \leq C = (1/\ln 2)E_s/N_o$ for reliable transmission. Moreover in such a case Shannon showed that the probability of error in decoding, which we may call $\rho(N)$, is $\sim O(2^{-N(C-R/R)}) = O(\exp(-\alpha N))$, where $\alpha = (C - R)/\log_2 e$ and where $N$ is the length of the code words sent through the channel.

It is clear that there is a big difference between a ‘computation rate’ which is at best $O(\log N/N)$, and a transmission rate $R$ which is independent of $N$. It was this situation that was investigated by Winograd and Cowan (1963) in their monograph ‘Reliable Computation in the Presence of Noise’. They noted that the main reason for the difference lies in the nature of the decoding process. In a communication system, only the channel is assumed to be noisy, and the various computations performed by the decoder are assumed to be error-free. This is not necessarily the case in performing computation with unreliable elements, in which decoding operations cannot be assumed to be error-free, since
they are performed by the same elements. In fact it was proved by Winograd (1963) that if \( t \) errors are to be corrected anywhere in a network, then the average number of inputs per device in the network, \( n \), is at least \((2t + 1)R\) where \( R \) is the computation rate, i.e. the ratio of the numbers of reliable to unreliable elements required to achieve reliability, as discussed above. Conversely, we may say that \( R \leq n/2t + 1 \), so that if the computation rate per element is required to remain constant as more and more errors are corrected, then the complexity of the elements, as measured by \( n \), must increase. Here is the crux of the problem, for if one is provided with elements which compute only functions of \( m \leq n \) arguments, then it can be shown that \( p = O(2^{n-m}) \) elements are required to compute each function of \( n \) arguments (Cowan & Winograd, unpublished). Evidently \( \epsilon \), the probability of error per element, increases exponentially with \( p \) so that the Shannon bound cannot be saturated in such a network. The only possibility obtains if \( \epsilon \) is independent of \( p \). This would be the case, for example, if \( m = n \), and if \( \epsilon \) does not then increase exponentially with \( n \). Such considerations led Winograd and Cowan to introduce a scheme for the reliable parallel computation of a set of \( k \) functions, each of which is an arbitrary Boolean function of some of their \( n \) arguments, that incorporated an \((N,k)\) error-correcting code into the set of computations. Such a code comprises a scheme in which messages \( k \)-digits long are encoded into signals \( N \)-symbols long for transmission through a noisy channel. In a conventional Hamming block code, \( N - k \) of the received symbols in each codeword are used for error detection and correction. The associated information transmission rate is \( k/N \). In the computing case this implies that \( k \) computations be implemented by an
network comprising $N/k$ times as many elements as would be required in the
error-free case. Fig. 2 shows an example of the application of a $(5, 2)$ Hamming
code to the network shown in Fig. 1. The first layer of elements combines the logical functions $x_1 \text{AND} x_2$ and $\text{NOT}(x_3 \text{AND} x_4)$ with the encoding function of the

![Threshold Logic Network](image)

Figure 1: A threshold logic network that implements the logical function $\text{NOT}[(x_1 \text{AND} x_2) \text{OR ELSE} \text{NOT}(x_3 \text{AND} x_4) \text{AND} x_5] \text{OR} x_5$.

$(5,2)$ Hamming code. This takes the following form. Let $y_1$ and $y_2$ be symbols
to be transmitted (or functions to be computed). The $(5,2)$ encoding consists
in also transmitting (or computing) $y_3 = y_1 \text{ OR ELSE } y_2$, $y_4 = y_2$, $y_5 = y_1$. In
the case shown above both $y_1$ and $y_2$ correspond to the parallel pair of logical
functions $x_1 \text{ AND } x_2$, and $\text{NOT}(x_3 \text{ AND } x_4)$. Therefore the first layer elements
implement the logical functions $f_{11} = f_{15} = x_1 \text{ AND } x_2$, $f_{12} = f_{14} = x_1 \text{ AND } x_2$;
$f_{21} = f_{25} = \text{NOT} (x_3 \text{ AND } x_5)$, $f_{22} = f_{24} = \text{NOT}(x_2 \text{ AND } x_3)$;
$f_{13} = f_{11} \text{ OR ELSE } f_{12}$, $f_{23} = f_{21} \text{ OR ELSE } f_{22}$. The second layer elements
are even more complicated since they implement the combination of two copies of the logical function $z_1 \text{ OR ELSE} (z_2 \text{ AND } x_4)$, where $z_1 = x_1 \text{ AND } x_2$ and $z_2 = \text{NOT}(x_2 \text{ AND } x_3)$, with the decoding function of the (5,2) Hamming code (needed to detect and correct errors occurring in the first layer), and the encoding function again (needed for transmission to the third layer). Finally the third

Figure 2: Redundant MP network in which two parallel implementations of the logical function NOT[[x1 AND x2] OR ELSE (NOT[x2 AND x3] AND x4)] OR x5] are combined with a (5,2) Hamming code. (See text for details).

layer elements combine the decoding function with two copies of the final logical function NOT($z_3$ OR $x_5$), where $z_3 = z_1$ OR ELSE($z_2$ AND $x_4$). The decoding function of the (5,2) Hamming code is just a parity check sum, implemented via the logical functions $(y_1$ OR ELSE $(y_1$ OR ELSE $y_6$) AND $(y_1$ OR ELSE $y_2$ OR ELSE $y_3$) and $(y_2$ OR ELSE $(y_2$ OR ELSE $y_4$) AND $(y_1$ OR ELSE $y_2$
OR ELSE \( y \). Thus the third layer elements implement the logical functions
\[ \text{NOT}(f_{31} \text{ OR ELSE } (f_{31} \text{ OR ELSE } f_{33})) \text{ AND } (f_{31} \text{ OR ELSE } f_{33}) \] OR \( x_{51} \) and \( \text{NOT}(f_{32} \text{ OR ELSE } (f_{32} \text{ OR ELSE } f_{34})) \text{ AND } (f_{31} \text{ OR ELSE } f_{32} \text{ OR ELSE } f_{33}) \) OR \( x_{52} \), where \( f_{3i} \) (\( i = 1, 2, \ldots, 5 \)) are the second layer logical functions.

One can compute the efficiency of such a scheme. Suppose the function to be computed has \( M \) elements, of which a fraction \( \alpha = u/M \) are output elements. Then the number of elements in the \( k \) precursors is just \( Mk \), and the number in the redundant network is \( (M - u)N + uk \). Thus the computation rate is
\[
R = \frac{k}{N - \alpha(N - k)} > \frac{k}{N}.
\]

The overall effect of such an encoding scheme is to distribute the logical functions to be implemented over the entire network. Such a scheme works most efficiently with large \( k \) and \( N \); in effect in a parallel distributed architecture. Thus the Winograd–Cowan scheme is an early example of *Parallel Distributed Processing* or *PDP*. Of course it can be argued that the scheme is not realistic, in that all the extra coding machinery is assumed to be error-free. As we have noted this is not true for simple logical elements, but it may be more plausible for real neurons. In unpublished work Winograd and Cowan studied this issue in more detail and found a realistic optimal scheme that combines von Neumann’s bundling scheme with *PDP* (Cowan and Winograd, In preparation).

### 1.1 From McCulloch–Pitts Elements to Adalines and Sigmoids

In the decades since these early investigations, work on fault and failure tolerance has been strongly influenced by the shift from synchronous binary logic
networks to asynchronous analog networks, in which the weight patterns required to implement encoding and decoding functions such as those described above, can be learned, or indeed in which decoding is performed by the dynamics of a fully connected recurrent network. In what follows we describe some of these developments.

Before doing so, however, we describe some recent calculations of Stevenson, Winter and Widrow (1990) on the effects on $\epsilon$ of both input errors and weight perturbations on Adalines. The basic idea is straightforward, and indeed is common to any threshold device. Consider an Adaline with $n$ inputs where $n$ is large. Let $X$ and $W$ be $(n+1)$-dimensional binary input and weight vectors, and let $\Delta X$ and $\Delta W$ be variations in their amplitudes with Hamming distance $t$, i.e. $t$ of the $n+1$ vector components are inverted, and let $\delta X = \Delta X/|X|$ and $\delta W = \Delta W/|W|$. Using the geometric properties of $n$-dimensional vectors it is easy to show that the probability of an Adaline decision error $P$ is approximately $(1/\pi)[\delta X^2 + \delta W^2]$ for small errors. Numerical results are shown in Fig. 3. From this one can calculate the probability of a Madaline decision error, $P_L$, i.e. the output probability of error of a network comprising $L$ Adaline layers. In general this will be binomially distributed. Stevenson et al found the remarkably accurate approximation for $P_L$ in the case of weight errors:

$$P_L = (\delta W/\pi)(1 + \beta(1 + \beta(1 + \beta(1 + \beta)^{1/2})^{1/2})^{1/2})^{1/2}$$

(2)

where $\beta = 4/(\pi \delta W)$, the number of square roots is $L - 1$, and the number $n$ of Adalines per layer is large, i.e. at least 100. Results are shown in Fig. 4.

One can draw two conclusions from this formula, both of which can be seen in

---

\(2\) The extra dimension is needed for the threshold setting.
simulations. The first is that as long as \( n \) is large, \( P_L \) is essentially independent of \( n \), and the second is that \( P_1 \), the probability of error for a one-layer network of Adalines is also essentially independent of the number of inputs per Adaline. These results evidently apply to our previous discussion concerning the validity of the Winograd–Cowan encoding scheme. Interestingly Alippi, Fluri and Sami (1994, Snowbird preprint) have recently extended the Stevenson et al. analysis to analog threshold devices based on the sigmoid.

2  **Associative Memories and Sparse Distributed Coding**

What we have described so far, applies to the general problem of computation. Operationally this includes such functions as pattern classification, and as we have seen, decoding. There is however another function performed by neural
networks, which has been the subject of many investigations, namely, the storage and retrieval of information in the form of associative memories. Work on this over the past decade has focussed mainly on Hopfield Networks and their properties, but there is also an older generation of associative memories, the analysis of which provides insights into fault and failure tolerance.

Consider for example, the network introduced by Willshaw, Buneman & Longuet–Higgins (1969) shown in Figure 5, and the pattern of associations stored therein shown in Table 1. The network comprises $V$ $A$–lines, $R$ $B$–lines, and $V \cdot R$ switches. Let a typical $A$–pattern correspond to the activation of $n$ randomly chosen $A$–lines, and a typical $B$–pattern to $m$ randomly chosen $B$–lines. Thus there will be a total of $n \cdot m$ doubly activated switches. Let these be turned on, if they are not already on. After $K$ pairs of patterns have been associated in this way, a fraction $p$ of the switches will have been turned on.
Figure 5: An associative network with input lines ($A_1 \ldots A_8$) and output lines ($B_1 \ldots B_8$). Filled semicircles represent switches which are on, open semicircles those which are off. (See text for further details) [Redrawn from Willshaw et al. 1969].

Table 1: The pattern of associations stored in the above network

<table>
<thead>
<tr>
<th>A-pattern</th>
<th>B-pattern</th>
<th>A-pattern</th>
<th>B-pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,3</td>
<td>4,6,7</td>
<td>2,4,6</td>
<td>2,3,6</td>
</tr>
<tr>
<td>2,5,8</td>
<td>1,5,7</td>
<td>1,3,7</td>
<td>3,4,8</td>
</tr>
</tbody>
</table>

If the patterns are random, $p$ will be approximated by

$$ p = 1 - e^{-K(n-m)/(V-R)}.$$  \hspace{1cm} (3)

To recall a pattern is simple. An $A$-pattern is put in, so that each $B$-line of the associated $B$-pattern is activated through $n$ switches (which, by hypothesis, are all on). However $B$-lines not belonging to the associated $B$-pattern will also be activated. The probability that such a $B$-line is activated through every one of its $n$ intersections with the activated $A$-lines is just $p^n$. So if the firing
threshold of each $B$-line is $n$, not only will all those lines fire which comprise the associated $B$-pattern, but a further $m \cdot p^n$ will probably also fire. If this number is less than one, no errors will occur, so the critical value is

$$m \cdot p^n = 1. \quad (4)$$

Now a single $B$-pattern requires

$$\log_2 \left( \frac{m}{R} \right)$$

bits to specify it, so if $K$ $B$-patterns can be accurately retrieved, the amount of available stored information is

$$K \cdot \log_2 \left( \frac{m}{R} \right) \approx K m \log_2 R \quad (5)$$

bits. The combination of these three equations leads to an expression for the number of bits stored in the network, namely

$$V \cdot R \log_2 p \log_e (1 - p).$$

This reaches its maximum value of $0.693 \cdot V \cdot R$ when $p = 0.5$. In such a case $n \approx \log_2 R$, i.e. the number of lines per $A$-patterns should be small, if the network is to be an efficient information store. This is an early example of the argument leading to sparse coding. In a separate paper Willshaw & Longuet-Higgins (1970) showed that associative networks could be made to function accurately even if the $A$-patterns are noisy, and even if the network is damaged, by raising $n$, and therefore storing fewer associations. The resulting sharp drop in the information storage density is consistent with the Winograd-Cowan results described in §1.
2.1 Superimposed Random Coding

The above analysis indicates that the need to retrieve patterns without error places a sharp bound on the number of lines per \( A \)-pattern which can be used to encode information about items to be stored. If there are \( V \) input lines, this means that there are at most

\[
\binom{V}{n}
\]

different \( A \)-patterns or descriptors which can be fed into the store. Since \( n \) has to be small, the only way to increase the possible number of descriptors, i.e. the vocabulary, is to increase \( V \). Is this meaningful? Remarkably, a paper dealing with this issue, and with the retrieval process, in a manner closely paralleling the Willshaw & Longuet-Higgins analysis, had already appeared [Greene 1965].

Greene’s paper is particularly interesting in that, although it is not obvious from a first reading, it is virtually identical in conceptual content with David Marr’s well-known paper on the Cerebellum [Marr 1969] which itself is closely related to the Willshaw & Longuet-Higgins analysis, and with later works such as that by Albus (1975) and Kanerva (1988).

Here we give a brief account of Greene’s contribution. Greene based his analysis on earlier work by Mooers (1949) on the storage and retrieval of information using decks of punched cards. The method is well-known: holes are punched in a number of locations on each card in a deck. These holes (or their absence) embody Boolean descriptions \( x_1 \& \bar{x}_2 \& x_3 \& x_4 \cdots \bar{x}_N \) etc. where the \( x_i \) represent various categories: female, married, Caucasian, etc: and because of the holes, decks of such cards can be automatically sorted, either mechanically or electrically. Consider now the combinatorics of the setup. If there are \( N \)
locations, there are $2^N$ possible binary patterns to be used as descriptions. Let us now suppose that $D$ categories are allocated to each descriptor, with no overlap. Then there are $M = N/D$ possible subfields, and a vocabulary comprising $V = M \cdot 2^D$ independent descriptors. So if $D = N$, $V = 2^N$; if $D = N/2$, $V = 2N$; and if $D = 1$, $V = 2N$.

Mooers' contribution, which he patented under the name "Zatocoding", is to improve on these possibilities by using randomly overlapping or *superimposed* subfields as descriptors. This generates a vocabulary of

$$V = \binom{N}{D}$$

independent descriptors. Now suppose that on the average, there are $K$ descriptors per card. Then the maximum total number of descriptions is

$$\binom{V}{K}$$

i.e.; $K$ out of $V$ $D$-tuples. It can be shown that the probability $P$ of any of the $N$ locations being in one of the $K$ descriptors is bounded by

$$1 - (1 - \frac{D}{N})^K,$$

so that the total number of locations used to form the $K$ descriptors is

$$G \approx N \cdot P = N \cdot (1 - (1 - \frac{D}{N})^K) \leq N \cdot (1 - e^{-X/N})$$

where $X = K \cdot D$. This number is maximized when $X \approx N \cdot \ln 2$, at the value $G \approx N/2$, when $P \approx 1/2$. This is an important result, and in fact Mooers' (1949) demonstrated its connection with Information Theory: the maximum "capacity" is obtained when each location on a card can signal one bit of information, and
this occurs when about half the locations are used. This of course is similar to the conclusion reached later by Longuet–Higgins et al. by a slightly different argument.

So much for the encoding process. What about the decoding process: retrieving information from the card deck? Suppose there are $R$ cards, and suppose that, as before, the joint presence of $n$ descriptors is required to select a card. Then approximately $2^{nD}$ different cards can be uniquely decoded, i.e.; $R \approx 2^{nD}$ or

$$D \approx \frac{\log_2 R}{n}.$$ 

Furthermore the fraction of wrong cards selected does not exceed

$$\left( \frac{G}{n} \right) \left( \frac{N}{n} \right) < \left( \frac{G}{N} \right)^n$$

This can be made as small as desired by adjusting $G$, $N$ or $n$. In the optimal case when $G \approx N/2$, it is less than $2^{-n}$.

It follows that one should first choose $D$ given $R$, then set a value of $N$ large enough to generate enough descriptors, on the average $K$ per card. For example, suppose $n = 3$ and $R = 4000$. Then $D \approx \log_2 4000/n = 4$. Choose $N = 40$, then $X \approx 40 \cdot \ln 2 \approx 28$ whence $K = X/D = 7$. This demonstrates that superimposed random coding is advantageous when there are a large number of descriptors, each occurring with a relatively low frequency, i.e. another argument for sparse coding. It would not be advantageous in case one or two descriptors are present on every card, in addition to other descriptors. It would be more economical to use a 1-location fixed field in each card to indicate these two, rather than $D$ locations in each card as required by the superimposed random coding method.
This is essentially the argument for *grandmother cells* vs. *sparse distributed coding*.

Greene followed up this analysis with suggestions for a neural network implementation. Let each card correspond to one of $R$ output neurons, and let there be $N$ input lines. Selecting random $D$-tuples from these lines will generate $V$ secondary lines, each of which contacts all the output neurons. Let these neurons have a common threshold of $n$ units of excitation. Suppose further that conducting excitatory synapses with these neurons are formed by a classical conditioning process: on the average about $K$ per neuron. As Greene noted this can be set up in such a way as to correspond to card selection by activation of $n$ out of $K$ descriptors; and of course it corresponds to the standard architecture of the associative memory network. Greene did not implement this scheme with circuit diagrams, nor did he develop in any detail the various inhibitory mechanisms he saw were needed to control the operation of the system. Remarkably, all these mechanisms were implemented by Marr (1969) in a virtually identical system proposed (apparently without prior knowledge of Greene’s paper).

It is evident that the Mooers et al scheme can be interpreted as the embodiment of an error-correcting code into a network for storing and retrieving patterns. Superimposed random subset coding provides the appropriate encoding scheme, and thresholding corresponds to the majority logic decoding. It is natural to look for studies of the fault tolerance of such a scheme. In fact, Carter, Rudolph and Nucci (1989) have carried out such an investigation of Albus’ version of the scheme [Albus 1975]. They considered both ‘loss-of-weight’ errors in large weights, and ‘saturated-weight’ errors in small weights—the worst
cases of weight errors. An important property of the networks studied is that they share some of the feature mapping properties of Kohonen Maps in that neighboring inputs activate similar weight subsets. This is referred to by Carter et al. as *local generalization*, and the number of weights activated by any input, $\gamma$, is referred to as the *generalization parameter*. Simulations were carried out on a network of fixed size (250 weights per threshold element) in which the function or association to be learned is fixed, and $\gamma$ is varied. The results show that for loss-of-weight errors, the overall network error $P_N$, decreases with increasing $\gamma$. Conversely, for saturated-weight errors $P_N$ increases with $\gamma$. The task in such cases was to reproduce a single cycle of a sinusoid. In case the task is to reproduce a discontinuous function, e.g., a step function, the errors are usually produced by loss-of-weight errors in large magnitude weights. It follows that decreasing $\gamma$ should minimize $P_N$. Carter et al. make the point that since in general, pattern classification is effected by learning a discontinuous function over the input feature space, these findings imply that improved fault tolerance in such tasks might be obtained by reducing the generalization parameter $\gamma$, and learning the task more slowly.

### 2.2 Hopfield Networks

It is evident that analysing the fault tolerance properties of associative memories, such as we have described, is by no means easy. The associative memories introduced by Hopfield (1982), based on systems of coupled analog neurons with sigmoid characteristics are much more amenable to analysis. As is well-known a Hopfield Network is a fully connected recurrent network of $N$ units with a symmetric weight matrix $W$. Since the eigenvalues of such a $W$ are real, we
expect the network to settle into some stationary, non-oscillatory state. In fact
the associated network equations admit an energy or Liapunov function which
guarantees the approach to the stationary state. Let this state be represented
by the vector of binary components \( \Sigma = \{ \sigma_1, \sigma_2, \ldots, \sigma_N \} \), where each \( \sigma_i = \pm 1 \).
To program the network Hopfield used the standard ‘outer-product’ algorithm.
Let \( \Sigma^P \) be a pattern to be stored in the network. Then choose the weights
such that \( w_{ij} = \sigma_i^P \cdot \sigma_j^P \). Starting from a random initial state \( \Sigma^{0} \) the network
will settle into the state \( \Sigma^P \). Now let there be \( m \) patterns to be stored in the
network, \( \Sigma^1, \Sigma^2, \ldots, \Sigma^m \). Let the weights be chosen via the algorithm:

\[
    w_{ij} = \sum_{P=1}^{m} \sigma_i^P \cdot \sigma_j^P
\]

(6)

It now follows that starting from \( \Sigma^{0} \), the network will settle into the nearest
\( \Sigma^P \). Evidently if \( m \) is too large, ambiguity will result. Thus there is obviously
an upper limit on \( m \), or a ‘capacity’ for the network, which limits the number
of memories that may be reliably stored and retrieved. It is evident that there
is an analogy between the processes of encoding and decoding messages in a
communications system, and the storage and retrieval of memories in a Hopfield network. This analogy was exploited by McEliece, Posner, Rodemich and
Venkatesh (1987) who proved that for error free retrieval \( m \leq O(N/4 \log N) \).
This result should be compared with that of Amit, Gutfreund, and Sompolinsky
(1985) who exploited the analogy between Hopfield Networks and Spin-Glasses
to show that if a 5% error-rate can be tolerated \( m \sim 0.14N \). Here then is an
expression of the fault tolerance of associative memories. More recently, Biswas
and Venkatesh (1991) have used Random Graph Theory to investigate the ef-

ts of damage resulting in catastrophic failures to such networks. They proved
the interesting result that if \( p \) is the probability that a link between two units survives damage, then catastrophic failure occurs only when \( p \leq o(\log N/N) \), i.e. each unit need retain only \( o(\log N) \) links out of a total of \( N \) possible links with other units, to preserve some associations. Biswas and Venkatesh also looked at the question of sparse coding via the theory of Block Graphs. They proved that if a network of \( N \) units is partitioned into subsets or blocks of \( B \) units with full intra-block connectivity and no inter-block connectivity, then the effective capacity is \( O((B/4\log N)^{N/B}) \). This implies that if \( B \) increases with \( N \), then super-polynomial capacities can be attained, i.e. for large enough \( N \), partitioning can generate very large storage capacities in increasingly sparse networks. This is essentially what occurs in the Moore-Greene-Marr networks. Biswas and Venkatesh actually proved more than this: they analyzed the effects of errors in terms of the radius \( \rho \) of the hypersphere around each memory in the \( N \)-dimensional state space of the network, needed to correct errors. In such a case the effective capacity is bounded by

\[
((1 - 2\rho)^2(B/4\log N)]^{N/B}
\]

Thus, more error correction and fault tolerance means smaller memory capacity, as expected.

To actually correct errors in associative networks is straightforward. A common scheme is that suggested by Marr (1969) who noted that since sparse memory systems work reliably when not too many lines are activated, it is necessary to raise the thresholds of units in the network via feedforward or feedback inhibition. A recent implementation of this for Hopfield networks uses a ‘winner-take-all’ circuit [Moopen, Khanna, Lambe and Thakoor 1986]. In
addition slight asymmetries in the weight matrix $W$ are introduced. Together these result in better elimination of the spurious memories that are found in all associative networks, and better tolerance of errors.

3 Error–Correcting Codes and Fault Tolerant Networks

We have described the Winograd–Cowan construction whereby a Hamming code was embedded into the structure of the network. In recent years there has been a growing appreciation of the connection between codes and neural networks. Indeed several papers have appeared in which neural networks are used to implement the decoding process in a variety of codes. In this review however, we concentrate on the control of errors within the decoding process itself. A particularly interesting use of error–correcting codes in this connection is to be found in the work of Petsche and Dickinson (1990). These authors start out by making a fundamental point about the nature of the representations used in neural networks, from the distributed representations used, as we have seen above, in associative memories of the Hopfield type, to the more local representations used in sparse distributed coded structures of the Moors type, and to purely local systems. The point is that such representations are equivalent to codes of various kinds, from block codes in which an entire input sequence is mapped onto an output sequence, as in the Winograd–Cowan construction, to codes which do not mix any symbols. Petsche and Dickinson note that there is an intermediate family of codes which provide a somewhat coarse coded or ‘receptive field’ representation of input sequences, in terms of the action of a few semi–local encodings, namely, convolution codes (Viterbi and Omura 1979).
3.1 Convolution Codes and Trellis Graphs

The basic idea underlying a convolution code is that relatively short overlapping subsets of an input vector are used to determine the output vector. This is essentially what is involved in the Mooers et al. schemes, and in the coarse-coded representations with overlapping receptive fields introduced by Hinton, McClelland and Rumelhart (1986). An example is provided by the linear (3,1) triplet code, which is made out of three generators \( g_0 = (111), g_1 = (110), \) and \( g_2 = (011) \). Let \( u = (u_1, u_2, \ldots, u_k) \) be an input vector and \( u = (u_1, u_2, \ldots, u_k) \) the output vector, where each \( v_i = (v_{i,1}, v_{i,2}, v_{i,3}) \) is a 3-bit subset. Then \( v \) can be written as the convolution of \( u \) with the generators \( g_i \), i.e.

\[
v_i = \sum_{k=\min(i,k)}^{\max(1,i-2)} u_k g_{i-k}. \tag{7}
\]

Examination of eqn.7 shows that each 3-bit subset \( v_i \) of \( v \) depends only on the \( i \)th bit of the input vector and the two previous bits. The number of bits of the input vector that uniquely determine each output subset is called the constraint length \( K \). In the noise free case, \( v_i \) contains no information about \( v_j \) for \( |i - j| \geq K \). For a coarse-coded representation this corresponds to non-overlapping receptive fields.

In addition, one can represent the above encoding process by the trellis graph shown in Fig 6, the lower portion of which illustrates how overlapping subsets of the input \( u_1, u_2, \ldots, u_k \) are mapped onto a trellis graph for a \( K = 3 \) code. Examination of this graph makes it clear how error detection and correction can be achieved. Consider the two cases shown in Fig 7. In the first, no node in the center stage is activated, in the second a wrong node is activated. Because
of the convolution encoding, it is easy to see which nodes should be activated, and which activations are illegal.

How does this apply to the representation of images? Consider for example that shown in Fig 8, in which an image one-pixel wide is shown in an 8x8 raster. Suppose this image is filtered (convolved) with the set of receptive fields shown in Fig 9. These fields overlap as they process the image and lead to the set of constraints shown in Fig 10. But this is just a trellis graph, and we can identify the overlapping receptive field representation with a convolution code.

3.2 Trellis–structured Networks

It follows that a neural network can be constructed which incorporates convolution codes. Such a network can detect and correct errors in its input. In addition if learning is allowed, the network can also repair itself, in that failed components can be replaced by other (spare) ones. Fig. 11 shows the network architecture, in which the spare neurons are shown without lateral connections,
Figure 7: Error detection with a trellis graph. (a) A correct sequence or path through the trellis. (b) A missing activated node. (c) A wrongly activated node [Redrawn from Petsche and Dickinson 1990].

Figure 8: A one-pixel-wide image in an (8 x 8) raster. See text for details [Redrawn from Petsche and Dickinson 1990].

and all neurons in a column mutually inhibit each other with a weight equal to −1. The algorithm embodied in the network is dynamic. Each neuron is represented by a differential equation of the sigmoid type, and the connectivity is such that there is an overall Liapunov function for the system. The weight changes are Hebbian—excitatory weights increase if the pre- and post-synaptic weights are correlated; inhibitory weights increase if such elements are anti-correlated. The network is initialized so that all neurons have a threshold of +1, except the
spares, which have a threshold set to 0. In addition all spare weights are set to 0. When the network is functioning correctly, any single pixel image is represented as a configuration of active and inactive neurons in the network. Because of the winner-take-all circuitry of each column (mutual inhibition), only one neuron in each column is activated: corresponding to a node in the associated trellis graph lying on the path representing the image. In general the output of a convolution encoder depends on the current and previous $K - 1$ bits, where $K$ is the constraint length. It follows that each path through the trellis graph that corresponds to such an image will differ from all other paths corresponding to images by at least $K - 1$ nodes, and so each legitimate configuration of activated neurons in the network will differ from any other by at least $K - 1$ neurons. So if a neuron that corresponds to a trellis graph node fails, if $K > 2$, each image will still be represented by a unique pattern. Thus a trellis network with $K > 2$, 

Figure 9: The set of receptive fields at each pixel and associated arcs. See text for details [Redrawn from Petsche and Dickinson 1990].
even one with no spares, is fault tolerant. Given the spares, however, and the Hebbian learning properties described above, a trellis network can repair faults as well. Failures are repaired via weight modification of a spare neuron which gradually takes over the role of a failed neuron. Simulations carried out by Petsche & Dickinson confirmed this behavior. Such network abilities are similar to, but go well beyond those generated in early approaches to the isolation of faults and/or the self-repair of neural networks, e.g. Löfgren (1962), and Pierce (1965).

Finally the efficiency of such a network can be computed and (roughly) compared with other schemes. In a simple \((N,1)\) encoding scheme there are \(MN\) neurons. Such a network will fail if as few as \(N\) neurons (all representing the same item) fail. Thus the \(M(N-1)\) spare neurons only provide \(N-1\)
levels of redundancy. Such a scheme was introduced recently by Lincoln and Skrzypek (1990) for Back-Propagation networks. For a self-repairing trellis network, however, with $S$ stages, $M$ active neurons per stage, and $M(N-1)$ spares per stage, failure will occur only if all the spares in at least $K-2$ stages fail. Thus more than $M(K-2)(N-1)$ must fail before the network fails, and the effective redundancy in the network is a factor of $M(K-2)$ greater than in the first scheme. This is comparable with the increased levels of redundancy obtained in the Winograd–Cowan implementation of an $(N,k)$ Hamming code.

4 Multi-layered Perceptrons and Fault Tolerance

It is apparent that the various encoding schemes we have described, using either Hamming codes, superimposed random subsets, or convolution with receptive fields, give rise to a redundant representation of the input embodied in a layer
of internal or “hidden” neurons. The decoding of this representation generates a further layer of output neurons. Thus there is a close correspondence between the architecture of multi-layered Perceptrons, and the standard Information Theory paradigm discussed earlier. This correspondence led Séquin and Clay (1990), Judd and Munro (1993), and Murray and Edwards (1993) to investigate the fault tolerance of Perceptrons, particularly in the case when weight noise is present during training. Simulations by Judd and Munro (1993), for example show that training in the presence of hidden unit faults (misfirings) produces

![Graph showing fault tolerance in a three layer Perceptron.](image)

**Figure 12:** Fault tolerance in a three layer Perceptron. See text for details [Redrawn from Judd and Munro 1993]

more fault tolerance during subsequent testing. Fig. 12 shows typical results: enhanced fault tolerance is found in networks trained with a higher rate of hidden unit misfiring.

A somewhat more detailed study was carried out by Murray and Edwards (1993) who analyzed the effects of hidden unit faults on learning by adding
weight noise during training. Thus each hidden unit weight \( w_{ij} \) is augmented by a term \( \Delta_{ij} w_{ij} \) where \( \Delta_{ij} \) is a measure of the noise amplitude. The effects of this are to add extra terms both to the error function and to the update rules that govern the training. Consider for example a multi-layer Perceptron with \( I \) input, \( J \) hidden and \( K \) output nodes, a set of \( P \) training vectors \( \mathbf{q}_p = \{ \mathbf{q}_p \} \), and the error function:

\[
\epsilon_{tot,p} = \frac{1}{2} \sum_{k=0}^{K-1} \epsilon_{kp}^2 = \frac{1}{2} \sum_{k=0}^{K-1} (\mathbf{q}_p \mathbf{f}_{w_{ij}} - \mathbf{o}_{kp})^2
\]

where \( \mathbf{o}_{kp} \) is the target output. By expanding \( \mathbf{o}_{kp} \) to second order around the noise-free weight \( w_{ij} \), and time averaging over the learning phase, one can easily show that two additional terms appear in the overall error function, namely:

\[
\frac{1}{2P} \sum_{p=1}^{P} \sum_{k=0}^{K-1} \Delta^2 \sum_{ij} \frac{\partial^2 \mathbf{o}_{kp}}{\partial w_{ij}^2} \left( \frac{\partial \mathbf{o}_{kp}}{\partial w_{ij}} \right)^2 + \epsilon_{kp} (\frac{\partial^2 \mathbf{o}_{kp}}{\partial w_{ij}^2})^2
\]

Furthermore, the update rule on the hidden-output layer becomes:

\[
< \delta w_{kj} > = -\tau \sum_p < \epsilon_{kp} \mathbf{o}_{j\mathbf{p}} \mathbf{d}_{kp} > -\tau\Delta^2 \sum_p < \mathbf{o}_{j\mathbf{p}} \mathbf{d}_{kp} > \cdot \sum_{ij} \frac{\partial^2 \mathbf{o}_{kp}}{\partial w_{ij}^2} (10)
\]

averaged over several training epochs in case that the adaptation rate \( \tau \) is small.

Murray and Edwards studied the effects of these additional terms on learning in the case of classifiers and character encoders. The results are as follows: the first term in eqn. 9 tends to favor a more even distribution of weights across the network. Combined with the term in eqn. 8 it tends to produce a more distributed representation which favors increased weight tolerance. Simulations similar to those carried out by Judd and Munro (1993) confirm this. The action of the second term in eqn. 9 is more subtle. It tends to structure the error surface in the early phases of training when \( \epsilon_{kp} \) is large, in such a way
as to favor lower overall error. It therefore tends to promote faster learning. Interestingly if weight-noise is too high this effect is swamped. Thus there is an optimal level of weight-noise of about 30% involved in reducing training times. Finally, Murray and Edwards considered the effects of injected weight-noise on \textit{generalization}—the ability to process new data correctly. Here the first term in eqn. 9 again has an effect: it tends to favor solutions with $o_{jp} = 0$ or 1, i.e. with hidden units either ON or OFF. In such a case hidden units are much less likely to be affected by weight-noise, \textit{and also by input-noise}, leading to increased generalization ability. Simulations confirm this. There is a dramatic increase of generalization ability in classification with increased levels of weight-noise during training. An improvement of about 8% is seen. The reader should compare this with the results obtained by Carter et al. (1989) discussed in §2.1 and also those of Séquin and Clay (1990).

\subsection{4.1 Regularization}

These results indicate that a process of nonlinear optimization is required to obtain the appropriate weight settings to produce fault tolerance. Injected weight-noise in effect adds constraints to the usual error or penalty functions which must be minimized in the process of optimization. As Neti, Schneider and Young (1992) have noted, this is similar to the \textit{regularization} processes employed by Poggio and Girosi (1990) for the learning of continuous maps from a finite number of examples—a classic “ill-posed” problem. Neti et al. in fact formulated the fault tolerance problem in such terms, and obtained what they called “maximally fault tolerant” networks for several auditory encoding problems. Their results are similar to those obtained by Murray and Edwards, in
that more distributed and uniform representations are more fault tolerant, and generalize better; and are also similar to those of Petsche and Dickinson (1990) in that increasing the number of hidden units, up to some limit, also leads to improved fault tolerance and better generalization. Recently, Bishop (1995) has shown that training with noise is equivalent to Tikhonov regularization, and that direct minimization of the regularized error function does indeed provide a practical alternative to training with noise.

5 Miscellaneous and Concluding Remarks

In this review we have discussed many papers on fault tolerance which use redundancy techniques in a variety of ways. One approach we have not discussed is that followed by Satyanarayana, Tsividis and Graf (1990) who constructed a reconfigurable analog VLSI neural network chip comprised of distributed-neurons, each with $N$ weights. Among the advantages gained by such a construction are that large current build-ups in each neuron are avoided, and since the chip is reconfigurable, defects leading to failures can be isolated and ignored, thus producing greater fault tolerance in the neurons themselves. This provides another reason to assume that neural firing error probabilities are largely independent of $N$, as we discussed in §1.

Finally we note that neurobiology is also relevant to the fault tolerance problem. Recent experimental measurements by Stevens (1994 Preprint) and Jack (Personal communication, 1994) of LTP and LTD in synaptic transmission indicates that learning first produces an increased reliability of transmission before there is an increase of synaptic weight. Thus intrinsic weight-noise is
lowered during learning. The consequences of this effect, for the various schemes we have discussed above, remain to be developed.

Acknowledgements

This work is supported in part by the Brain Research Foundation of the University of Chicago.

References


Dobrushin, R.L. & Ortyukov, S.I. 1977, “Lower bound for the redundancy of


Moers, C.N. 1949, “Application of Random Codes to the Gathering of Statis-


