Technical Report: YCS 154

Investigating Fault Tolerance in Artificial Neural Networks

George Bolt[†]

Advanced Computer Architecture Group
Department of Computer Science
University of York
Heslington, York, YO1 5DD, U.K.

25/3/91

† This work was supported by SERC and also by a CASE sponsorship with British Aerospace Brough, MAL

Table of Contents

Abstract iii

Section 1 Introduction 1

Section 2 Review of Past and Current Work 3
2.1 A Framework for Fault Tolerance 3
2.2 Redundancy 4
2.3 Computational Complexity and Capacity 4
2.4 Reliability during the Learning Phase 5
2.5 Retraining 5
2.6 Input and Output Representations 6
2.7 Basins of Attraction 6
2.8 Associative Neural Networks 7
2.9 Hopfield Neural Network Model 8
2.10 Multi-Layer Perceptron Model 9
2.11 CMAC Networks 10
2.12 Compacta Networks 10
2.13 Fault Tolerance of "Real" Neural Networks 11
2.14 Conclusion 11

Section 3 A Framework for Reliable Neural Networks 12
3.1 Fault Tolerance and Robustness in Neural Networks 12
3.2 Determination of Fault Tolerance in a Neural Network 13
3.4 Error Recovery 14
3.6 Influences on Fault Tolerance and Robustness in Neural Networks 15
3.5 Graceful Degradation 16
3.7 Conventional Fault Tolerant Techniques in Neural Networks 16
3.8 Conclusions 17

Section 4 Fault Models 18
4.1 Conventional Fault Models 20
4.2 Fault Locations 21
4.2.1 *Example* 23

4.3 Fault Manifestations 24
4.3.1 *Example* 24
4.4 Functional Fault Models 28
4.5 Spatial and Temporal Considerations 28
4.6 Conclusions 29

Section 5 Assessing Reliability 30
5.1 Failure in Neural Networks 30
5.1.1 Measuring Failure 30
5.1.2 Applying Failure Measures 32
5.1.3 *Example* 32
5.2 Timescales 33
5.3 Fault Injection Methods 34
5.3.1 *Example* 34
5.4 Mean-Time-Before-Failure Methods 35
5.4.1 *Example* 36
5.5 Service Degradation Methods 36
5.5.1 *Example* 37

Section 6 Conclusions 38

ABSTRACT

A review of fault tolerance in neural networks is presented. Work relating to the various issues of redundancy, reliability, complexity and capacity are considered, as well as covering both empirical results and a general treatment of theoretical work. It is shown that in the majority of the work, few sound theoretical methods have been applied, and that conventional fault tolerant techniques cannot straightforwardly be transferred to neural networks. It is concluded that although neural networks are often cited as being fault tolerant, little substantial evidence is available to support this claim.

A proposal for a framework which can be used to assess fault tolerance and robustness in neural networks, and also to guide work in this field is given. Various factors which might influence the reliability of such a system are discussed.

To support this framework, two fundamental prerequisite stages are described in sections 4 and 5 which can act as a base for research into the fault tolerance of neural networks. Section 4 describes how fault models can be developed for neural networks visualised at the abstract level. This involves first locating where faults can occur, and then defining the attributes of the faults. Section 5 uses this fault model to develop various methods which can be used to measure the reliability of the neural network. Fault Injection and Mean-Time-Between-Failure methods are examined, and from these a more appropriate Service Degradation Method is developed. Two critical issues of how to measure the degree of failure within a neural network and how to choose a suitable timescale are discussed. The multi-layer perceptron network model is used in examples which illustrate how ideas described here can be applied.

1. Introduction

Any viable system requires that prior knowledge exists on how it will behave during its lifetime, which could range from seconds to decades. Since in critical applications it is almost always nonsensical to use a system for which there is no idea of how long it will run without a failure, how robust it will be to noise, or how much damage it can sustain, it is vital that these factors be understood.

Neural networks offer a parallel distributed method of processing information unlike that of conventional serial computing systems, also their underlying basis of computation is analogue rather than digital. They were inspired from studies of the structure of the brain [32], though it must be realised that artificial neural networks are a very simplified model of biological neural networks, and are also very much smaller. However, it is generally accepted that neural networks are well suited to solve problems which are very successfully tackled by biological neural systems such as our brain. Artificial neural networks consist of a large number of simple processing units (often termed *neurons*) which are highly interconnected. Each unit forms a weighted sum of its inputs, then thresholds it with respect to some internal bias value using some bounded non-linear function. Control of the order in which the neural networks' units are updated varies; models tend to be either totally autonomous, synchronous, or sequential. Examples of application areas in which neural networks are often applied include associative and content-addressable memory systems, image and speech recognition, hard optimisation problems, noise filtering, control of dynamical systems, and classification tasks. Since some of these are likely to require provably reliable systems, it is necessary that some indication can be found of what will happen during the lifetime of such a system which incorporates neural networks. So the reliability of neural networks must be determined, and this will then indicate the suitability of their use when such a reliable, robust system is required.

One great advantage of a system that employs neural networks is that they can be taught to solve problems either by example which is generally known as supervised learning [41] , using a reward/punishment system which is often called reinforcement learning [53] , or by some internally directed method which is termed unsupervised learning [25]. However, this semi-autonomous feature of neural networks immediately raises the question of whether it can be verified that a neural network system really performs the task that it has been directed to learn, as well as establishing whether it is reliable during operation, generally considering a long time period. The required degree of both operational accuracy and reliability will depend on its use; in a safety-critical situation such as controlling an aircraft it is vital that the neural network system is highly dependable.

The common assertion is that neural networks are inherently fault tolerant or robust. If this is so, then a system is available which requires little or no extra cost such as extra components, power requirements, maintenance, etc. for achieving the required reliability as would otherwise be the case for a conventional computing system. The hope is that neural networks exhibit fault tolerance as a natural consequence of the system's computation rather than due to additional features. Biological neural networks are known to be remarkably tolerant of failures as well as of noise in input sensory data. This can be seen from the numerous lesion studies which have been performed, and also the noisy data which our senses receive is another good indication. If our neural networks are good models of biological neural networks, then it is reasonable to suppose that this characteristic of biological neural networks will carry over to our artificial neural networks. However, analysis, supported by comprehensive experimental results, must be carried out rather than relying on the illustrative ideas above. Such investigations of the reliability of

neural networks will require both fault models and reliability measures to be defined for them.

The first part of this report consists of a review of the literature in the field of reliable neural networks, followed by a general discussion in section 3 of an investigative framework for reliability in neural networks and possible factors which might be important in influencing the fault tolerance and robustness of a neural network. Before any such investigation can take place, a requirement exists for both fault models and reliability measures to be defined for neural networks. Methods by which these can be constructed are given in sections 4 and 5. Finally, conclusions are drawn in section 6.

2. Review of Past and Current Work

There are few major pieces of work which study the field of reliable neural networks. Papers or reports which have been published tend to be either a general discussion of fault tolerance, or experimental results of the effects of noise/faults in neural networks; a common misunderstanding is that of confusing fault tolerance with robustness to noisy inputs. Very little theoretical work has been carried out. Overall it can be seen that no common consensus exists on how to investigate the reliability of neural networks, and so the vast majority of work tends to be fragmented and disjoint.

Fault tolerance is often attributed to neural networks [4, 6, 9, 14, 15, 19, 23, 27, 29, 52, 55, 56], but only as a passing comment or informal verification that a degree of fault tolerance exists in the neural network model being discussed, rather than being the main objective of a serious investigation.

2.1. A Framework for Fault Tolerance

A requirement exists for a quantitative theory describing the reliability of a neural network, as also has been noted by Carter [16]. This paper is by far the most detailed published work on the notion of fault tolerance in neural networks, though understandably it is far from comprehensive. The scope is limited to *"applications of pattern recognition and signal processing"*, recognising that neural network systems which solve optimisation problems are qualitatively different. Also, to avoid possible confusion with classical terminology in which the generally accepted definition of fault tolerance is the notion that a system provides *"error-free computation in the presence of faults"*, Carter uses the term *"robust"* rather than fault tolerant to describe a neural network since they only ever give approximate solutions [27]. He also draws a very significant distinction for the analysis of fault tolerance in a neural network system between the two phases of neural network processing: *Training* and *Operation*. The effects of faults are likely to be different during these two distinct periods in a neural network's lifecycle. Carter also identifies *implementation-specific fault tolerance* to be another area for separate analysis in **conjunction** with the other two areas mentioned above. However, this seems to be an incorrect partitioning for the analysis of reliability in neural networks, since the implementation method used is quite likely to affect very differently the fault tolerant properties of neural networks during the operational and training phases. For example, the weights of connections are only changed during the learning cycle, and so the method used in the implementation for weight alteration will lead to reliability issues that are only appropriate during this cycle. An alternative taxonomy which is more likely to reflect the way in which factors affecting the fault tolerance of a neural network can be investigated is described in section 3.

Brause [12] has indicated just how important the fault tolerance of neural networks is when considering how to implement systems that will perform tasks that are currently very hard to solve. He points out that threshold functions are important for quelling noise in signals, and has shown this for associative memory systems.

2.2. Redundancy

Redundancy of neural network components (both units and connections) has often been identified as the factor producing a reliable system. Moore [37] compares conventional techniques to introduce fault tolerance into a computer system with apparent mechanisms in biological neural networks, and then draws various implications from this. He points out that biological networks use both spatial and temporal redundancy (both in relation to components and input/output representations), as do conventional computing systems to achieve a greater degree of fault tolerance. However, von Seelen and Mallot [50] question whether redundancy really is the key issue in determining the reliability of a neural network, both in terms of fault tolerance and graceful degradation. They say, very plausibly, that a neural network does not have redundancy in the sense of reserve capacity, but rather it utilises all of its resources to gain the best trade-off between accuracy and computation time. Fault tolerance comes from *"isomorphic implementation, natural representation, a small number of computation steps, and a balanced utilization of all available resources."* [50] By isomorphic implementation, they mean that the output of the neural network can be directly related to the internal processing within the network. However, though this is reasonable for simple networks such as the Hopfield model where a clear trajectory is followed through state space, for more complex models it does not seem to be quite such a valid claim. Also, a natural representation is often a redundant signal in its own right (e.g. retina image), so some of the influences on fault tolerance in neural networks that they identify do not seem to be wholly justified.

2.3. Computational Complexity and Capacity

The concept proposed by von Seelen and Mallot [50] that a neural network utilises resources to the full, advances Carter's [16] notion that fault tolerance in a neural network is due to redundancy being introduced because of "spare capacity" when the complexity of the problem to be solved is less than the computational capacity of the network.

If redundancy does originate in this manner, then it is obviously important to be able to determine both the "computational capacity" of a neural network and the "computational complexity" of a problem. Much of the work on memory capacities of neural network systems can be applied here [4, 14, 23], though some more general work has also been done.

Abu-Mostafa [2] shows that a neural network can solve any finite problem by simulating boolean logic gates. However, due to practicality and efficiency considerations, and also since feedback is not incorporated into the argument, then this interesting result is somewhat degraded. He observes that the complexity of the problem to be solved greatly depends upon the representation of the input presented to the neural network, since it can be viewed that a network is just performing a change of representation. This observation can be extended to include the choice of output representation as well in the case of supervised learning. An important point also made is that the capacity (in a general sense) grows faster than the number of neurons/units in a neural network, so it will be more efficient solving random problems [1] such as complicated pattern recognition for example, rather than small structured problems. This incidently parallels with the problem solving capabilities of humans.

Hartley and Szu [21] have found that a large number of neural network models can be shown to be equivalent in power to a Turing machine if the number of units is infinite, else they have the power of a finite state machine. This result parallels Abu-Mostafa's claim. They point out that if

further restrictions are placed on the network, such as having a symmetrical weight matrix, then the power of the neural network is greatly decreased to below that of a finite state machine.

Some related work which is directed towards finding out the correct sized linear threshold network for valid generalisation has been done by Baum and Haussler [8], in which two measures of capacity are used. These are the maximum number of dichotomies that can be induced on *m* inputs and the *Vapnick-Chervonenkis (VC) Dimension*. This latter value is the dimension of the largest input space that can be completely dichotomised by some set of functions. The first can be related to the maximum possible computational complexity of a problem set in *m* dimensional space, and the second can be seen to closely relate to the computational capacity of a network. However, the VC dimension only applies to units which output a boolean value.

2.4. Reliability during the Learning Phase

The nature of the reliability of a neural network during the training phase is important to determine, as has been noted by Carter [16]. He questions what accuracy the neural network can achieve given the faults that might occur during training, how much longer it will take to train the network, and also how fault tolerant the final network will be. The answers to all of these questions will depend in part on the difference in computational capacity of the network and the complexity of the problem. Moore [37] claims that neural networks will adapt to any faults due to their learning ability, however, this claim is presented with no justification other than that biological neural networks behave in this fashion. However, von Seelen and Mallot [50] note that in lesion experiments carried out in their laboratory, there was *"no compensation for [the] deficits even after prolonged learning"*. These lesions were carried out on the visual cortex areas. It is unlikely though that this result will be exactly equivalent to what will happen in an artificial neural network during learning. Localised representations are less robust than distributed representations to such structured damage, whereas they are probably just as robust to random damage.

Anderson [6] has made an interesting point that for distributed neural networks, any fault that occurs during teaching will itself be distributed, and so will be both hard to locate and also to remove. Since the dynamics of the system during operation are very complex, the distributed nature of the error will lead to important implications for the verification of neural network systems, since it may only manifest itself clearly in only a few situations.

A study has been made by Pemberton and Vidal [39] on the consequences of having noisy training data during learning in a single threshold logic unit, trained using the perceptron (discrete), the Widrow-Hoff (linear), and the generalised delta (non-linear) learning algorithms. It was found that the output error rate of a unit trained using the perceptron rule followed approximately linearly to the training signal error rate. For the linear and non-linear training algorithms, the output error rate did not significantly increase until the training signal error rate reached about 40%, the non-linear rule being slightly better. It was also found that the value of the learning rate greatly affected the robustness of the unit to training signal noise for the linear and non-linear rules, as did the scale factor used in the threshold function in the non-linear case.

2.5. Retraining

Once faults have occurred in a neural network system, its performance will be degraded since computation in a neural network is distributed evenly amongst its components, as von Seelen and Mallot [50] have noted, so a fault will always result in a deficit. Hence retraining could be considered as a method to alleviate the problem and restore accuracy in processing. Tanaka [47] has experimented with relearning in a multi-layer perceptron (MLP) neural network, and found

that it was possible to regain high levels of accuracy even after many faults had occurred, though the time taken to recover increased beyond reasonable limits for many such faults. This possibly relates to the computational capacity of the network diminishing towards the minimum needed to solve the problem.

Bedworth and Lowe [10] have also investigated relearning, again in a MLP network. Their network was considerably larger than Tanaka's, and even with half the connections in the network removed, it could still recover near original performance after a fifth of the initial training time.

The ability to retrain a neural network system at intervals may well produce a system that can easily meet long term reliability goals [37], though the retraining system will then become a key factor in the reliability equation of the overall system.

2.6. Input and Output Representations

It has already been discussed how that a distributed internal representation can lead to a more reliable neural network system, but the same can also be said for the input and output representation used. Methods for forming distributed representations are discussed by Miikkulainen and Dyer [35, 34], and include extending the back propagation algorithm to modify to input vector passed to the network. They found that the final neural network was fault tolerant to damage in its input layer of units due to the learned distributed representation, and also that it degraded in an approximately linear manner. However, the system requires a lexicon to map actual "world" input vectors to the distributed input vector the network requires, and this could become the keystone for the reliability of the overall system.

Various input representations for numerical values are considered by Takeda and Goodman [46], such as a binary and simple sum scheme, in view of examining how the input representation chosen affects the learning capabilities of the neural network. However, they do note that the binary scheme is not particularly fault tolerant, but the simple sum scheme is, since a single bit error will only cause a small change in the number represented. Hancock [20] describes various other possible representations, but only considers the effects on learning.

2.7. Basins of Attraction

The concept of a "basin of attraction" in a neural network is linked to the visualisation of its energy landscape. It can be viewed as an area of this landscape over which a stored pattern has complete influence. They have mainly been associated with Hopfield networks used as auto-associative memories [45, 4] where formulas have been developed that give the size of the basin of attraction in terms of the maximum number of bits different in the initial state vector to that of the stored pattern, and still recover it.

More generally, Krauth *et al* [26] have studied basins of attraction in neural networks that are based on the perceptron model [32, 36]. The structure of the networks that they study are two-layer feed-forward types composed of binary units. They claim that the existence and large size of basins of attraction in the neural network are important factors in rendering the network fault tolerant. However, it should more strictly be said that they give rise to robustness, or resistance to noise. It is unclear as to how the size and existence of basins of attraction will be affected by faults occurring within the network.

2.8. Associative Neural Networks

Reliability of recall in auto-associative neural networks has been investigated by Lansner and Ekeberg [28], and more importantly they have considered the quantitative nature of the relationship between the storage capacity and reconstructive power in such systems. However, although their work is interesting, especially since theoretical predications are given, they have only considered the elimination of noise from the input pattern rather than the effects of faults occurring within the neural network. Two very useful definitions which they give are the expected local recall reliability (LRR) and the global recall reliability (GRR). They define LRR as the probability that the next unit to activate will be in the correct associated pattern given *n* units already activated. GRR is merely the extension of LRR to the probability that the input pattern is associated correctly. Note that these definitions rely on a relaxation process occurring in the neural network and also that at most only one unit is allowed to become active at each relaxation step.

The use of the brain-state-in-a-box model (BSB) [7] as an associative memory is considered by Anderson [6]. This neural network model consists of a positive feedback loop with limits placed on the output values the units can attain. Anderson suggests that the system might well be useful as a preprocessor for noisy input data due to its auto-associative properties. Wood [54] has carried out simulations of faults occurring in the feedback matrix and found that the results lead to a mixed conclusion. Although a gradual decrease in accuracy might be expected from statistical predictions, the simulations showed that both distribution and localisation occurred in the neural network. It can be concluded that it would be very useful to have some measure of the degree of the distributed nature of information held in a neural network.

Internal representations have been studied by Baum *et al* [9] for a class of feed forward neural network models which use an intermediate layer of units between the inputs and outputs. They investigated both unary (Grandmother cells) and distributed representations. They point out that obviously the unary representation gives rise to fault intolerance, though redundancy can be introduced by duplicating the grandmother units (which is similar to ideas in Legendy's compacta networks [29]). However, this is not completely true since redundancy will still occur in the connections feeding each grandmother unit, and faults affecting the connections rather than the unit directly might not cause a sufficiently large change in the internal state to alter the outcome of the winner-take-all process.

They stress that a distributed representation must not simply be several unary representations combined together, i.e. individual units in the hidden layer still responding to only one stored pattern, but truly distributed in the sense that units in the hidden layer respond to several stored patterns, though this overlap must be controlled to minimise interference. They then go on to consider the effects of faults causing some input bits to be forced to zero, and derive equations for a particular training algorithm which indicate the memory capacity for some required output accuracy. It is pointed out that it must be considered that the redundancy introduced by the distributed representation is balanced by the need for connection weights to be more than simply one of two states. They also point out that sparsification of the internal representation improves the memory capacity still further, though it is likely that this reduces the fault tolerance due to the movement towards a unary representation. A compromise clearly exists between fault tolerance and the capacity of the neural network arising from the chosen internal representation.

Anderson [6] has also differentiated between unary and distributed representations by considering feature detectors which either consist of one neuron (microfeature) or of several (macrofeature). However, the vector feature model (BSB) he mentions that can occur by using lateral excitation, as occurs in the cerebral cortex, will result in several units behaving as a feature

group, but this is not the distributed representation as defined by Baum *et al* above. This may be the reason for the rather mixed results which Wood found, as described above.

Venkatesh [49] has considered allowing errors from a known distribution to occur in the outputs of an associative neural network, and also specifying that some of the output units are redundant (i.e. don't care what their value is) so as to improve the memory capacity. Basically it can be viewed that extra redundancy is introduced into the output layer of units, and this then improves the network's robustness to noise and also to any unit failures. The results of this rather strange method are not based on any particular neural network model but are generally applicable. It is found that the capacity can be increased, and is related to the number of units in the neural network and the proportion of allowable errors made by an output unit.

2.9. Hopfield Neural Network Model

The Hopfield neural network [23] has been analysed for various objectives by many researchers, and since it is a model for spin-glass theory, it is extremely amenable for such mathematical analysis. However, only a few people have examined the fault tolerant properties of the model, and the work that has been done tends to cover the same issues. Amit *et al* [4] present a classic example of the application of spin-glass theory to the Hopfield network. They also highlight some issues which are important for robustness in the model, but do not consider the effects of faults. The relaxing of the rather extreme condition of full interconnectivity in a Hopfield network was found to decrease the storage capacity and quality of recall of the network only gradually. Also, by not restricting the weight matrix to be symmetrical a similar result occurred. This condition is one which Hartley and Szu [21] noted would decrease the power of a network. Two other issues, the saturation of the network and noise at synapses are also shown only to decrease the storage capacity gradually.

A purely empirical study of the Hopfield network has been carried out by Palumbo [38] which examines the effects of unit faults (s-a-0 and s-a-1) for networks trained to solve the travelling salesman problem (TSP), an assignment problem, and also a task allocation and load balancing problem. A degree of fault tolerance is shown to be exhibited by the neural network, but no measure of performance is given. An indication is made that this latter point is an important problem that needs more research.

A VLSI architecture for implementing the Hopfield neural network [43] was found to exhibit the inherent fault tolerance of the Hopfield model, but the network was so small (stored only 2 patterns) that it is unclear whether the results would scale to larger networks.

The theoretical implications of faults in a Hopfield network have been partially studied in [45] where the effects on the probability of correct recall and the signal-to-noise ratio are given for both faults in units and connections. Initially, results of simulations are given which show that so long as not too many patterns are stored in the neural network, the fault tolerance with respect to faulty connections is very good; with up to 40% faults there is still a good probability of accurate recall. Analogies are drawn between the fault modes possible in the abstract model with what might happen in an implementation of the network, but it is unclear if these are valid. Theoretical results are then given for faults in both units and connections which are supported by the experimental data. The results are obtained by considering the probability that the number of failed units exceeds the size of the basin of attraction [33]. Tai concludes that *"the network fault tolerance depends heavily on the chosen fault model and on the number of stored patterns."*

Belfore and Johnson [11] present an analogue implementation of the Hopfield network, and then proceed to list the possible fault modes that could occur in their *"electrical neurons."* This reflects

the overall approach of this paper towards investigating fault tolerance at the implementation level. The effects of faults were examined for the network both solving the travelling salesman problem (TSP) and also for it acting as an associative memory. Once again, the conclusion is drawn that neural networks seem to be inherently fault tolerant, and they draw an analogy with holograms only losing resolution when portions are cut away.

2.10. Multi-Layer Perceptron Model

Several researchers have looked at the multi-layer perceptron model trained using the back-propagation algorithm [41] with respect to its fault tolerant properties. Damarla and Bhagat [18] trained both two and three layered networks (2-10-1 and 2-5-5-1) to solve the boolean exclusive-or problem (XOR), keeping the number of hidden units constant in an attempt to allow the results from both neural networks to be comparable. However, it is generally accepted that the number of connections in the network is the significant factor, and there are only 30 weighted connections in the first network as compared to 40 in the second. They reported that if the weights were left unconstrained then no significant results were found. This was probably due to the units with large weights dominating the network. Constraining the weights greatly increased the training time for small networks, but the robustness of the network to noise and also to units being removed was greatly improved over the unconstrained network. However, given that only two hidden units are required to solve the XOR problem, the reliability observed was probably due to the network "overtraining", and becoming more like an associative memory system [48, 42], than it discovering categorisation rules because of the comparatively very large neural networks that were used. Further, the vast majority of the constrained weights had saturated to the clipping value. Considering Abu-Mostafa's [1] indication that neural networks are more likely to perform better on a random rather than a structured problem (such as XOR), then the results of this paper are unlikely to be representative.

A more in-depth, though again only empirical, investigation has been carried out by Bedworth and Lowe [10] in which they indicate that fault tolerance arises due to MLP networks *"lead[ing] to distributed rather than localised representations."* They investigated a large neural network (760-16-8) trained to recognise the confusable 'EE' sounds from the English alphabet. The neural network was corrupted in various ways which were designed such that the effects might be similar to what would occur if it was implemented in hardware. Performance was measured by two factors; the first being the number of correct classifications, and the second the normalised error between the actual output and the desired output. In general, no constraints were placed on the weights. They found that the robustness of the neural network was very good, except for faults which occurred in the connections feeding the output units and also for hidden units forced to output zero; which is equivalent to removing many connections to the output units.

Tanaka [47] also claims that fault tolerance in neural networks is due to their *"isotropic architecture"*, and draws parallels from the fact that every day many neurons in the brain die. Again a large neural network is used to collect experimental results (90-50-10), but the problem is merely one of classifying the figures '0' to '9' represented as dots in a 15 by 6 matrix. For only 10 input patterns, the number of hidden units seems excessive, and the network is likely to tend towards an associative memory rather than developing feature detectors. Once again a high degree of fault tolerance is displayed by the network.

Although many investigations of fault tolerance in multi-layer perceptron neural networks have been carried out, little theoretical work has been done, mainly since MLP's are notoriously hard to analyse mathematically. Zymslowski has given some very general equations on the effects of

parameter changes in a neural network [57], but unfortunately there is no obvious way in which these can be applied to produce a reliable neural network. However, he does show that both redundancy of connections and feedback lead to fault tolerance, but stresses that more effective mechanisms should be sought.

A probabilistic multi-layer perceptron type neural network (PNN) has been looked at by Specht [44] where the more usual sigmoid threshold functions are replaced by probabilistic ones. It is shown that given certain trivial conditions, the PNN will asymptotically approach the Bayes optimal decision surface. A very useful calculation can be made from the input values to an output unit which gives the probability that the input to the PNN belongs to the class which that particular output unit represents. This, although not a suitable measure for the reliability of the network with respect to fault tolerance, does give a confidence value for the output classification which might indicate the noise level in the inputs. However, if the input is corrupted such that it resembles another input class, as might happen for input classes that are not too dissimilar, the confidence value will be incorrect, so it is not totally reliable.

2.11. CMAC Networks

Carter *et al* [17] have looked at the fault tolerance during operational use (rather than during the training phase) of the Cerebellar Model Arithmetic Computer (CMAC) described by Albus [3]. This paper follows the guidelines for investigating fault tolerance as given by Carter [16] in an earlier paper.

The CMAC network was designed to be used for robot manipulator control, and as such it can learn to approximate nonlinear functions. The object of the paper is to study the sensitivity of the output to faults in the network, though these were limited to only the adjustable weight layer due to the complexity of analysing the effects of faults in the rest of the network. Two severe fault modes were considered, the first being the loss of a weight, and the second, a weight value being saturated. They continued to follow the strategy of aiming to cause the greatest possible effect in the network by placing loss of weight faults where weights were large, and saturated weight faults where weights were small. This strategy was followed so that hopefully the limits of the robustness of the network would be reflected in the results. They found that the network became more robust for loss of weight faults as the generalisation parameter in the CMAC network was increased, but not for saturated weight faults. However, for discrete mappings the generalisation parameter had to be decreased to improve the robustness of the network.

They concluded that the CMAC network is not so fault tolerant as it would at first intuitively appear, and that the robustness to faults is not uniform. Quite rightly they also stressed that *"one must be cautious in making assessments of the fault-tolerance of a fixed network on the basis of tests using a single mapping."*

2.12. Compacta Networks

Compacta networks are based on a theory for information storage in the human brain advanced by Legendy [29] in 1967. The neural network model has a sparse, random interconnectivity of input and output units based on McCulloch-Pitts neurons [32] arranged in a hierarchical fashion, and a simple learning mechanism. The network is updated synchronously. Diffuse groups of units (minor compacta) represent a single entity in the overall distributed representation, so a loss of many units will only cause a few units to be lost in each compacta on average, and so fault tolerance emanates from the redundancy of units within each minor compacta. Legendy considers both the loss of units and also the effects of noise from spuriously firing units (internal noise, not

external). He shows that with 10% or less units faulty, the effects on the "ignition" of minor compacta (all units in the minor compacta firing) is negligible. However, for internal noise, less than 0.4% of units must be spuriously firing, else the dynamic threshold is temporarily raised by the system causing all activity within the network to cease temporarily.

Worden and Womack [55] have recently proposed a more detailed study of the fault tolerance of compacta networks. They are interested in the effects of faults with respect to both the capacity and the accuracy in such a network. The paper merely lays out the guidelines for their proposed study, and mentions possible factors that might affect the fault tolerant properties of compacta networks, but no simulations or analysis has yet been performed.

2.13. Fault Tolerance of "Real" Neural Networks

It is vital to investigate the fault tolerance of neural network models at the abstract level, but consideration must be also be made of how to implement a neural network model using some technology (electronic, optical, biological, etc.) such that the inherent fault tolerance of the model is retained, and also of how to enhance the reliability of the system at the implementation and technological level as well (see section 3). This latter consideration might also be vital to protect the inherent fault tolerance of the model, depending on the implementation.

In large systems, due to the multiplicity of individual units, the loss of a few units is unlikely to cause any noticeable decrease in accuracy in the overall system. Indeed, some work has shown that unit losses of up to 40% can be tolerated in the Hopfield model [45]. This tends to suggest that non-critical components within the neural network system need not be particularly reliable due to the inherent overall fault tolerance, as has also been noted by Belfore and Johnson [11]. However, Moore [37] makes a contentious claim that because of the large number of components *"neural computers will need to be designed with high quality components."*

Once neural networks are actually included in commercial systems, it will be necessary that the reliability of a neural network implementation can be assessed. So although it is vital to analyse neural network models at the abstract level initially, it is also very important to be able to analyse various implementation architectures, as well as taking into consideration the technology chosen.

2.14. Conclusion

Very few sound theoretical approaches to fault tolerance in artificial neural networks have been made, and those that have, tend to raise yet more questions to be answered. A complete framework for investigating fault tolerance issues has still to be completed, and it is unlikely to fully emerge in the near future since there is a vast amount of work that needs to be done.

Simulations to discover quantitative results of the reliability of various neural network models is useful but limited. It is difficult to compare the results from one model to another in a meaningful way since there is no agreed quantitative measure for fault tolerance that is independent of the particular neural network being studied. Also, the fault model chosen for such a study has been shown to strongly influence the fault tolerance of the neural network. The various correlations and independencies between all of these factors (as well as others) which can influence the inherent and measured fault tolerance need to be placed into the investigative framework mentioned above. Finally, although it is interesting to look at the fault tolerance of implementations of neural networks in silicon or some other technology, in the long term this type of investigation is of little or no use, it is the fault tolerance that arises purely from the model’s abstract structure and dynamics that should be at the core for the study of reliability in neural networks.

3. A Framework for Reliable Neural Networks

It is clear that this field of study is very wide, encompassing many different neural network models and several conceptual viewpoints. For results to be comparable, a framework must be built which will allow for a flexible study of the reliability of neural networks such that general results can be obtained. This requires a categorisation of the causes of failures affecting the reliability, the relevant architectural and operational features of a neural network system, and also distinguishing between various conceptual viewpoints. This section addresses this problem and also discusses various methods which may lead towards increasing the reliability of a neural network system.

There are various key terms which relate to aspects in the field of reliable systems. A *fault* is a physical breakdown, deficiency or flaw in some item of hardware or software. *Errors* occur in information, and are generally caused by a fault in some component. Although faults or errors may exist in a system, they may not be noticed from the systems' actions. When a deviation from expected behaviour is noticed, this is termed a *failure*. *Fault tolerant* implies that a system will continue to perform according to specification in the presence of faults, whereas *robust* is considered to mean that a system will tolerate noise in its inputs. The occurrence of noise affecting a system's information processing (internal noise) should be associated with fault tolerance, rather than robustness, since the cause should have been anticipated in the specification. For example, if limited precision is an outcome when a system is designed, then allowance should have been made, similarly for any possible external agent causing corruption. Two other key terms, reliability and dependability are often confused. *Reliability* is the probability that a system will still be functioning according to specification after a given time period given that it was originally performing correctly. *Dependability* is more a qualitative term relating to the degree of reliance that can be placed on a system with respect to the availability and correctness of the services that it supplies.

3.1. Fault Tolerance and Robustness in Neural Networks

There are basically three main areas that need to be explored:

(1) The capability of neural networks to withstand failure of internal components,

(2) Producing satisfactory results in the presence of noise in input data from the external environment,

(3) Being able to tolerate noise or imprecision between its internal sub-systems.

Each of these areas can in turn be partioned into two further distinct areas that should be investigated. These relate to the (possibly different) effects of noise or failures, firstly, during the training of the neural network, and, secondly, during the execution or real-life use of the tuned system. This hierarchy of exploration domains needs to be viewed at three conceptual levels: the abstract model, the system architecture, and finally the concrete implementation where the architectural design is transformed into reality in some technology.

The examination of abstract models of neural networks is important, and also useful since it may allow new (different) architectures for them to be designed such that the fault tolerance and robustness inherent in the model is fully exploited. Analysis of current architectural models is useful in producing results showing how well it, and possible implementations of the neural network will perform, and again will be likely to lead to better fault tolerant/robust designs,

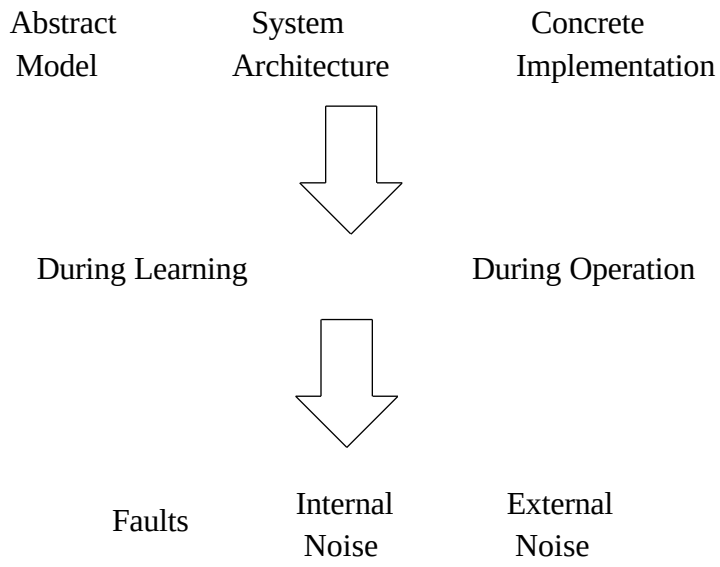


Figure 3.1 Taxonomy for Reliability

though due to the restrictions placed on the system by moving down from the abstract level to the architectural level, the gains possible to achieve will probably be less significant. At the lowest level (the implementation), although very precise aspects of the fault tolerance/ robustness can be analysed, very few changes, if any, can be made to the system to increase the inherent reliability.

Thus, it can be seen that any possible gain in fault tolerance/robustness through analysis of the system is reduced as the conceptual view lowers from abstract to implementation, but also the ease of analysis and description of the fault tolerance/robustness of the system increases. Moving down from the level of abstraction to real implementations, the measurement of reliability becomes easier, but the system is less likely to be malleable to changes to improve this measure.

3.2. Determination of Fault Tolerance in a Neural Network

A requirement exists for a meaningful measurement of the amount of fault tolerance exhibited by a neural network system. For a first approximation, as well as for use in any function that is constructed, the definition of various failure modes will be required, and will lead to a firm foundation for empirical investigations of the reliability of the system. Also, if a suitable reliability function can be found for a neural network, then these failure modes can be applied to it. However, since the component parts of the system depend on each other in a highly complex fashion, it is likely to be difficult to define a function for most neural network models which generates the overall reliability given the reliability of each individual unit/connection. Both of these issues will be more fully addressed in sections 4 and 5.

Several failure modes can occur at the various description levels of the network, i.e. abstract, architectural, implementation. The fault model must cater for both defects and internal noise. At the abstract level, it is reasonable to imagine individual units or connections being either lost or

damaged, this damage possibly causing noise in the output signal of an unit for example. This relates well to the continual loss of neurons/synapses in the brain. The architectural level is impossible to generalise for, and will depend on the individual design. At the implementation level, the failure modes are characteristic of the technology used. The classic is the "stuck-at" fault model which will describe most common failures for most technologies, e.g. short-circuits and open-circuits. An example of a technology dependent fault model is the "latch-up" fault in CMOS.

A consideration that must be taken into account at all levels, is that the most common characteristic of faults is that they are frequently only temporary, often called "transient" or "intermittent" faults. Note that there are two types of temporary faults, transient faults which are non-recurring and intermittent faults which occur given a set of internal/external conditions, i.e. recurring. This latter form of temporary fault can become permanent.

The second requirement which must be resolved is the definition of suitable measures, which give an indication of the effect of the faults induced in the neural network using the fault models defined above. Unlike conventional computers, it is very unlikely that the fault(s) will cause the system to fail completely; it will degrade gracefully, i.e. "fail-soft". These measures may be some function of the expected output and the actual output of the network, though the relative severity of various erroneous outputs must be taken into account, i.e. an output causing possible loss of life compared to an output causing a minor fluctuation may well be similar in "distance" from the expected output, but they are very different in terms of the expected outcome.

3.3. Error Recovery

This is an important activity for any system. Faults do, and will occur in a system over time, and although a neural network may well be inherently fault tolerant and robust, there will come a time when performance is below acceptable limits. Various features of neural networks will cause recovery from errors, others can be easily added to the system to provide automatic recovery. A neural network will naturally recover from "glitches" or temporary faults to provide fault-free operation. If the system involves internal feedback, then it is possible that the process of error recovery will be enhanced since the effects of transient faults will be self-corrected.

As a system degrades due to faults, in appropriate application areas it is better if it settles into a predetermined state which will affect the overall system reliability less severely, i.e. *fail-safe*. By appropriate weak self-excitation or inhibition of units, this might be achieved in neural networks, though safe states would be extremely hard to determine.

Permanent faults that occur due to the loss of connections and units may cause a slight degradation in the system’s performance. This may be purged by the neural network relearning for a short period of time, which should be far less than the original learning time. This will then remove any old permanent faults from the system, though it must be remembered that this discussion also applies to fault tolerance and robustness during the training period. It is conceivable, depending on the training procedure, that this could be done in background, i.e. whilst the system is still executing, either continuously or as the need arises.

To provide further specific error recovery, detection has to be employed to locate faults in the system (implying that the failures caused by the fault are detected) and action can then be taken to remedy the situation. On a unit failure, disconnecting it from the network, either by forcing it to output zero via feedback or employing some re-routing strategy is sufficient. This then allows any "weaker" redundant units to take over. A connection failure will probably affect the entire unit

which it feeds, though only when the unit from which it is fed is active, i.e. a recurring fault. This causes a difficulty when devising a fault recovery scheme. Note that fault location is likely to be extremely difficult in neural network systems.

3.4. Influences on Fault Tolerance and Robustness in Neural Networks

One of the main architectural differences between a conventional computing system and a neural network is that the former uses a few sparsely interconnected, high-performance processing units, whilst the latter uses a large number of highly interconnected, very simple processing elements. With the knowledge that often many thousands of connections are made to any particular neuron, factors leading to fault tolerance/robustness can be examined in neural networks. It is the difference in power that suggests that each element cannot be particularly important in terms of the overall computational system, its very simplicity limits its global capability. Considering also that the "knowledge" of a neural network is viewed as being held in the weighted connections between units, then due to their large number (many more than the number of units), again a particular connection cannot be vital for the system to succeed. In contrast, assuming no explicit safeguards are built-in, a comparatively small fault occurring in a conventional computer usually results in a catastrophic failure.

The principle characteristic of neural networks leading to fault tolerance is the distribution of information or "knowledge" throughout all of the network. When the system is functioning, almost all of the units or connections play a role in producing an output either directly or indirectly. So the loss of a particular unit or connection should not greatly affect the system's function if there is sufficient redundancy. However, local "semantically" meaningful representations can occur, and these are generally called *feature detectors*. This is accepted to mean that a unit will activate only when it sees a pattern on its input's which is sufficiently close to its exemplar. If this unit failed a recognisable decrease in overall system reliability would be expected, however this degradation might be reduced if this was only one of a group of similar "feature detectors" working together in a distributed manner.

In biological neural networks, a neuron will synapse to another neuron many times. In artificial neural networks, pairs of units only have a single connection. It may well prove beneficial to replicate this aspect of biological neural networks as it will ensure that a loss of a connection will be unlikely to disconnect two units, and so cause information flow to be disrupted.

Another major factor which will result in improved fault tolerance and especially robustness, is the choice of input and output representations [35, 9] used, though this is not been comprehensively addressed. If a small group of the input units have noisy or incomplete data, or some of the input/output units actually fail, then it would be extremely useful if this could also be tolerated by the system in a similar fashion to failure in internal units. Indeed, the inputs and outputs form a critical interface to the overall system. For a distributed representation the comments in the previous paragraph are relevant, but when a different representation is employed, e.g. a continuous representation, then the robustness characteristics of the system may be radically altered.

If an input or output representation is topographically ordered in such a way such that cells near to each other denote input or output types that are similar, and also that several cells are devoted to a category, then it is very plausible that reliability would again be improved. Further, if the numbers of such cells are related to the "importance" of that input or output category, then redundancy will be well apportioned. For example, in Kohonen nets [25], if an output unit fails,

redundancy exists in the surrounding units also representing the same class. Note also that if a layer of cells is connected such that nearby units are excited but distant units inhibited (centre-surround cells), then local errors will be masked if the input signal to the layer of units is smooth.

The format of the actual data must be a further consideration [20]. Outputs, and possibly also inputs as well, can be chosen to be real-valued numbers rather than binary. This leads to the possibility of confidence values being used by the network so that that with a suitable choice of input or output representation, they may well improve the robustness of the overall system.

Neural networks which utilise feedback are also likely to alter the reliability since it will add temporal redundancy [37, 50]. However, it is unclear whether errors in internal signals caused either by noise or by failures of units/connections will be damped out given time, or that an erroneous signal will be amplified. Synchronous updating in such a system is likely to decrease the reliability as would be achieved if using asynchronous updating of the units. This is due to the requirement of a global update signal. Similar arguments apply to the various methods, such as random or sequential selection, for choosing the next unit to be updated in asynchronous updating.

Further temporal and spatial redundancy can be gained by employing Time Delay Networks [51] which should improve the reliability of a standard network. Here, a small window scans over the input giving several samples of the input at different discrete times, thus input data is distributed over time in the network. If the input value would not be expected to change greatly over the time period, then using a short time window should improve the robustness of the network to noise, or increase the redundancy of the input representation if the time window is larger.

Finally, the notion of the processing elements in the neural network being simple and of very similar, if not the same design (rather than various complex processors) is an additional influence to the overall reliability and dependability of the system. The reliability of each element can easily be made excellent, thus further improving the overall reliability of the system if required.

3.5. Graceful Degradation

Unlike conventional computers, due to a neural network's distributed computation, it seems very unlikely that faults or noise affecting inputs and internal components will cause it to fail catastrophically, but instead it will probably degrade gracefully, i.e. *fail-soft*. This has been observed in [6, 10, 11, 17]. This feature of neural networks occurs naturally, unlike with conventional computing systems where the design has to address this feature explicitly. Eventually the inherent fault tolerance of the neural network will become insufficient and the performance will fall sharply. Note that this will definitely occur when the outputs become disconnected from the inputs due to internal connection loss. Graceful degradation will also allow a neural network system to be continuously monitored for approaching failure.

3.6. Conventional Fault Tolerant Techniques in Neural Networks

So far, only the inherent (or easily achieved) fault tolerance and robustness of neural networks has been examined. However, there is a large amount of work that has been done on improving the reliability of conventional computing systems, and it is possible that some of the ideas there might be suitable for use in increasing the reliability of a neural network.

In every conventional method for providing better fault tolerance or robustness in a system, extra requirements arise and the complexity of the system is likely to increase. However, to improve the overall reliability, the reliability of the extra components introduced must not degrade the system.

The well-known method for increasing the reliability by use of static redundancy, is that of Triple Modular Redundancy (TMR), or its expanded version, N-Modular Redundancy (NMR). Here a sub-system is replicated N times (N=3 for TMR) and then the outputs of each sub-system are fed into a voter which selects its majority input. An NMR system will tolerate up to (N-1)/2 failures, so N is usually chosen to be odd. The reliability of a NMR system is increased (at least for a time) over a single system. However, it is not clear how a neural network with, say, 3 times the number of units compares to a neural network with TMR applied to each unit. Also, there are certain problems with the NMR system, such as the voter becoming the critical point of the system for failures, and the difficulty of recovery from transient faults.

Dynamic fault recovery by having spare unused units available and switching one in when a failure occurs, is also a possible way to increase the reliability of the system. This implies a method to detect when a particular unit becomes faulty. Again, it is unclear as to whether or not it would be better to use the units in the network, accept failures, but then have a larger distribution of information.

It appears that classical fault tolerance techniques from conventional computing systems do not directly apply to neural networks, which is not particularly surprising due to their novel unconventional information processing architecture, however loose analogies can be made. For instance, the strict definition of *fault tolerant* does not apply to most neural networks since they do not give error-free results at any time, merely an approximation. Likewise, the idea of using TMR as described above in a neural network at the abstract level is simplistic since it is almost impossible to discover individual units which might be vital to the neural network's processing, and also since neural networks are generally considered to process information in a completely distributed manner, applying TMR to each unit would be prohibitively expensive. Note that at the implementation level, TMR may well be a method to further increase the reliability of critical components in a neural network system. The methods in reliable software design cannot generally be applied to neural network systems since they do not run a program in the same way that a conventional computer system does, though if a software simulator is implemented on such a system, then techniques for improving the reliability could well be used in it.

3.7. Conclusions

It has been seen that neural networks hold a great potential since they are systems that are both inherently fault tolerant and robust. This comes from the distributed nature of information, both in internal structure and the input/output representations. Faults are masked from the external environment, rather than being detected sometime later and then corrected, hopefully also correcting all causal errors as well. As failures occur in the neural network, the overall performance should gracefully degrade for a period, so improving the dependability of the system. It is clear that mathematical models for describing the reliability of a neural network and methods for empirically testing it need to be developed. The next two sections address these requirements.

4. Fault Models

Fault models are an essential aid in determining the reliability of a neural network system. They describe the types of faults, and where and how they will occur in a system. However, they become more difficult to formulate sensibly as a system is viewed at an increasingly more abstract level, especially the definition of *how* a fault manifests itself. It will be shown below how sensible locations for faults can be defined in a neural network viewed at the abstract level, and then the complex problem of detailing the fault will be approached.

A fault model lists which components can become defective in a system, and also when and how they will misbehave. Such entities need not necessarily physically exist, but may be abstractions of real-world objects. In general, a fault model is an abstracted representation of the physical defects which can occur in a system, such that it can be employed to usefully, and reasonably accurately, simulate the behaviour of the system over its intended lifetime with respect to its reliability. Four major goals exist when devising a fault model:

(1) The abstract faults described in the model should adequately cover the effects of the physical faults which occur in the real-world system.

(2) The computational requirements for simulation should be satisfiable.

(3) The fault model should be conceptually simple and easy to use.

(4) It should provide an insight into introducing fault tolerance in a design.

However, these four requirements often conflict with each other resulting in the fault model being compromised. For instance, simplicity, which leads to lower computational requirements, may result in an inaccurate model if carried to excess.

Several levels of abstraction exist for visualising neural networks (see figure 4.1), namely the abstract, architectural and implementation levels. These relate in a complex manner to the levels of abstraction (electrical, logical and functional) for which fault models are defined in digital systems [22]. Considering only the implementation level, all three fault model abstraction levels can be applied, just as for conventional computer systems. For the architectural level, only the logical and functional abstraction levels of digital systems apply. Finally, the abstract visualisation level for neural networks can only be related loosely to the functional level of digital systems. In general, just as for the levels of abstraction for viewing digital systems, it becomes progressively harder to define good fault models as one moves from the implementation level to the abstract level for neural networks. For example, in digital systems it is impossible to model current leakage at the logical level, but it can be modelled at the lower electrical level. Restrictions also arise due to the simplification of continuous electrical parameters to logical values.

Concepts from the electrical and logical levels could be used to probe the reliability of neural networks. However, the objective here is to investigate the inherent fault tolerance of neural networks which arises from their distributed computational nature. It is not to directly examine the characteristics of physical implementations, either at the architectural or implementational level, where reliability will be influenced by the physical components and fabrication techniques employed. Also, the implementation and architectural levels are too specific; technologies change and architectures are numerous. This investigative direction will allow implementations of neural networks to be designed in such a way so as to retain the inherent fault tolerance within the model, as well as to enhance it by means of standard fault tolerance design procedures. Examining neural networks at the abstract level suggests that the definition of a fault model is likely to be difficult, and also at first view, there seems to be a conflict with the requirement that a fault model should adequately cover real-world defects. However, although it is possible to identify reasonably the

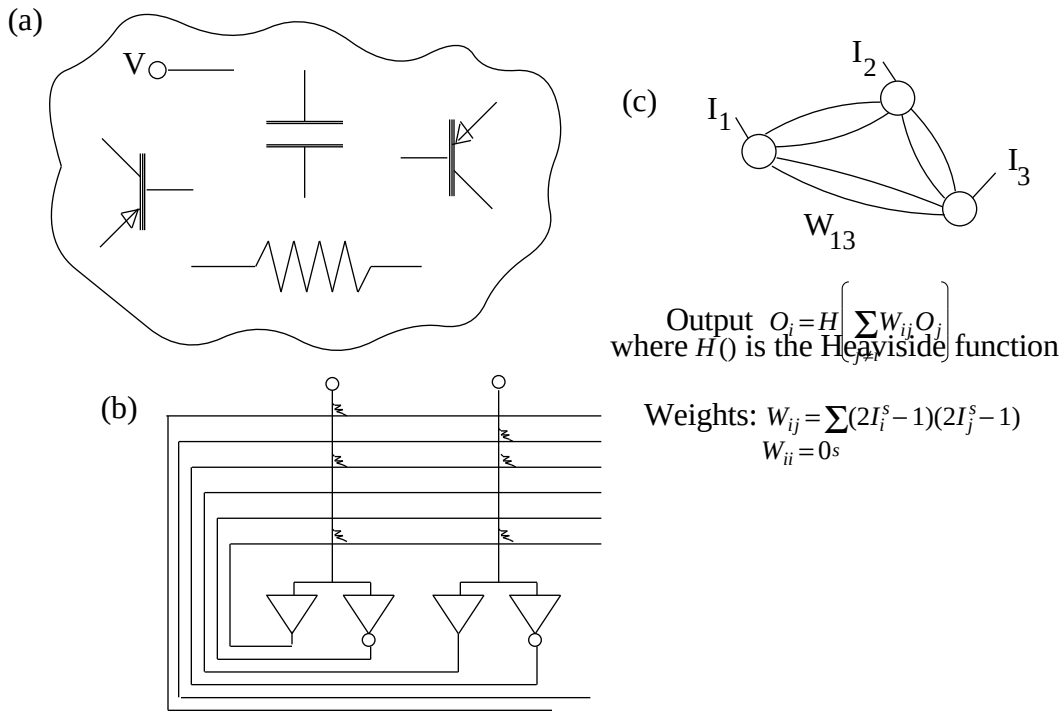


Figure 4.1 Visualisation Levels for Neural Networks

potential faulty entities in the abstract model of the neural network, reference occasionally must be made to implementation aspects of possible designs when describing the nature of their deviation from proper behaviour. A further goal in the development of a fault model is to disassociate it from any particular class of neural network. This will allow comparisons between results from such models, thus indicating their relative merits.

The fault model once defined can be used for two purposes. Firstly, if it covers physical faults satisfactorily, then it can be used to generate a test procedure to ensure that implemented systems work according to specification. Secondly, the fault model can be used in simulations of the neural network system to evaluate a measure for its reliability. It is this latter case which is of interest here. Two approaches exist for developing a measure for reliability in a neural network system. The first is to use measures from existing reliability theory and apply them to neural network systems, the other is to develop new measures. Both of these avenues will be explored further in section 5. The overall objective is to define measures which are generic in nature; they should apply across a wide range of neural network architectures such that valid comparisons can be made between them, and again, as for the fault models, simplicity and ease of use should be major considerations.

Measures for reliability should not be confused with measures for fault tolerance; reliability and fault tolerance correspond in some areas, but are are unrelated in others. Fault tolerant design methods are a technique employed to improve reliability. The definition of reliability is the probability that a system does not suffer a *failure* for a time period T, given that it was working correctly at time $t = 0$. Clearly it is perfectly possible for faults to be suffered without diminishing reliability if they do not cause a failure during time T. Also, the reliability can be diminished not

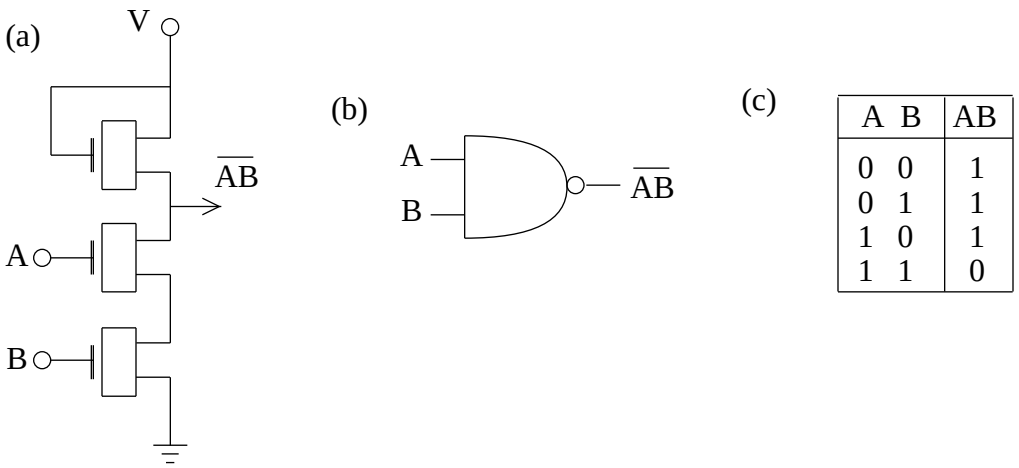
by faults, but due to the system not meeting its specification and failures resulting because of this. Conversely, fault tolerance characterises how the system behaves as faults are introduced into it. High fault tolerance indicates that the system will not be adversely affected by faults, whereas low fault tolerance implies that it will be very sensitive to any faults which occur. Hence measures of reliability do not strictly assess the fault tolerance of a system, but, if the system is correct (i.e. it meets its specification), they can give an indication of the effects of faults by the length of the time period before the reliability begins to decrease. Also, any faults will always influence a neural network's output due to their processing manner being both distributed and parallel; all components are involved in any computation. This is in contrast to a conventional computer system where a fault will only become a factor when that part of the system is used, except for common mode faults.

The distributed computational nature of neural networks seems at first to present a serious drawback. Although faults will always identify themselves to some degree, they cannot be located easily. In an implementation, each component would require extra circuitry to detect and signal the occurrence of a fault, and the cost, as well as the possible reduction in overall reliability of the system, would probably render this approach unsatisfactory. It seems that this would constitute a major disincentive to the use of neural networks for reliable computation. However, neural networks also have another important property, they can learn. This feature will allow a faulty system, once detected, to be retrained to either remove or compensate for the faults. The relearning process will be relatively fast compared to the original learning time since the neural network will only be distorted by the faults, not completely randomised.

4.1. Conventional Fault Models

In formulating the locations for faults in an abstract visualisation of a neural network, it is helpful to first examine existing fault models for conventional digital systems. Considering a system at its most basic level, the physical faults which occur depend upon the fabrication techniques used to implement the circuit, such as TTL or CMOS for example. A few examples of such physical faults for the latter technology are defects in the silicon, short circuits in metal, and holes in oxides used in transistors. These very real faults are modelled by some more abstract representation in the related fault model such that both accuracy and simplicity are hopefully achieved. Three levels of abstraction tend to be considered for viewing systems, each with its own associated class of fault model.

At the very detailed electrical level, example definitions of faults are changes in various continuous variables such as voltage, resistance, and current levels. Next, the logical level only considers signal values which map to the logic (and discrete) *0* and *1* values, and the corresponding faults are similarly more abstract. The faults defined at this level include, but are not limited to the well-known *stuck-at* faults, e.g. *stuck-at-1*, *stuck-at-0*. Although the faults defined in the logical fault model are a grossly simplified version of the physical faults which actually occur, they do bear a reasonably acceptable functional relation to them. Also, computational costs are reduced, though they are still considerable for present day circuit sizes. However, physical faults such as current leaks and threshold voltage shifts cannot be represented by the logical fault model. Finally, at the functional level the fault model is defined using high-level information, such as input/output specifications and circuit diagrams for example. This highly abstract fault model is required when very large circuits are being considered or when no



A	B	AB
0	0	1
0	1	1
1	0	1
1	1	0

Figure 4.2 NAND gate at 3 levels: (a) MOS (b) Logic (c) Truth table

information is available on the internal structure of the circuits' components. It may also be used when the computational requirements of using a reasonably abstract logical fault model are far too great. The very high level of abstraction results in the functional fault model being implementation independent, but often also very imprecise and heuristic in nature. See figure 4.2 for an example of a component represented in electrical, logical, and functional form.

The quality of fault models are very variable. Although some may be uncomplicated and conceptually simple to apply in simulations, they may not be particularly accurate with respect to the faults a system would suffer in actual use. Although it is obviously beneficial to have the former characteristics, the latter feature should be treated as a primary objective. This then leads to a measure which has been used to indicate a fault model's quality. *Fault coverage* is defined as the percentage of physical faults which are identified by the fault model. However, this value is often very difficult to determine.

4.2. Fault Locations

As each level (electrical, logical, functional) in a digital system requires a corresponding fault model, the case is similar when considering fault models for neural networks. When they are viewed from the implementation level, the conventional fault models above can be employed, and for the higher architectural viewpoint, a functional fault model is suitable. However, fault models for the implementation and architectural levels cannot be specified generically since they are highly dependent upon the design (e.g. the technology used), though standard existing fault models for individual components (e.g. diodes, transistors, shift registers, etc.) are envisaged as building blocks in developing such a fault model. However, the objective here is that of formulating a fault model for a neural network visualised from the abstract level. As a further goal, the fault model should apply across a wide range of neural network models. It is possible that functional fault models could be used since are independent of system implementation, though they may be limited since the abstract description of a neural network differs widely from model to model. Functional fault models applied directly to abstract neural network models will be discussed later.

Examining the application of more conventional fault models to neural networks viewed at the implementation and architectural levels illustrates how locations for faults in a abstractly defined neural network can be found. By noting the common features, and then extrapolating from these observations, it will be shown how new fault models can be devised for the abstract visualisation level.

At the implementation level the electrical fault model involves physical objects such as connection wires, capacitors, and transistors. The logical fault model refers to the slightly more abstract signal lines which interconnect logic gates, and it is these entities which are chosen to be possibly faulty. For example *s-a-0*, *s-a-1*, short-circuited, etc. Next, at the higher architectural level, components such as individual IC's or communication lines are chosen to be candidates for fault locations.

In both cases it can be seen that it is *atomic entities with respect to the conceptual level at which the system is being viewed*, or interaction of a few such atomic entities, which are considered to be eligible as locations for faults. Also, these entities can be seen to be either acting as functional units or information channels. So, just as the chosen elements at these two conceptual levels which construct the fault model cannot be subdivided, similarly at the abstract level, the entities selected from the mathematical model should also not be capable of being fragmented in terms of their role. For example, weights, links, and threshold functions could all be candidates. This provides a basis for a fault model which is independent of any possible implementation, and simulation results should indicate the fault tolerance *inherent* within the neural network model, i.e. that which arises as a consequence of the nature of the computational method of neural networks.

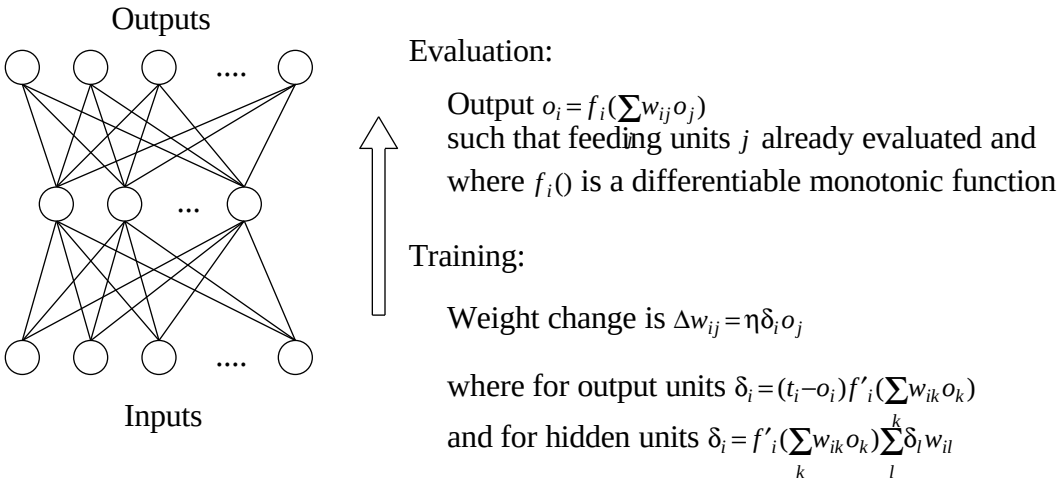


Figure 4.3 Multi-Layer Perceptron Neural Network

4.2.1. *Example*

As an illustration of the above discussion on the determination of fault locations, the multi-layer perceptron neural network architecture [41] will be considered, and reasonable fault locations will be identified from the abstract model. The description of this abstract model is given in figure 4.3

which shows a graphical representation of the neural network and the mathematical equations governing the system.

Faults should be considered for both the operational and training phases of a neural network, though only the former is vital when operational systems are going to be "cloned" from a single once-only trained neural network. However, this is not the case for an autonomous system employing neural networks since learning will be an active function throughout its lifetime. For completeness, both phases will be considered here.

Two broad types of fault locations can be identified, *stable* entities whose associated information does not change at any time, and *temporary* entities whose information is valid only for a limited period†.

Stable items from both the operational and training phases are:

Weights, not only for the operational phase where they are fixed values in the multi-layer perceptron network once training has finished, but also for the training phase. This is since they are a fixed base for the weight change Δw_{ij}, and it is this that is the temporary element (see below). For simplicity, bias values θ_i are viewed as weights on connections from a dummy unit which is permanently active.

Threshold Functions, a fault in a threshold function will alter the transformation of the activation to an output value in some manner. This will obviously affect both phases.

Differential of Threshold Functions, this fault will only affect the system during the training phase. It is identified as a separate fault since its function is generally different to that of $f_i()$.

Constant Values, faults affecting any constant values are fixed by definition. During the training phase an example would be the learning rate η.

Target Values t_i, these are not included in the constant values above since it is conceivable that a MLP network may be trained on data that is modified as time progresses (e.g. Miikkulainen and Dyer [35]). However, they are defined as stable items since this change is external to the system.

Topology, the connectivity of the neural network could easily be subject to faults in various ways.

Temporary items from both the operational and training phases are:

Output Values o_i, for the dummy input units, faults are merely some form of external noise, but for the output and hidden units, their output values will obviously be different for various input patterns.

Activation Values $a_i = \sum_j w_{ij} o_j$, similarly as for output values.

Delta δ_i, faults in these are only relevant during the training phase.

Weight Change Δw_{ij}, as mentioned above, these are the alteration for the stable weight base value, and similarly as for δ_i, faults are only applicable during the training phase.

Note that the concept of a "unit" becoming faulty is not specified above, it is only a further abstraction from fault locations such as the threshold function, output values o_i, input weights w_{ij}, etc. This is analogous to a "stuck-at" fault in a digital circuit covering many (more concrete) physical faults.

---

† These should not be confused with the classification of faults as permanent or transient.

4.3. Fault Manifestations

Although the entities causing faults have been identified at the abstract level from the mathematical model of a neural network, the actual nature of the fault they suffer has yet to be defined. For instance, a threshold function might be said to saturate (i.e. output one of its extreme values), a link in the topology might be lost, a weight might be distorted in some way, etc. The latter example is especially difficult to define sensibly since it is uncertain as to what form of distortion might reasonably occur in the abstract universe. It is due to the abstract nature of the conceptual level at which the neural network model is being viewed that this difficulty arises. Two directions which can be followed for defining fault manifestations are either to look solely at the abstract description of the neural network, else fault locations can be related to high-level implementation details or physical components. The details of the fault can then be extracted from these comparisons and any constraints that arise.

The use of the first guiding principle can be approached by defining the manifestation(s) of a fault to be such that the maximum harm is caused to the system by the fault. This will capture all possible lesser manifestations, whether likely or unlikely. The notion of *maximum harm* will depend specifically on the component's context for which the fault is to be defined. In some cases, several manifestations may suggest themselves, for instance, due to symmetry such as in the sigmoid thresholding function.

The alternative concept of considering the faults which could occur given certain implementations restrictions will obviously degrade the independence of the fault model from implementation. However, in certain cases it may be possible to minimise this by developing fault models whose construction only relies on an abstracted view of possible implementations. This knowledge can then be used in fully defining the fault either by using the maximisation of damage principle given above, or relying more heavily on information derived from possible implementations considered already.

4.3.1. *Example*

Using these concepts and the fault locations already identified previously, the fault model for the multi-layer perceptron neural network can be fully defined. First, the various permanent entities which were chosen as fault locations will be discussed, then the temporary entities. It will be seen that no absolute fault model can be developed, only a general framework. From this a fault model can be selected according to requirements such as the degree of implementation independence, simplicity leading to computationally feasible simulations, etc.

The failure modes of the threshold function $f()$, when considered from the abstract viewpoint, can best be defined by examining its graph (see figure 4.4a). The clear symmetries in the threshold function suggests three possible failure modes. The first two relate to the well-known *stuck-at* faults, and are defined here to be *stuck-at-minus-one* and *stuck-at-plus-one*. These follow the concept of drawing slightly on implementation details. Maximisation of damage suggests the third failure mode, and this is dynamic, rather than the static *stuck-at* faults. The threshold function is defined to saturate to $+1$ when the fault-free output would be less than zero, else -1 otherwise. This is a rather harsh fault during operational use, since it implies that the associated unit always outputs the incorrect value irrespective of "how sure" it is, i.e. how large its activation is, and hence how close its fault-free output would be to ± 1. It could possibly be modified to take account of this by making the fault probabilistic. Note that during training, although more time will be

required to teach a neural network if a unit permanently reverses its threshold function, it is still very likely that it will be able to learn the training set to the same degree if no such fault had occurred. The faults considered so far only apply to a continuous or analogue system. It is also possible to define failure modes for a digital form of the threshold function (see figure 4.4b), though of course, this implicitly introduces reliance by the fault model on implementation details. Example failure modes include elements of the digitised function being corrupted, either randomly, or, by following the concept of maximum damage, set to the opposite extreme of their fault-free value.

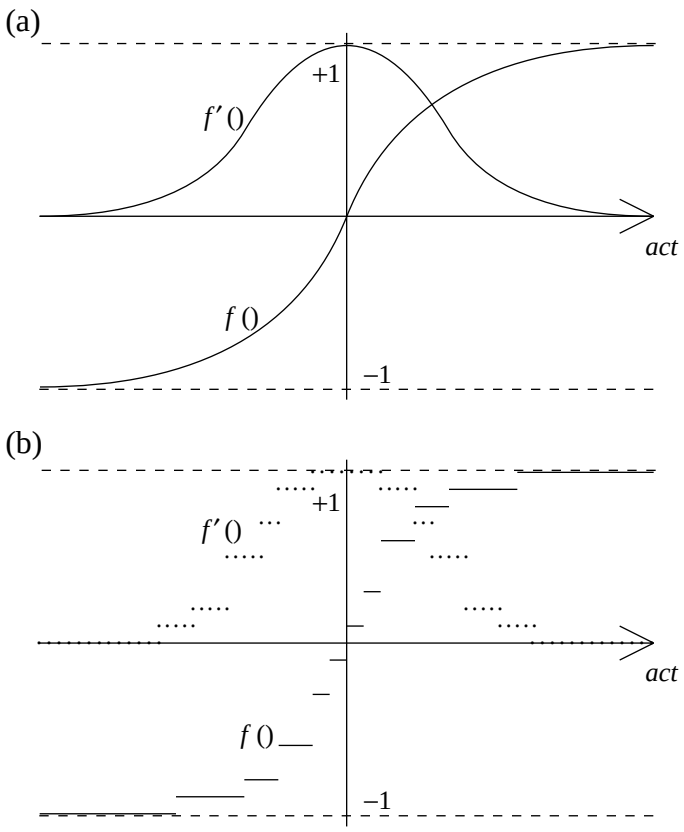


Figure 4.4 Graph of Threshold Function: (a) continuous, (b) discrete

The failure modes associated with the differential of the threshold function $f'()$ are similar to the above. Since the graph peaks at $+1$ when $act = 0$, and is symmetric about this point (see figure 4.4), the two *stuck-at* faults should be *stuck-at-plus-one* and *stuck-at-zero*. To maximise the damage, the output should be $+1$ as the fault-free value tends to 0, otherwise it should be 0. The point of change could be defined to be where the sign of the curvature alters. Similar failure modes could be introduced for a discrete version of the function as described above.

Faults which affect the weights w_{ij} in a neural network are very hard to define sensibly. Using "vague" implementation information, a classification can be identified. The weights can be considered as being held in a discrete form, such as binary encoding, and then individual components can be corrupted in a similar fashion to that for the discrete threshold function. The other alternative would be storing weights using a continuous representation such as resistors for

example. The model of the fault can then be based on the fault characteristics of the component(s) used. For example, a resistor is likely to either go open-circuit which can be modelled by causing the weight to saturate to its maximum value, else it will become noisy which can be modelled by adding noise from a Gaussian distribution.

Purely abstract failure modes can also be defined for weights, and as such, they will be independent of any implementation. Two very simple failure modes would be to either set the weight to zero, thus causing the loss of any partial information (due to the distributed nature of neural network processing) that the weight held. The other following the concept of maximising damage suggests that the weight could be multiplied by –1. This latter failure mode can also be derived (with a slight modification to decrease its rather fierce nature) by examining the activation equation:

$$act_i = \sum_{j=1}^{n} w_{ij} o_j - \theta_i \qquad (4.1)$$

$$= \underline{W_i} \,.\, \underline{O} - \theta_i$$

The vector $\underline{W_i}$ is normal to a hyperplane in *n-D* space positioned at θ_i, and classifies vectors $\underline{O}$ into a dichotomy depending upon which side of the hyperplane they fall. Following the notion of causing maximum damage, the failure mode of the weight should be chosen such that the probability of any vector $\underline{O}$ being misclassified is maximised. i.e. if $\underline{W_i}'$ is the faulty weight vector, then for a particular input vector $\underline{O}$:

$$\underline{W_i} \,.\, \underline{O} > \theta_i \;\Rightarrow\; \underline{W_i}' \,.\, \underline{O} < \theta_i \qquad (4.2)$$

$$< \theta_i \;\Rightarrow\; \underline{W_i}' \,.\, \underline{O} > \theta_i$$

For the first case (the second being similar), say w_{i2} is faulty, then for input vector $\underline{O} = (o_1, o_2, \cdots, o_n)$ to be misclassified:

$$-w_{i2} \geq \frac{1}{o_2} \left(w_{i1} o_1 + w_{i3} o_3 + \cdots + w_{in} o_n - \theta_i \right) \qquad (4.3)$$

Since o_i is a continuous value over the range of the threshold function $f()$, defined here to be the interval $(-1, +1)$, this implies that for o_2 very small, w_{i2} will have to be very large to cause an incorrect classification, in general:

$$w_{i2} \to \infty \quad as \quad o_2 \to 0 \qquad (4.4)$$

Even disregarding the size of o_2, a single weight would still have to be of large magnitude to dominate all of the other inputs to the unit for many input vectors. So, this fault definition is clearly too severe since the result of one weight being faulty will cause the overall unit to always give the incorrect answer. This destroys the notion of high fan-in causing individual inputs to be unimportant globally. Also, it would be unlikely that an implementation would allow potentially infinite weights, so this suggests a constraint can be applied setting a saturation limit on any faulty weight. If $\underline{O}$ is discrete, then this constraint will still apply.

The topology of a neural network was another area identified as potentially being affected by faults, and manifestations need to be defined. An obvious fault is the loss of a connection between two units, and this relates to the loss of an arc in a directed acyclic graph (DAG) which abstractly represents the topology of a neural network. Another possible failure mode would be to randomly reconnect a link to another unit in the neural network (possibly due to a short-circuit), though this

fault would be far less likely than the simple loss of a link. However, the consequences of this type of fault would be more severe than the first since the nature of the neural network might be completely distorted, e.g. the MLP becoming a feedback network and so possibly non-deterministic.

The final permanent entities which were classed as fault locations are various constants such as the learning rate η, and the target values t_i. Constant values tend to be chosen from a limited interval, and the definition of how the fault will affect them will depend upon their function in the case of trying to cause maximum damage. For the learning rate η, extreme failure modes would be to set it to zero or to its highest possible value. Its value will typically be in the range $(0, 1]$. A fault affecting a target value t_i could be that its value is changed to be opposite to that in a fault-free situation, thus maximising damage to the neural network's learning. Target values generally only take the two values at the extreme ends of the threshold function range. As a less severe fault, randomisation either supplying an offset or a new absolute value might be more suitable in both cases. Considering general implementation details, constants or target values might be encoded in binary form or produced by dedicated hardware, and so similar failure modes could be used as described above for the weights.

Temporary entities are almost exclusively intermediate results, and faults effecting them will either act as a persistent bias to new values or cause them to become permanently fixed. Due to their temporary nature, it is especially difficult to define possible fault manifestations for these entities. Considering the abstract model, they will either have to be stored in short term memory where they will be used in future steps of the algorithm, or else they may merely be signal information. Implementations will generally have to store temporary entities (i.e. latched) due to timing considerations.

The entities identified in the MLP network are the output values o_i, activation values act_i, delta values δ_i, and weight changes Δw_{ij}. The output and activation values are required throughout both the evaluation and training stages, though delta values are required only in the latter stage, but all will need to be temporarily stored. To cause maximum damage, their respective failure modes would be either to limit them to their opposite value, possibly constraining the activation and delta values to some limited range, else to apply some randomisation process to them in a similar manner to that for the weights (see above). Weight changes need only be considered if they are required to be temporarily stored, such as when momentum is used during training, and then their failure modes will be similar to those already given.

4.4. Functional Fault Models

The functional fault model for conventional digital systems offers independence from implementation details, though often at the expense of exactness and completeness. For combinatorial circuits, the faults are represented by modifications to the truth table, and similarly the state transition table for sequential circuits. When considering higher level components (e.g. RAMs) as the atomic entities of the circuit, more complex descriptions than truth or state transition tables need to be employed, and generally, some formal descriptive language embedding boolean expressions is used [22]. Since the majority of neural networks are of a continuous nature (rather than the logical *1* and *0* of digital circuits), such methods are not applicable. However, for Boolean neural networks [24] they can be directly applied since each unit can be viewed as

performing a fixed boolean expression which can be described by a truth table. If the neural network involves feedback, then obviously a state transition table must be used. However, such functional fault models are generally only suitable for testing systems for faults rather than acting as a model to aid in the simulation of a system to identify its fault tolerance characteristics. Also, for large systems, the computational requirements quickly become impracticable.

4.5. Spatial and Temporal Considerations

It is important to consider the relative merits of each fault in the system. If a fault only occurs in relatively few places whilst another is widespread, then so long as the former does not occur with very large probability with relation to the latter, it would be acceptable to disregard it. For example, in a RAM chip the amount of circuitry for the actual storage of bits far outweighs that for addressing, so when simulating the chip to examine its reliability, the addressing circuitry is considered to be fault-free. Since in a large neural network the number of weighted connections is likely to be far larger than the number of unit associated entities (e.g. threshold functions), then it would be reasonable that these could be ignored during investigations. However, if a fault which occurs with low probability has catastrophic effects, then it should be included.

Faults can be classified by two temporal characteristics, they are either *permanent* or *transient* in nature. The most frequently occurring case is the latter [31], and it can be further subdivided into "transient" and "intermittent" categories. Transient faults are non-recurring, but intermittent faults occur given a set of internal/external conditions, i.e. recurring. This latter form of temporary fault can become permanent. Note that stable and temporary fault entities should not be confused with permanent and transient faults; no connection exists. Both entity types can suffer both types of faults, though obviously the effects are likely to be different in each possible combination.

Due to the relative domination by transient faults, it suggests that any simulations or theory developed for neural network models should be based on these occurring, though this will only become especially relevant for feedback neural networks.

The effect a fault has on a system will also depend upon when it actually occurs with respect to the system’s operation. For example, a fault affecting a weight which occurs sometime between the forward and backward passes of the back-propagation algorithm for multi-layer perceptron neural networks will have different consequences if it happened to occur at the start of the forward pass.

4.6. Conclusions

The measure indicating to what extent the fault model captures the multitude of physical faults that occur in an implementation is termed *fault coverage*. To evaluate the coverage of the fault models which have been discussed in this section, their two aspects of fault location and fault manifestation need to be considered separately. Since the aim was to develop fault models for neural networks visualised at the abstract level, the location of faults in the fault model cannot easily be related to that which would occur in any possible implementation, and so the fault coverage is hard to determine. Obviously, when "vague" implementation details are considered in defining the failure modes this is improved, but implementation independence is decreased. Although the abstract failure modes may not have such a large fault coverage, they will be

implementation independent. For both of these fault models, and also the briefly mentioned functional fault models, the fault coverage is generally very hard to determine. However, the abstract nature of the fault model increases the possibility of them being generic in nature, due to the independence of implementation. The fault models developed here can now be used in the process of measuring the reliability of a neural network system, and this is the area covered in the next section.

5. Assessing Reliability

A basic requirement for almost all systems is some knowledge of how long a system will continue to function correctly. The reliability of a system depends upon a number of factors such as the environment in which it will be used (e.g. spaceborne as opposed to an air conditioned computer room), the design of the system which includes the quality and type of parts used, fault tolerance techniques employed, and quality control during assembly. All of these factors are related in a complex manner to each other involving many trade-offs and mutual reinforcements. However, since neural network systems are only being considered abstractly here, their inherent fault tolerance (which is one factor for reliability) can be observed by investigating their reliability. Only in an actual implementation will the other factors will become relevant in determining the reliability of the system. However, although the emphasis will be on abstract neural network models, the reliability measures discussed will be equally applicable for implementations in producing results, though for some methodologies, such as fault injection for instance, it may be difficult to do so due to physical limitations.

Although it appears that neural networks do seem to exhibit some inherent fault tolerance [13, 40, 45, 47], a needs exists for a *generic* approach towards measuring just how fault tolerant such a neural network system is. This will allow comparisons between various neural network architectures, and also hopefully various models as well. Two methods which could supply the required assessment for a neural network system are Fault Injection and Mean-Time-Before-Failure. Such techniques for assessing reliability as these, as well as others which may be developed in the future, all require a detailed description of the faults which can occur in the neural network system which is being investigated. The fault models described previously in section 4 will be used to meet this requirement.

5.1. Failure in Neural Networks

The nature of neural network's style of computation does not lend itself to applications requiring exact and precise answers, rather they are suitable for *soft* problem areas. This means that failure will likewise be an imprecise event in most situations. The assumption of failure in conventional systems being a discrete event is not realistic for neural networks. This implies that the measurement of a neural network's degree of failure must be done in a continuous manner. This is likely to be very difficult since they are essentially a black-box and so their functionality can only be judged from their interfaces. Thus measures which indicate the reliability of a neural network can only use external information such as inputs, outputs, training data, etc. Although specific measures may suggest themselves for particular neural networks, more generic measures can be defined by considering various characteristics of neural networks.

5.1.1. Measuring Failure

Neural network models which use some form of continuous threshold unit do not compute definite, clear-cut answers to problems presented to them, but instead their output merely indicates a tendency for a particular answer, and so the question of whether a neural network has failed is hard to address. This problem is made worse still if the neural network exhibits graceful degradation[†] since the output units will not suddenly change in value, but rather will slowly degrade towards uncertainty. To define the failure of one of these units, a continuous measure

must be employed which reflects either the degree of certainty in its response with respect to the wrong answer(s), or else the uncertainty in its response with respect to what the answer(s) should be. Note that this includes neural network systems which use their output units to indicate confidence since reliability measures relate to failure, and only indirectly to faults. In this case, as an output unit degrades towards increasing uncertainty, failure occurs with respect to the specification, and so will be detected by the reliability measure. However, the increase in uncertainty may be due to the input presented to the neural network and not caused by faults.

Conversely, for neural network models which require output units to be either on or off (i.e. discrete valued rather than continuous representation), generally a Heaviside function is used. These are possibly substituted for sigmoid threshold functions in the output layer if used during training. To gauge failure in these units, the variable which should be used is the activation, and then a similar method can be followed as above for continuous threshold units. Activation must be considered since the thresholded output value does not indicate where a unit falls between the extremes of absolute certainty (saturated activation) and near uncertainty, that is, in the worst case a unit may be on the verge of misclassifying an input.

However, output representations can be redundant, and so the overall degree of failure in the output units considered as a whole will be reduced, possibly completely. So, any measure for the degree of failure of the neural network must not solely consider failure of output units individually and independently, but must also take into account this data representation redundancy. It might be argued that if the output representation is redundant, then the degree of failure of individual output units can be disregarded and only the entire output vector considered. However, unless it is possible to measure in a continuous fashion how close the redundant output is to the critical point where the redundancy becomes insufficient to mask multiple partial unit failures, i.e. the redundancy is not hidden, the output units must still be considered individually as well. Another reason to consider only the entire output vector (or subgroups of it) is if an output representation is used which defines the neural networks response as an interpolation between the output levels of several adjacent output units [30].

As well as the above, for applications which require a stream of outputs from a neural network system (e.g. controlling a dynamic system) rather than just presenting a single input to obtain a result, qualitative aspects of their function must also be taken into consideration when evaluating the degree of failure of the system. For example, a neural network which balances a pole may do so in many different equally successful ways, one of which might require very gentle motions to keep the pole balanced, but another might involve large forceful oscillations to do so. There is a clear qualitative difference between them, but a quantitative measure is required which will take account both of these differences and also of how correct the output is, irrespective of application or neural network model.

All of these factors must be combined together to produce a function which will supply a continuous value indicating the overall degree of failure within the neural network. To summarise, correctness of output must obviously be incorporated, this must take account of the appropriate value attribute of individual output units with respect to target values, and also the overall output vector due to possible data representation redundancy. To include information on the degree of failure in a dynamic system, the derivative of the output of a unit can be used to indicate fluctuating behaviour, and some measure of deviation to capture extreme swings. Both of the latter

---

† The property of a faulty system continuing to deliver an acceptable, though possibly reduced, performance is termed *graceful degradation.*

values are needed since fast small changes or slow large changes would not be adequately detected by either on its own. The actual way in which these various factors are combined will depend upon the application, focus of interest, etc.

5.1.2. Applying Failure Measures

To detect failure in a system, the monitor must have pre-knowledge of the correct processing results for any input presented, and all of the above techniques for measuring the degree of failure have implicitly required this. Generally, it is possible either to specify exactly the mapping which the neural network is supposed to have learned, else a suitable test set can be constructed which reflects the nature of the input domain of the problem. However, for neural networks which are required to generalise and where the mapping cannot be exactly specified, this test set may be more difficult to construct. In cases where an acceptable test set cannot be formed, the failure measure adopted can be determined by characteristics of the application area, though this will greatly reduce its generality.

Since neural networks are black-box systems, the function for measuring the degree of failure can only judge them based on the results at the output units for presented input data. Hidden units cannot be used. The choice of this input test data may be critical for certain applications, e.g. a neural network may not generalise correctly in a particular input region, and so cause a failure which can only be discovered if an input is presented to the neural network from this incorrectly generalised region of input space [5]. However, such failures will only result from deficits during training, or perhaps due to faults in units which act as specific feature detectors. Any faults occurring during operational use will cause an identifiable change in the output independent of the input presented since neural networks process their inputs in a distributed and parallel fashion; all components are actively involved in processing any input presentation. This is unlike conventional computer systems where a fault may only cause a failure for a specific input, and so the selection of a test set can be extremely difficult. The problem of choosing a wide-ranging input test set for neural networks is not so critical, though if reliance is placed upon generalisation, then difficulties may arise.

5.1.3. *Example*

For the multi-layer perceptron network (see figure 4.3) the definition of failure is based on the existence of a training set composed of pairs of input and output patterns. Two cases exist for the definition of failure depending upon whether generalisation is required. Note that if generalisation is relied upon then the training set should adequately sample the input-output space.

First, if generalisation is not required, then the distance of the output pattern $\underline{o}_p$ to the nearest incorrect target pattern $\underline{t}_q$ can be considered. For failure not to occur,

$$\forall p. \left(\|\underline{t}_p - \underline{o}_p\| < \forall i \neq p. \|\underline{t}_i - \underline{o}_p\| \right)$$

The Euclidean metric $\|\underline{x} - \underline{y}\|$ could be used to determine the distance, though other metrics could be substituted as appropriate.

However, if generalisation is required, then a threshold HD can be set on the maximum distance that the actual output pattern $\underline{o}_p$ can differ from the correct pattern $\underline{t}_p$.

$$\forall p. \|\underline{t}_p - \underline{o}_p\| < HD$$

The concept of a distance threshold HD has analogies to *basins of attraction*, and should be set to a fairly small value if generalisation is heavily relied upon. It should certainly not exceed the minimum distance of any pattern to another in the training set.

If the MLP is required to exhibit some degree of generalisation, then the target values should be augmented by additional input-output vectors which were not used in the training set, and represent suitable choices for testing required generalisation properties. There obviously exists a trade-off between degree of coverage of the input-output range and the available simulation resources which may be severely taxed by large test sets.

5.2. Timescales

Some techniques for assessing the reliability of a neural network will require the concept of time to be defined. For instance, so that fault rates can be specified, or so that the time before failure occurs can be measured. The choice of timescale (e.g. real-world seconds, CPU seconds, number of transactions, etc.) is determined by various factors, which are often in conflict with each other. Generally, the timescale should relate sensibly to the characteristics of the application area, and to a lesser extent to the neural network architecture used and the method of implementation. For instance, a choice of measuring time in real-world seconds might be suitable for a neural network system controlling some dynamical system, but not for a classification application area where time would be better given the units of number of patterns presented. Similarly, it would not be suitable to choose real-world seconds for a software simulation of a neural network, CPU seconds or number of transactions would be better. However, where a neural network model takes a non-deterministic number of iterations to process an input (e.g. the Hopfield model), the units of time cannot be based on a transaction count, but must rather be related to the number of iterations performed by the system in evaluating an output, i.e. a measure that is invariant to external controls or influences.

Not only must the timescale provide a suitable base from which to assess a particular individual neural network's reliability, it must also allow valid comparisons to be made between various different systems. These may or may not be based on the same neural network model, and may even be non-neural systems. This means that the timescale chosen must also take into account various factors such as the architecture and implementation of the neural network model (e.g. evaluation algorithm, internal components, etc.).

When comparing similar neural networks based on the same architecture all performing the same task (e.g. MLP's with varying numbers of layers, hidden units), a large network may well have better reliability when time is measured in number of pattern presentations due to higher redundancy. However, an actual implementation of it will take longer to process an input pattern than a smaller network performing the same task, and so the number of faults occurring may well be greater in the long term. This discrepancy should be compensated for in any comparative studies made. Producing results that can be compared when using different types of neural network models or non-neural systems requires similar consideration.

Two possible guidelines for choosing a timescale for a neural network system are either examining the architecture and grouping together all of the parallel operations that are required during its processing stages, and then defining one unit of time to be the execution (in parallel) of any particular group. The other possibility is to examine the abstract description of the neural network model (e.g. see figure 4.3), and to define a unit of time to be a recognizable mathematical

operation. Both of these will allow comparisons between the same neural networks model but with varying internal structure, though to compare different (or non-deterministic) neural network models, some allowance must be be made for the complexity of operation for each possible time unit such that the various models are evenly balanced.

5.3. Fault Injection Methods

Fault injection techniques involve subjecting a system to a known number of faults, then measuring the subsequent degradation. This has to be repeated many times to achieve a statistically significant result. The measure used to assess the system must be related to the *degree* of failure of the system, since it is reliability which is of interest here. Note that a system may maintain perfect performance until a fault threshold is reached, when it suffers total failure. The discussion above on measuring the degree of failure of a neural network is applicable here.

The resulting plots from experiments of the measure of reliability against many and possibly various types of faults injected into a system, which can be termed *fault curves*, will indicate how an operational system will behave if the rate at which each type of fault occurs is known.

Fault injection techniques do suffer from a number of shortcomings. By far the most damaging is when a system can suffer more than a single type of fault, as will almost certainly be the case. Fault injection simulations are very good at indicating the isolated effects of a number of identical faults occurring in a system, but complications arise when many different fault types have to be taken into account since their effects will not be independent. This makes it very difficult to predict with any degree of accuracy the effects of various faults on a system which would occur in real-life use. Combining in some fashion the effects of particular individual faults occurring in isolation is very unlikely to be similar to the effects of all faults occurring together over a period of time; the effects of individual fault types cannot simply be added together due to correlations between them.

In conclusion, fault injection methods are only useful to gain a very basic indication of the reliability of a neural network system, though they may identify especially critical faults which can then be protected against in any implementation design.

5.3.1. *Example*

For the multi-layer perceptron network (see figure 4.3), the fault model defined previously in section 4 can be used for fault injection experiments. Since a continuous measure is required for fault injection techniques, the partial failure characteristic of neural networks due to their soft application areas can be exploited.

The definition of failure in the previous example can be used in that of a function f measuring reliability, and since this is a probability, its codomain must range over $[0, 1]$. It should also be a continuous monotonic mapping since as the degree of failure increases, the reliability should decrease. As before, two cases exist depending upon whether generalisation is required, though they only differ in the argument given to f.

If generalisation is not required, then for a single pattern p, the measure of reliability is

$$f_p\left(\max\left\{\min\left\{\forall i \neq p. \|\underline{t_i} - \underline{o_p}\|\right\} - \|\underline{t_p} - \underline{o_p}\|, 0\right\}\right)$$

$$\text{such that } f_p(0) = 0$$

$$\text{and } f_p\left(\min\left\{ \forall i \neq p. \| \underline{t}_j - \underline{o}_p \| \right\} \right) = 1$$

However, if generalisation is relied upon, then for a single pattern p, then the measure of reliability is

$$f_p\left(\max\left\{ HD - \| \underline{t}_p - \underline{o}_p \|, 0 \right\} \right)$$

$$\text{such that } f_p(0) = 0$$

$$\text{and } f_p(HD) = 1$$

To extend these two definitions to cover all patterns p, the maximum degree of failure should be chosen to gain an idea of the on-line performance,

$$f = \max\left\{ \forall p. f_p \right\}$$

and their average (possibly weighted) for off-line, i.e. if ρ_p is an indication of the importance of input-output space around pattern p

$$f = \sum_p \rho_p f_p$$

5.4. Mean-Time-Before-Failure Methods

An alternative method for judging the reliability of a system is to measure the average time period before failure *first* occurs. Just as for fault injection methods, the results obtained are statistical in nature, and so precise conclusions cannot be made. However, a major differences between the two methods is that failure is considered as a discrete event here, rather than as a continuous variable. The discussion above on the definition of a suitable timescale is clearly relevant here. Note that both the timescale chosen and the definition of discrete failure will be somewhat dependent upon the application and neural network architecture being considered, though some generalities may exist between sub-groups.

As mentioned above, failure of a neural network is difficult to define since generally, unlike most conventional computing systems, they do not spectacularly crash when faults occur; some degree of graceful degradation or fail-soft nature is apparent. Also, many of the possible applications for which they could be applied are equally flexible when it comes to defining failure, such as for the neural network which balances a pole mentioned above. However, the treatment of "failure" is different for MTBF methods from that used in fault injection methods. Here, failure is a discrete event, it either happens or does not happen, and so the continuous measures of failure used in fault injection investigations cannot be directly applied. Instead, some rules need to be defined which specify when failure has been deemed to have occurred. A general definition of failure is that *it occurs whenever the system does not meet its specification.* This places the burden of responsibility onto the specifier of a system, and the specification must define in detail the acceptable behaviour of the system. This will include the limits to which degradation can occur, and so creates the distinction between failure and non-failure. These limits can be defined using

the various general conditions that were discussed above for fault injection methods, though others which are specific to the neural network or application may be included by the designer as appropriate. For example, an output unit could be defined to have failed when its output deviates by at least 20%. A more global definition might be that failure occurs when a neural network incorrectly classifies more than 5% its inputs.

The basic MTBF technique can be extended when investigating neural networks to assess the time *between* sequential failures since they can have the property of automatic recovery from failures. This occurs since their functionality is unaffected by errors in information processing caused either by transient faults or due to uneven distribution of information. However, if feedback occurs in the neural network's topology, then this might disrupt recovery since errors could be amplified.

In conclusion however, the rather gross simplification of failure from the continuous degradation which actually occurs in a neural network to the discrete on-off event used here, detracts from the usefulness of MTBF models for assessing the reliability of a neural network system.

5.4.1. *Example*

To apply MTBF methods to the multi-layer perceptron (MLP) neural network, the following requirements need to be met. A reasonable fault model needs to be developed, a suitable timescale needs to be chosen, and also the notion of failure in the MLP. The fault model defined earlier in section 4 can be applied here. A suitable choice of timescale will depend to a large extent upon the application chosen, for a classification problem, the timescale could relate to the number of patterns presented. Failure can be treated similarly as in the above example, but replacing the function f by one which jumps from 0 to 1 when the distance threshold HD is reached if generalisation is relied upon, or else when the output pattern $\underline{o}_p$ is closer to $\underline{t}_q$ *where* $p \neq q$ if it is not.

By running many simulations, a plot of the cumulative number of simulation runs against MTBF against the number of times a simulation has already failed (i.e. a 3D graph) can be made. This will show the distribution of the MLP's failure rate, and also it will show how a system will behave after it has suffered N previous failures. However, it will not indicate the degree of graceful degradation exhibited due to the discrete failure event.

5.5. Service Degradation Methods

As mentioned above, both the fault injection and MTBF methods for measuring the reliability of a neural network have their shortcomings. However, a combination of the two methods can be devised which draws on their strengths, and removes their associated problems. The continuous measures used in fault injection experiments are combined with the timescales and fault rates of the extended MTBF methods to produce a means by which to assess the global reliability of a neural network system as time progresses. Since most neural networks exhibit graceful degradation, this method provides a clear indication of impending catastrophe in the system.

To achieve a continuous-valued indication of the global reliability of the system, it is possible to assign a probability to each particular fault mode which indicates both how likely it will manifest itself in a single unit of time, and also the fraction of locations in which it will occur. Faults can then be generated probabilistically during the simulation run. It is important to take into account

both of these factors since a fault which is unlikely to occur, but has numerous fault locations could well be more likely to occur than a highly probable fault that can only occur in a very few locations. By dynamically generating various types of faults during the simulation, any correlations between their effects will automatically be taken into account. The degree of failure in the system can then be probed by using the reliability measures discussed above in section 5.1. Similarly as for the MTBF reliability methods, another problem is that of choosing a valid and reasonable timescale for faults, and the discussion in section 5.2 applies equally well here to service degradation methods. However, although this method results in a clear picture of a neural network's graceful degradation of reliability, to collect statistically meaningful results using this method, many simulation runs will have to be performed, and the total computational cost could be very large. For safety-critical systems though, failure would be far more costly.

5.5.1. *Example*

By using the fault model developed earlier, the timescale as given in the example for MTBF methods, and also the continuous reliability measure defined in the example for fault injection techniques, the reliability of the MLP can be assessed. This is done by running many simulations (to collect statistically valid data), placing faults probabilistically according to the predefined fault rates, and measuring the reliability of the MLP at each time step. This produces a plot of the reliability of the MLP against time, and its performance can then be judged. Depending upon the generic nature of the fault model, timescale, and reliability measure used, the results obtained from various different experiments (e.g. different size MLP's) can be compared and contrasted.

6. Conclusions

A review of work examining the fault tolerance of neural networks has been presented. Also, a requirement was identified for a structured approach when investigating this area, and a suitable framework was proposed. This lead to a method for deriving a fault model for a neural network visualised at the abstract level to be constructed, and then it was incorporated into various measures for assessing the reliability of a neural network. Together, the fault model and reliability measures allow the fault tolerance of a neural network system to be investigated constructively. Also, various possible influences on the fault tolerance of neural networks were discussed.

It is clear that no wide-sweeping generalisations are possible for both fault models and reliability measures, though this is obviously a very desirable feature since it would allow easy and valid comparisons to be made between various neural and non-neural systems. However, for a particular application, it is possible to define both a fault model and a reliability measure (coupled with a suitable timescale) such that a single neural network can be investigated, or a comparative study can be made between various similarly based neural network systems.

Several major advantages of neural networks with respect to fault tolerance and reliability characteristics have been noted. Faults are not masked physically (e.g. as in N-Modular Redundancy architectures), but merely by the nature of neural networks' processing characteristics. Any fault will influence the output to some degree since all components participate in any computation. This leads to graceful degradation being exhibited by most neural networks, i.e. neural networks will not suffer catastrophic failure, and also allows approaching failure to be detected by using a continuous reliability measure. The fault tolerance which results in this reliability is inherent within neural networks, it does not need to be specifically designed and built into them, and so the architectural complexity which often arises due to various fault tolerance techniques being used is absent in neural network systems. Finally, although any faults which do occur cannot be located, they can be removed from the system since neural networks can learn.

References

1. Abu-Mostafa, Y.S., ‘‘Complexity of random problems’’, in *Complexity in Information Theory*, Springer-Verlag (1986).
2. Abu-Mostafa, Y.S., ‘‘Neural Networks for Computing?’’, *AIP Conference Proceedings 151*, pp. 1-7, American Institute of Physics (1986).
3. Albus, J.S., ‘‘A new approach to manipulator control: the Cerebellar Model Articulation Controller (CMAC)’’, *Trans. ASME-J. Dynamic Syst., Meas., Contr.* **97**, pp. 220-7 (1975).
4. Amit, D.J. and Gutfreund, H., ‘‘Statistical Mechanics of Neural Networks near Saturation’’, *Annals of Physics* **173**, pp. 30-67 (1987).
5. Ammann, P.E. and Knight, J.C., ‘‘Data Diversity: An Approach to Software Fault Tolerance’’, *IEEE Transactions on Computers* **37**(4), pp. 418-425 (April 1988).
6. Anderson, J.A., ‘‘Cognitive and Psychological Computation with Neural Models’’, *IEEE Trans Systems Man and Cybernetics* **SMC-13**, pp. 799-815 (1983).
7. Anderson, J. A., ‘‘Neural models with cognitive implications.’’, pp. 27-90 in *Basic processes in reading perception and comprehension*, ed. , D. LaBerge and S. J. Sanuels, Erlbaum, Hillsdale, NJ (1977).
8. Baum, E.B. and Haussler, D., ‘‘What Size Net gives Valid Generalisation?’’, *NIPS-89, Denver*, Morgan Kaufmann (1987).
9. Baum, E.B., Moody, J. and Wilczek, F., ‘‘Internal Representations for Associative Memory’’, *Biological Cybernetics* **59**, pp. 217-228 (1988).
10. Bedworth, M.D. and Lowe, D., *Fault Tolerance in Multi-Layer Perceptrons: a preliminary study*, RSRE: Pattern Processing and Machine Intelligence Division (July 1988).
11. Belfore, L.A. and Johnson, B.W., ‘‘The fault-tolerance of neural networks’’, *The International Journal of Neural Networks Research and Applications* **1**, pp. 24-41 (Jan 1989).
12. Brause, R., ‘‘Fault Tolerance in Intelligent User Interfaces’’, *Inf. Tech. IT (West Germany)* **30**, pp. 219-24 (1988).
13. Brause, R., ‘‘Fault Tolerance in Neural Network Associative Memory’’, *Proceedings of HICCS-24* (1990).
14. Bruce, A., Canning, A., Forrest, B., Gardner, E. and Wallace, D.J., ‘‘Learning and Memory Properties in Fully Connected Networks’’, *AIP Conference Proceedings 151*, pp. 65-70, American Institute of Physics (1986).
15. Cannon, S.C., Robinson, D.A. and Shamma, S., ‘‘A Proposed Neural Network for the Integrator of the Oculomotor System’’, *Biological Cybernetics* **49**, pp. 127-36 (1983).
16. Carter, M.J., ‘‘The ’Illusion’ of Fault Tolerance in Neural Networks for Pattern Recognition and Signal Processing’’, *Proc. Technical Session on Fault-Tolerant Integrated Systems, Durham*, Durham NH: University of New Hampshire (1988).
17. Carter, M.J., Rudolph, Frank and Nucci, Adam, ‘‘Operational Fault Tolerance of CMAC Networks’’, *NIPS-90, Denver*, Morgan Kaufmann (1990).
18. Damarla, T.R. and Bhagat, P.K., ‘‘Fault Tolerance in Neural Networks’’, *Southeastcon ’89 Proceedings: Energy and Information Technologies in the S.E.* **1**, pp. 328-31 (1989).

19. Fogelman-Soulie F., Gallinari P., Cun, Le Y. and Thiria S., ‘‘Evaluation of network architectures on test learning tasks’’, *Proceedings of the first IEEE International Conference on Neural Networks, San-Diego* **II**, pp. 653-660 (1987).

20. Hancock, P., *Data representation in neural nets: an empirical study*, Morgan Kaufmann, Carnegie-Mellon University (1988).

21. Hartley, R. and Szu, H., ‘‘A Comparison of the Computational Power of Neural Network Models’’, *Proceedings of the first IEEE International Conference on Neural Networks, San-Diego* **3**, pp. 15-22 (1987).

22. Hayes, John P., ‘‘Fault Modeling’’, *IEEE Design & Test*, University of Michigan (April 1985).

23. Hopfield J.J., ‘‘Neural networks and physical systems with emergent collective computational abilities’’, *Proceedings of the National Acadamy of Sciences, USA* **79**, pp. 2554-8 (1982).

24. Kauffman S.A., ‘‘Metabolic stability and epigenesis in randomly connected genetic nets’’, *Journal of Theoretical Biology* **22**, pp. 437-467 (1969).

25. Kohonen T., ‘‘Self-organized formation of topologically correct feature maps’’, *Biological Cybernetics* **43**, pp. 59-69 (1982).

26. Krauth, W., Mezard, M. and Nadal, J.P., ‘‘Basins of Attraction in a Perceptron-Like Neural Network’’, *Complex Systems* **2**, pp. 387-408 (1988).

27. Kung, S.Y., ‘‘Parallel Architectures for Artificial Neural Nets’’, *Proc. International Conference on Systolic Arrays*, pp. 163-74 (1988).

28. Lansner, A. and Ekeberg, O., ‘‘Reliability and Speed of Recall in an Associative Network’’, *IEEE Trans Pattern Analysis and Machine Intelligence* **PAMI-7** (1985).

29. Legendy, C.R., ‘‘On the Scheme by Which the Human Brain Stores Information’’, *Mathematical Biosciences* **1**, pp. 555-97 (1967).

30. Lehky S.R. and Sejnowski T.J., ‘‘Network model of shape-from shading:neural function arises from both receptive and projective fields’’, *Nature* **333**, pp. 452-454.

31. Maestri, G H, ‘‘The retryable processor’’, *AFIPS, Fall Joint Computer Conference* **41**(1), pp. 273 - 277 (1972).

32. McCullogh W.S. and Pitts A., ‘‘A logical calculus of the ideas immanent in nervous activity’’, *Bulletin of Mathematical Biophysics* **5**, pp. 115-133 (1943).

33. McEliece, R.J., Posner, E., Rodemich, E. and Venkatesh, S., ‘‘The Capacity of the Hopfield Associative Memory’’, *IEEE Trans. Info. Theory* **IT-33**, pp. 461-82 (1987).

34. Miikkulainen, Risto and Dyer, Michael G., ‘‘Natural Language Processing with Modular Neural Networks and Distributed Lexicon.’’, *Cognitive Science* (1989).

35. Miikkulainen, Risto and Dyer, Michael G., ‘‘Encoding Input/Output Representations in Connectionist Cognitive Systems’’, *Proceedings of the 1988 Connectionist Models Summer School* (1989).

36. Minsky M. and Papert S., *Perceptrons: An introduction to computational geometry*, MIT Press, Cambridge, MA (1969).

37. Moore, W.R., ‘‘Conventional Fault-Tolerance and Neural Computers’’, pp. 29-37 in *Neural Computers*, ed. Eckmiller, R and von der Malsburg, C, Berlin: Springer-Verlag (1988).

38. Palumbo, Daniel L., ‘‘Assessing the Fault Tolerance of Neural Networks’’, *WNN-AIND-90*,

p. 3, NASA Langley Research Centre (Feb 1990).

39. Pemberton, J.C. and Vidal, J.J., ‘‘The effect of training signal errors on node learning’’, *Technical Report: CSD-890041*, University of California (June 1989).
40. Protzel, P.W. and Arras, M.K., ‘‘Fault-Tolerance of Optimization Networks: Treating Faults as Additional Constraints’’, *IJCNN-90, Washington DC* (Jan 1990).
41. Rumelhart, D.E., Hinton, G.E. and Williams, R.J., ‘‘Learning Internal Representations by Error Propagation’’, in *Parallel Distributed Processing*, MIT Press, Cambridge, MA (1988).
42. Scalettar, R. and Zee, A., ‘‘A feed-forward memory with decay’’, *Institute for Theoretical Physics preprint: NSF-ITP-86-118* (1986).
43. Sivilotti, M.A., Emerling, M.R. and Mead, C.A., ‘‘VLSI Architectures for Implementation of Neural Networks’’, *AIP Conference Proceedings 151*, pp. 408-413, American Institute of Physics (1986).
44. Specht, Donald F., ‘‘Probabilistic Neural Networks’’, *Neural Networks* **3**, pp. 109-118 (1990).
45. Tai, Heng-Ming, ‘‘Fault Tolerance in Neural Networks’’, *WNN-AIND-90*, p. 59, NASA Langley Research Centre (Feb 1990).
46. Takeda, M. and Goodman, J.W., ‘‘Neural Networks for computation: number representations and programming complexity’’, *Applied Optics* **25** (1986).
47. Tanaka, H., ‘‘A Study of a High Reliable System against Electric Noises and Element Failures’’, *Proceedings of the 1989 International Symposium on Noise and Clutter Rejection in Radars and Imaging Sensors*, pp. 415-20 (1989).
48. Tesauro, G. and Sejnowski, T.J., ‘‘A Parallel Network that Learns to Play Backgammon’’, *Technical Report CCSR-88-2*, Center for Complex Systems Research, University of Illinois, Urbana-Champaign, IL (1988).
49. Venkatesh, S.S., ‘‘Epsilon Capacity of Neural Networks’’, *AIP Conference Proceedings 151*, pp. 440-445, American Institute of Physics (1986).
50. von Seelen, W. and Mallot, H.A., ‘‘Parallelism and Redundancy in Neural Networks’’, pp. 50-60 in *Neural Computers*, ed. Eckmiller, R and von der Malsburg, C, Berlin: Springer-Verlag (1988).
51. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. and Lang, K., ‘‘Phoneme recognition using time-delay neural networks’’, *Tech. Rep. TR-I-0006*, Japan: Advanced Telecommunications Research Institute (1987).
52. Wechsler, H. and Zimmerman, G.L., ‘‘Fault-Tolerant Recognition using DAM’s’’, pp. 719-26 in *Proceeding of the IEEE First Annual International Conference on Neural Networks* (June 1987).
53. Widrow B., Gupta N.K. and Maitra S., ‘‘Punish/Reward: Learning with a critic in adaptive threshold systems’’, *IEEE Trans. Systems, Man, and Cybernetics* **SMC-3**, pp. 455-465 (1973).
54. Wood, C., ‘‘Implications of simulated lesion experiments for the interpretation of lesions in real nervous systems’’, in *Neural Models of Language Processes*, ed. Arbib, M.A., Caplan, D. and Marshall, J.C., New York: Academic (1983).
55. Worden, S.J. and Womack, B.F., ‘‘Analysis of small compacta networks’’, *Proceedings of*

1986 IEEE Conference on Systems, Man and Cybernetics, pp. 61-4 (1986).

56. Zhou, Y.T., Chellappa, R. and Jenkins, B.K., ‘‘A Novel Approach to Image Restoration Based on a Neural Network’’, pp. 269-76 in *Proceeding of the IEEE First Annual International Conference on Neural Networks* (June 1987).

57. Zymslowski, W., ‘‘Some problems of sensitivity of neuronal nets to variations of parameters of their elements’’, *IFAC Symposium on Automatic Control and Computers in the Medical Field*, pp. 133-7 (1971).