NAG1-613

August 1990

UILU-ENG-90-2229
CRHC-90-1

Center for Reliable and High-Performance Computing

P-108

EXPERIMENTAL FAULT CHARACTERIZATION OF A NEURAL NETWORK

Chang-Huong Tan

(NASA-CR-186895) EXPERIMENTAL FAULT CHARACTERIZATION OF A NEURAL NETWORK (Illinois Univ.) 106 p CSCL 09B

N90-28373

Unclas
G3/62 0297568

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188
1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS None		
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-90-2229 CRHC-90-1		5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NASA		
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Ave. Urbana, IL 61801		7b. ADDRESS (City, State, and ZIP Code) Langley Research Center Hampton, VA 23665		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NASA	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NAG-1-613		
8c. ADDRESS (City, State, and ZIP Code) Langley Research Center Hampton, VA 23665		10. SOURCE OF FUNDING NUMBERS		
		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) "Experimental Fault Characterization of a Neural Network"				
12. PERSONAL AUTHOR(S) Chang-Huong Tan				
13a. TYPE OF REPORT Technical	13b. TIME COVERED FROM ____ TO ____	14. DATE OF REPORT (Year, Month, Day) 1990 August 3	15. PAGE COUNT 93	
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES: FIELD / GROUP / SUB-GROUP		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) neural network, experimental analysis, simulation, fault injection, fault models, evaluation		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this thesis, the effects of a variety of faults on a neural network is quantified via simulation. The neural network consists of a single-layered clustering network and a three-layered classification network. The percentage of vectors mistagged by the clustering network, the percentage of vectors misclassified by the classification network, the time taken for the network to stabilize, and the output values are all measured. The results show that both transient and permanent faults have a significant impact on the performance of the measured network. The corresponding mistag and misclassification percentages are typically within 5% to 10% of each other. The average mistag percentage and the average misclassification percentage are both about 25%. After relearning, the percentage of misclassifications is reduced to 9%. In addition, transient faults are found to cause the network to be increasingly unstable as the duration of a transient is increased. The impact of link faults is relatively insignificant in comparison with node faults (1% versus 19% misclassified after relearning). There is a linear increase in the mistag and misclassification percentages with decreasing hardware redundancy. In addition, the mistag and misclassification percentages linearly decrease with increasing network size.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

GRADUATE COLLEGE DEPARTMENTAL FORMAT APPROVAL

THIS IS TO CERTIFY THAT THE FORMAT AND QUALITY OF PRESENTATION OF THE THESIS SUBMITTED BY CHANG-HUONG TAN AS ONE OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE ARE ACCEPTABLE TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING.

Full Name of Department, Division or Unit

27 July 1990

Date of Approval

[illegible]

Departmental Representative

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

JULY 1990

WE HEREBY RECOMMEND THAT THE THESIS BY

CHANG-HUONG TAN

ENTITLED EXPERIMENTAL FAULT CHARACTERIZATION OF A NEURAL NETWORK

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

[illegible]
Director of Thesis Research

[illegible]
Head of Department

Committee on Final Examination†

Chairperson

† Required for doctor's degree but not for master's.

O-517

EXPERIMENTAL FAULT CHARACTERIZATION OF A NEURAL NETWORK

BY

CHANG-HUONG TAN

B.S., University of Illinois, 1989

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1991

Urbana, Illinois

ABSTRACT

In this thesis, the effects of a variety of faults on a neural network is quantified via simulation. The neural network consists of a single-layered clustering network and a three-layered classification network. The percentage of vectors mistagged by the clustering network, the percentage of vectors misclassified by the classification network, the time taken for the network to stabilize, and the output values are all measured. The results show that both transient and permanent faults have a significant impact on the performance of the measured network. The corresponding mistag and misclassification percentages are typically within 5% to 10% of each other. The average mistag percentage and the average misclassification percentage are both about 25%. After relearning, the percentage of misclassifications is reduced to 9%. In addition, transient faults are found to cause the network to be increasingly unstable as the duration of a transient is increased. The impact of link faults is relatively insignificant in comparison with node faults (1% versus 19% misclassified after relearning). There is a linear increase in the mistag and misclassification percentages with decreasing hardware redundancy. In addition, the mistag and misclassification percentages linearly decrease with increasing network size.

DEDICATION

To my parents.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Ravi Iyer, who spent many of his precious hours reviewing this thesis and providing excellent advice. Thanks are due to Robert Dimpsey, Dennis Sng, Sylvia Chong, Rene Llames, Hing-Yan Lee and Bruce Ferguson for their comments on an early draft of this thesis. Thanks are also due to Aloke Gupta for his assistance in preparing this thesis. Special thanks go to my friends and colleagues here who have provided their friendship, help and support. Finally, I would like to express my heart-felt appreciation and gratitude to my parents and grandparents who have made everything possible for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1.

INTRODUCTION

Artificial Neural Networks (ANNs) are based on simplified biological neural models. Each node in an ANN mimics the function of a biological neuron. ANNs have been extensively studied and have been used reasonably successfully in areas such as speech and pattern recognition. Research in neural networks began with the work of McCulloch, Pitts, Hebb, Rosenblatt, Widrow and others some 40 years ago [1]. Today, interest has re-emerged with the development of new network topologies, learning algorithms and analog VLSI techniques [2, 3].

A typical neural network consists of massively interconnected processing elements known as artificial neurons. Weights are associated with each interconnection in the network. The simplest element, sums N weighted inputs and passes the sum through a non-linear function such as a sigmoid. The processing elements may also contain small local memories which store the weights associated with the interconnections. The learning process corresponds to searching for a set of optimal link-weights which minimizes the error in the output function.

The claim is often made that neural networks are highly robust and fault tolerant. This property is usually attributed to the fact that these networks have massive interconnections and also to the manner in which learning occurs in these networks. These two factors are jointly expected to provide considerable redundancy in both information and hardware. The networks are thus expected to be highly adaptive to failure situations. This belief, however, has not been systematically investigated. For example, although the loss of data may not seem significant [4], total or partial major component failure may have an adverse impact on the system. This is particularly true if the network is expected to perform in real time.

The two neural network models studied in this thesis are: Kohonen's Feature Map [5] and the multilayered perceptron network [6]. Both networks have been used in a variety of applications. Kohonen's Feature Map has been used for speech and pattern recognition [7,8], and a neural network chip based on it has been fabricated [9]. The multilayered perceptron model is one of the most extensively studied neural network in the literature. It has been used to implement medical expert systems [10,11], pattern recognition systems [12,13], and an adaptive closed-loop controller [14]. A VLSI implementation of the multilayered perceptron model is also currently being developed [14]. Thus, with these networks going to VLSI chip implementations, the fault characteristics of each must be investigated.

This thesis reports on the results of a set of experiments conducted to quantify the effect of a variety of faults on a neural network consisting of a single-layered clustering network and a three-layered classification network. The network was used as a target for analyzing the impact of faults on system behavior, via simulation. The percentage of vectors mistagged by the clustering network, the percentage of misclassification by the classification network, the time taken for the network to stabilize, and the output values were all measured after 600 learning cycles (the average learning duration). The network was allowed to relearn for an additional 600 cycles, and the benefits of relearning were quantified. It was considered inefficient to allow the relearning process to continue beyond 600 cycles since it would then be better to initiate a fresh start.

The results show that both transient and permanent faults have a significant impact on the performance of the network. The corresponding mistag and misclassification percentages are typically within 5% to 10% of each other. This is attributed to the short relearning duration in the experiment. In addition, transient faults are found to cause the network to be increasingly unstable as the duration of the transient is increased.

The average mistag percentage and the average misclassification percentage are both about 25%. After relearning, the percentage of misclassifications is reduced to 9%. The impact of the link faults is relatively insignificant in comparison with node faults (1% versus 19% misclassified after relearning). This is because the massive interconnections provide considerable information redundancy.

A study of hardware redundancy shows a linear increase in the mistag or misclassification percentages with decreasing hardware redundancy. Thus, the penalty per unit decrease in the hardware redundancy is constant. A study of hardware size shows that the mistag and misclassification percentages linearly decrease as the minimal hardware size is increased. Thus, large networks result in lower mistag and misclassification percentages than small networks. Node faults are found to cause about three times the percentage of mistags or misclassifications as compared to link faults.

The next chapter discusses related research and the motivation for this thesis. In Chapter 3, a neural network implementation of the diagnostic system is discussed. Chapter 4 defines the fault models used to inject specific fault types into the simulated network. Chapter 5 defines the appropriate performance measures for the network. The overall experimental approach and the measurements taken are discussed in Chapter 6. Chapters 7, 8 and 9 discuss the experimental results. Chapter 10 is the concluding chapter which highlights the major results of this work and makes suggestions for future research.

CHAPTER 2.

RELATED RESEARCH

The issue of reliability in neural networks has not been fully investigated. One area that has been addressed is the impact of noisy environments on the recall capabilities of neural associative memories. In [5], a neural network implemented as a content associative memory is shown to be able to retrieve the complete image, given fragments of the original. In [15], it is shown that neural network pattern recognition is tolerant of faults in the input information. Another area that has been addressed is the impact of low output values in a multilayered perceptron model. Low output values arise because input patterns cannot be confidently assigned to a unique class. In [10], the problem of low output values is dealt with by lowering the threshold.

In [16], the stuck-at-1 and stuck-at-0 link-fault models were used to study the fault tolerance of a neural associative memory. The authors measured the recall capabilities of the network with an increasing number of link failures. It was shown that a 16-node network was able to function effectively with 20% link failures. In [17], the fault tolerance of a character recognition system, implemented as a multilayered perceptron network, was investigated. The impact of data faults was studied by measuring the percentage of characters erroneously recognized as a function of the number of training cycles for three different Hamming distances.[1] The results indicated that as the Hamming distance was increased, the steady-state error percentage also increased. It was also shown that the parameters of the learning algorithm did not affect the classification error. The hardware fault tolerance was studied by measuring the percentage of erroneous recognitions as a function of the number of broken links. For example, a network trained for 500 cycles could tolerate up to 20% link failures. The results also showed that the minimum number of links required for the network to function effectively decreased as the network was trained for a longer period. With 1600 cycles, the same network could tolerate about 60% link failures.

In the above research, the impact of faults was quantified by the recall capability and the classification accuracy of the measured network. The authors investigated the impact of catastrophic failures, i.e., multiple link failures. In addition, the fault types were limited to link and input faults. Traditional fault tolerant design, however, assumes single point failures in a system. To date, there is no systematic study which investigates the impact of single faults in a neural network. It is also unclear how the network would behave when node faults also occur. This is important since link redundancy can easily be an order of magnitude higher than node redundancy. Further, issues such as the impact of faults on the

[1]The authors measured data fault tolerance by the maximum number of different bits (Hamming distance) between the input and the training patterns.

learning duration, the effect of hardware redundancy and the effect of hardware size must be studied to address the issue of fault tolerance of neural networks.

In this thesis, the effects of a variety of single fault events, both transient and permanent, on two different network models are studied. The impact of faults is quantified by the learning duration, the accuracy of classification and the output values of the network. The effects of hardware redundancy and information size are also investigated.

CHAPTER 3.

TARGET NEURAL NETWORK SYSTEM

For this thesis, a neural network implementation of an on-line failure diagnostic system was developed. The implementation was based on Pau's automated failure diagnostic model [18]. The model consists of a learning phase and a recognition phase. In the learning phase, an N-attribute vector which describes the system behavior is defined. For example, in aircraft failure diagnosis, some of the attributes would include stress and strain factors, number of flying hours, load factors, service information and overhaul information. A total of P failure observations are made and the set of P N-dimensional vectors constitutes the training vectors for the system. Statistical clustering is then used to reduce the training set to an optimal number of classes. These classes (denoted by the cluster centroids) represent the principal identifiable failure modes of the system.

When a failure occurs, the recognition phase is invoked. The new failure vector is provided as input to the diagnostic system and a nearest neighbor rule is used to determine the most probable failure mode. The new vector is assigned to the cluster that yields the minimum Euclidean distance between the vector and the cluster centroid.

A neural network implementation of the above diagnostic system consists of two subnetworks: a clustering network and a classification network. In the learning phase, two procedures are performed: the clustering of the set of P training vectors and the training of the classification network. The clustering network reduces the set of P vectors to an optimal number of clusters. Each of the clusters represents a most probable failure mode. A tag which identifies the appropriate cluster is added to each training vector. For example, if an input vector with four attributes, such as (0.23, 0.33, 0.54, 0.83), belongs to cluster 1, then the corresponding tagged vector is (0.23, 0.33, 0.54, 0.83, 1). The set of P tagged vectors is used to train the classification network.

The classification network is trained to classify the set of P failure vectors into the desired clusters as stipulated by the added tags. The learning objective is to obtain a set of optimal link-weights so that the classification network can classify each of the P vectors to its nearest cluster. When a failure occurs, the trained classification network assigns the failure vector to the most probable failure mode, i.e., to the cluster which has the closest Euclidean distance to the failure vector. The next two sections provide an overview of the detailed functions of the two subnetworks.

3.1. Clustering Network

The clustering network used in this thesis is based on Kohonen's Feature Map [5]. Figure 1 shows a typical structure of the network. There are N input nodes which represent the N attributes in the failure vector and an array of K output nodes (in the figure, K=16). Each input node is linked to all K output nodes. The link-weights for each output node correspond to the coordinates of the cluster centroid. We start the clustering process by

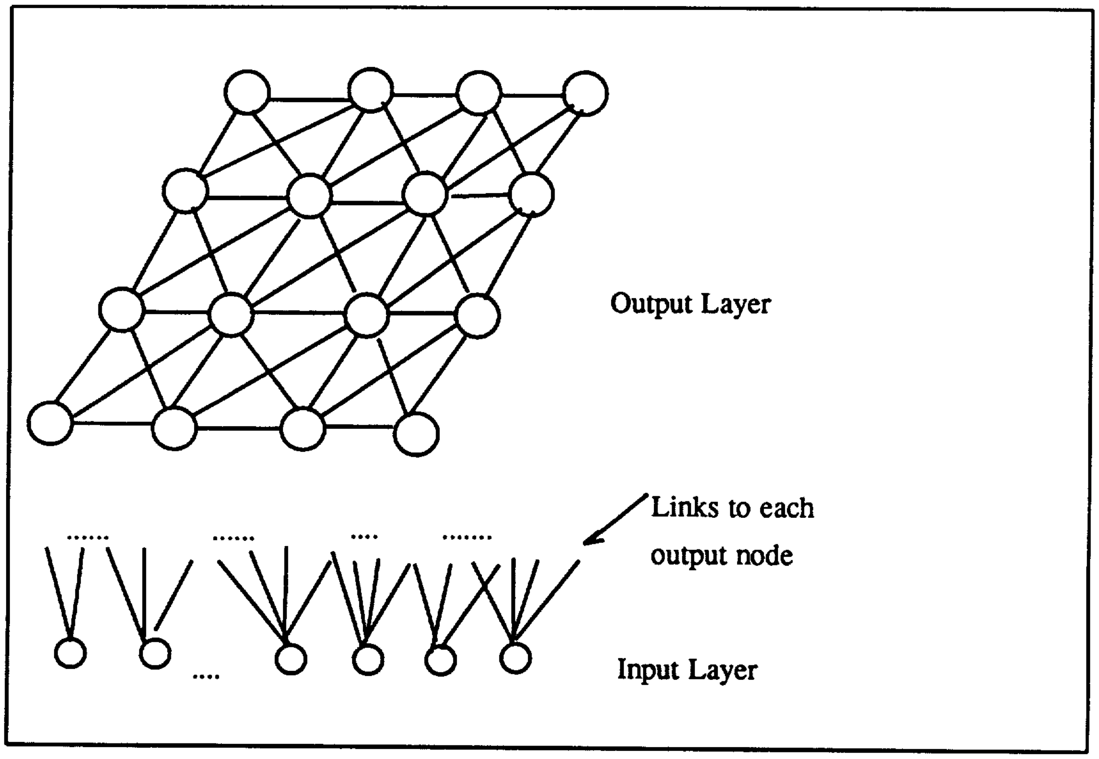

Figure 1. Kohonen's Feature Map with sixteen output nodes.

assigning random link-weights. These weights are then iteratively modified according to the Kohonen Feature Map Algorithm [5], which is a neural network implementation of the k-means clustering algorithm [2]. Once the link-weights are stable, only one output node is active for a given input vector. This output node represents the cluster which is nearest to the input vector, i.e., the Euclidean distance between the input vector and the specific cluster centroid is the smallest.

A minimal clustering network is defined as one where the number of output nodes matches the number of optimal clusters in the vector set. This type of network will be extremely useful in our experiments. A sketch of the algorithm and the clustering process appear in Appendix A.

3.2. Classification Network

The classification network uses the three-layer perceptron model [6]. Figure 2 shows a typical structure of the network. It consists of an input layer, two hidden layers and an

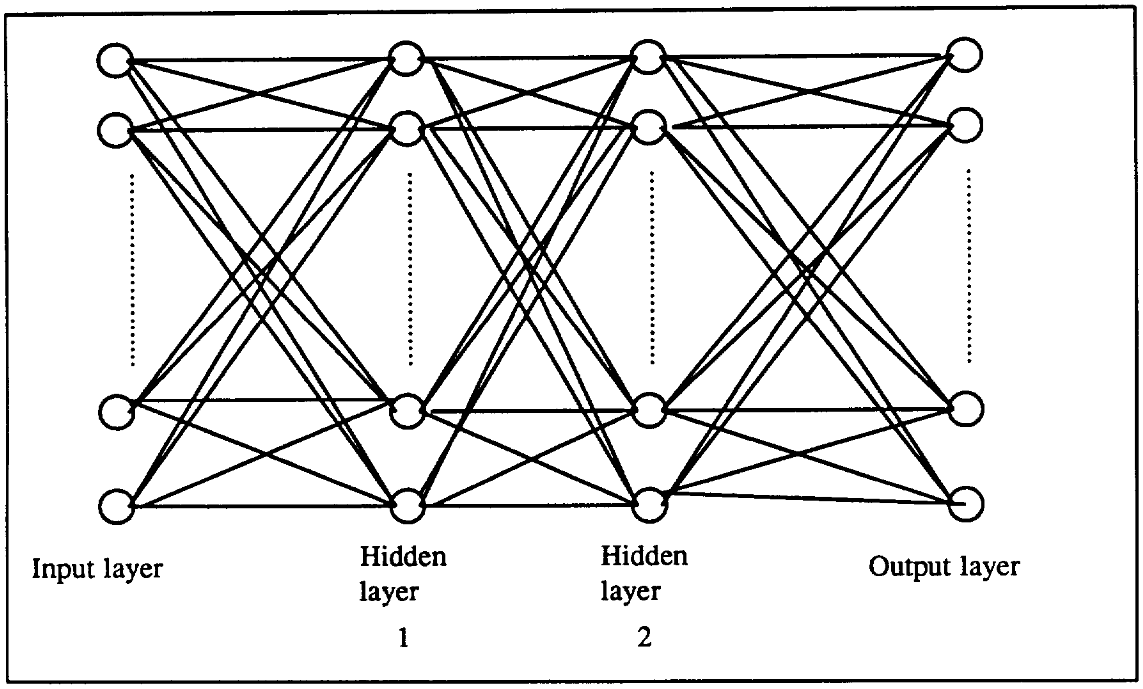

Figure 2. A three-layered perceptron model.

output layer.[2] There are N nodes in the input layer, which in our case, denote the N attributes in a failure observation. There are K output nodes, each represents a cluster centroid, which in turn denotes a principal identifiable failure mode. The number of nodes in the hidden layers depends on the number of clusters in the input vector space[3] [2]. Each node, in each of the layers, is linked to every node in the next layer. Thus, the nodes in the first hidden layer are linked to the nodes in the second hidden layer, and the nodes in the second hidden layer are linked to the output nodes. A weight is associated with each of the links.

The classification network takes an N-attribute failure vector and assigns it to a nearest cluster. In order to perform the classification, the network is trained such that, upon presentation of an input vector, only the output unit that represents the cluster nearest to the input vector is activated. For example, in a system with four attributes and six clusters, if an input vector (0.23, 0.44, 0.78, 0.65) belonging to cluster 3 is presented to the classification network, then ideally, the output pattern should be (0, 0, 1, 0, 0, 0).

The learning process determines a set of optimal link-weights to perform the classification. The tagged vectors from the clustering network are used as the training set. Each vector in the training set is presented to the classification network and the corresponding output pattern is determined. The tag of each input vector is then used to determine the corresponding desired output patterns. For example, if the tagged input vector is (0.23, 0.44, 0.78, 0.65, 3), then the desired output pattern is (0, 0, 1, 0, 0, 0). The learning process

[2]By convention, the three layers refer to the two hidden layers and the output layer.

[3]A three-layered perceptron model can form any arbitrary hyperspace which can partition the input vector space into distinct classes. The number of nodes in the output layer equals the number of clusters in the input space. The worst-case number of nodes in the second hidden layer is equal to the number of disconnected clusters in the input space. There should be three times as many nodes in the second hidden layer as in the first layer [2].

minimizes the difference between the actual and the desired output pattern. The back-propagation learning algorithm with the generalized delta learning rule [6] is used to train the network. This algorithm minimizes the difference between the current output pattern and the desired output pattern by iteratively modifying the link-weights. The minimization usually leads to some optimal output values close to one or zero. For instance, the above example might produce an output pattern such as (0.01, 0.07, 0.96, 0.02, 0.11, 0.07). Appendix B provides a more detailed account of the learning algorithm.

As stated earlier, the number of nodes in the classification network depends on the number of optimal clusters in the training set. As with the minimal clustering network, we define a minimal classification network as one which has the minimum number of nodes and yet, yields no misclassifications. For each training set used in this study, a corresponding minimal network was determined. In order to initially determine the size of the minimal network, Lippmann's method [2] was used. The network was then trained with the specific training set. If the network was minimal, the removal of any node in the network would yield misclassifications. Thus, by decreasing the number of nodes until misclassification occurs, the minimal network size required for the specific training set was determined.

CHAPTER 4.

FAULT MODELS

This chapter discusses the types of faults injected into the diagnostic system. A network implementation, in which each node is a simple nonlinear element (with a sigmoid function) was assumed. Further, every node was assumed to have a small local memory (512 k-bytes) which stored the incoming link-weights. A special purpose simulator, FANS, which incorporates automatic fault injection and analysis was developed for the purpose of investigating these networks. The details of FANS, the acronym for a Fault Analysis Tool for Neural Nets, are discussed in Appendix C.

Both transient and permanent faults were injected into the interconnection links and nodes. Transient faults were modeled by limiting the duration of the fault. Permanent faults were modeled by removing the links or nodes throughout the duration of the simulation. In addition, Byzantine faults were also injected into the processing nodes. A Byzantine node fault was modeled by forcing a node to send erroneous information to some nodes, but correct information to others. In keeping with common practice, only one fault was assumed to occur at any one time. Figure 3 summarizes the different fault types which were injected.

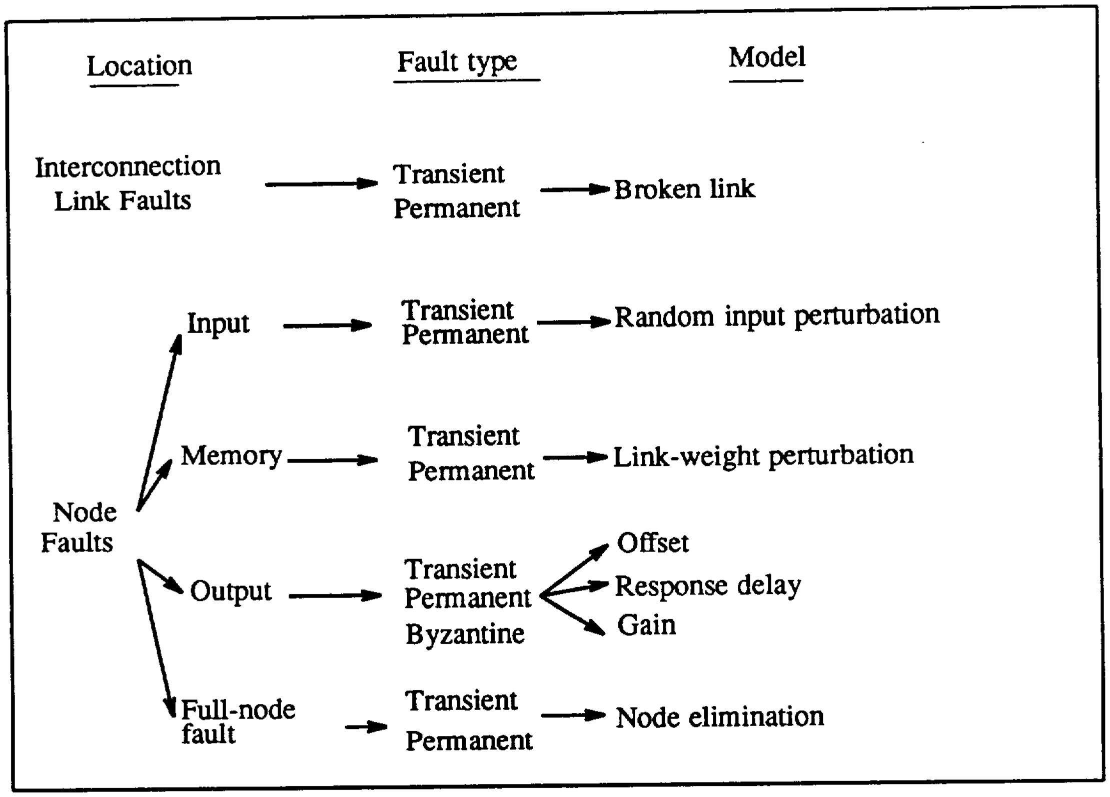

Figure 3. Categories of faults used in this experimental study.

4.1. Interconnection Link Faults

The interconnection links pass activation values from one node to another. The impact of faults on a specific link is such that no activation values can pass through. This fault is modeled by removing the targeted link. The fault can be transient or permanent.

4.2. Node Faults

Node faults are further categorized into input faults, memory faults, output faults and full-node faults. Node faults can be transient or permanent. In addition, a node output fault can also be Byzantine. A Byzantine fault at the node output models all cases of a Byzantine node because, by definition, a Byzantine node is one which is unreliable and sends erroneous outputs to only certain nodes [19].

4.2.1. Input faults

Input faults (faults occurring at the node inputs) are usually caused by external noise or glitches. This situation is modeled by a small random perturbation at the selected node input. Input faults can occur at the input of any node in the network and are not restricted to the nodes in the input layer. Both transient and permanent input faults were modeled.

4.2.2. Memory faults

A memory fault causes the weight stored in the memory to become unreliable. The fault is modeled by assigning a random value to a targeted link-weight in the network. Both transient and permanent faults were modeled.

4.2.3. Output faults

A node is defined as a nonlinear element with a sigmoid output function. Thus, the output faults in this thesis are typical perturbations in nonlinear circuit elements. Three types of node output fault models are used: the offset error, the response delay and the gain error. The offset error is modeled by the addition of a user-defined positive or negative constant to the output value. The node output response delay is modeled by adding a time delay Δt to the output function. Thus, an output $\Phi(t)$ is replaced by $\Phi(t-\Delta t)$. The gain error is modeled by multiplying the output value by a scaling constant. Output faults can be transient, permanent, or Byzantine.

4.2.4. Full-node faults

A full-node fault refers to a targeted node failing completely (i.e., no response occurs). It is simply modeled by removing the selected node from the network. This fault can be either transient or permanent.

CHAPTER 5.

MEASUREMENTS

Based on the fault models described in the last chapter, the following measurements were made to evaluate the impact of faults in the target network: the time τ taken to achieve stability, the value ϕ of the output node with the highest activation value, the difference δ between the two highest output nodes, the percentage γ of vectors mistagged by the clustering network, and the percentage β of the input vectors misclassified by the classification network. The following sections discuss these measurements in detail.

5.1. Time Taken to Achieve Stability

The amount of time, in simulation cycles, required for the link-weights to converge is denoted by τ. In our experiment, one simulation cycle, on both the clustering network and the classification network, consists of a forward and backward propagation.

5.2. Highest Output Value

The value of the highest output node in the classification network is denoted by ϕ. Recall that, ideally, for every input vector, the classification network will have a single output node at value "one"; the rest will be equal to zero. The error minimization process, however, often results in output values being close to one or zero. A value close to one indicates that the input vector can be assigned unambiguously to a specific cluster. Thus, ϕ can be interpreted as the degree of confidence with which an input vector can be assigned to a specific cluster.

5.3. Difference Between the Two Highest Output Values

The difference between the two highest output nodes is denoted by δ. If two nodes have close output values, the corresponding input vector cannot be confidently assigned to a single cluster. For example, if the values at the output layer of a classification network with four output nodes are:

$$(\text{node\_1}, \text{node\_2}, \text{node\_3}, \text{node\_4}) = (0.9, 0.2, 0.1, 0.05),$$

then, δ=0.70. If the values at the output nodes happen to be

$$(\text{node\_1}, \text{node\_2}, \text{node\_3}, \text{node\_4}) = (0.5, 0.6, 0.1, 0.05),$$

then δ=0.10. In the latter case, it is not clear whether the vector belongs to cluster 1 or cluster 2.

5.4. Percentage of Mistagged Vectors

The impact of faults in the clustering network is quantified by the percentage γ of mistagged vectors. A vector is said to be mistagged if the assigned tag does not represent the cluster which is nearest to the vector (as measured by the distance between the cluster centroid and the vector).

5.5. Percentage of Misclassifications

The impact of faults in the classification network is quantified by the percentage β of misclassifications. The classification network is said to misclassify a vector if either of the following two conditions holds: an input vector is assigned to an incorrect cluster or the difference δ between the two highest output values is less than a user-specified threshold θ. An input vector is assigned to an incorrect cluster if the distance between the specific cluster centroid and the input vector is not the minimum. The second condition requires that the input vector be confidently assignable to a unique cluster.

CHAPTER 6.

EXPERIMENTAL APPROACH

Three major categories of experiments were conducted on the target neural network. In Experiment 1, the impact of faults on the clustering network was investigated. Experiment 2 investigated the impact of faults on the classification network. Experiment 3 investigated the impact of faults, due to propagation, from the clustering network to the classification network. Three definitions are used in the following discussions. First, the *hardware size* refers to the number of nodes in the specified network. Second, the *information size* refers to the number of optimal clusters into which a training set of P failure vectors can be classified. Third, the *hardware redundancy* refers to the difference between the hardware size and the information size. Thus, a network with a high degree of hardware redundancy has a large hardware size and a small information size.

The three categories of experiments study the clustering network and the classification network, focusing on the effects of hardware redundancy and hardware size. The impact of a fault is also affected by the learning algorithm used. In this study, we ignore the effects of the learning algorithm.

To study the effect of hardware redundancy, we start with a network which is much larger than the minimal. Recall that a minimal clustering network is one which has the same number of output nodes as the number of optimal clusters in the vector set. A minimal classification network is one which has a minimum number of nodes and yields no errors during classification. For each injected fault, both the percentage of mistagged vectors in the clustering network and the percentage of misclassifications in the classification were measured. The information size was then increased until it approached the hardware size. As the information size was increased, the amount of hardware redundancy decreased (since the number of clusters in the vector set increased). The impact of hardware redundancy was studied by plotting the average mistag and misclassification percentages as functions of information size.

In order to study the impact of hardware size, we conducted fault injections on minimal networks of increasing size. In a neural network, such as the three-layered perceptron model, we would expect that larger networks will have considerably more redundancy than smaller networks.[4] Again, for each injected fault, the percentage of mistagged vectors in the clustering network and the percentage of misclassifications in the classification network were measured. The mistag and misclassification percentages were plotted against increasing network size. The time taken for the network to stabilize and the steady state output values were also recorded.

[4]If we increase the number of nodes in a network by X, the number of links increases by K times X, where K is greater than one and is dependent on the number of incoming links in each node.

The input data for the experiments were obtained from an IBM mainframe system. The attributes consisted of ten resource usage parameters such as the CPU usage, the number of I/O operations per second, the number of disk accesses per second, etc. The size P of the training set was chosen to be 200. This number was selected to provide a statistically significant number of points in each cluster.[5] Four different sets of training vectors were used in our experiments. These are referred to in the subsequent discussion as the *experimental training sets*. The number of clusters in each set was six, ten, thirteen and seventeen, respectively. It will be seen that the results thus obtained, using these four experimental training sets, are sufficient to illustrate trends in mistag and misclassification percentages. The following three sections describe each of the three experiments in detail.

6.1. Experiment 1

In Experiment 1, the impact of faults on the clustering network was investigated. Figure 4 shows the sequence of events during each run of this experiment. At time t_0, the clustering process was initiated. Between times t_0 and t_1, the clustering process as defined in Section 3.1 was performed. Thus, at time t_1, an optimal set of link-weights, i.e., the optimal cluster centroids, was determined. During the interval $(t_1\ ,\ t_2)$, the training vectors were tagged. A fault was injected into a randomly selected location in the network at time t_1, and the impact of the fault on the tagging process was evaluated. Faults were injected during the tagging process (as opposed to during clustering) because the network is considered to be stable during this period. The impact of faults during the clustering process is not an

[5]Statistical cluster analysis was used to first determine the number of optimal clusters in each of the vector sets. Each set of vectors contained about two hundred observations; there were a sufficient number of vectors in all the clusters.

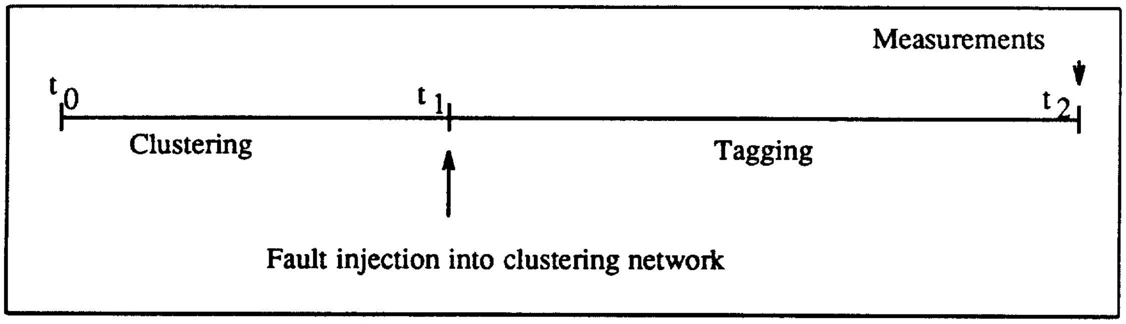

Figure 4. Sequence of events for Experiment 1.

interesting case because the effect of these faults is alleviated when the link-weights are iteratively modified.

The effect of the faults was measured by comparing the tagged vectors obtained from a fault-free network with those obtained from a faulty network, and by calculating the percentage of vectors which were mistagged. The time for the network to stabilize ($t_1 - t_0$) and the steady-state output values were also measured. Two hundred vectors were input during the clustering process and the measurements defined in Chapter 5 were recorded.

For each injected fault, the sequence depicted in Figure 4 was repeated. Twenty injections[6] were made for each fault type. Since there were 17 fault types, 340 (17×20) injections were performed for each set of input vectors. Given the four experimental training sets, in all, 2720 (2×4×340) injections were performed to investigate both the impacts of hardware redundancy and hardware size. It will be clear later that these injections are sufficient to show the major trends in the results. The next two subsections describe the experimental

[6]In our experiment, we found that an average of twenty repetitions gave a representative result.

procedures for studying the effects of varying the amount of hardware redundancy and the hardware size, respectively.

6.1.1. Impact of hardware redundancy

The objective of this experiment was to study the effect of decreasing the hardware redundancy in the clustering network. The decrease was achieved by fixing the size of the hardware and increasing the size of the information until the two approached each other, i.e, we fixed the size of the network and decreased the amount of redundancy. For example, with a network of K output nodes and a set of vectors with K optimal clusters, there is no node redundancy. However, if we use a set of vectors with L ($K > L$) optimal clusters, then there are (K-L) output nodes which provide the network with a certain degree of node redundancy.

For each network and the corresponding vector set, we performed the clustering process, fault injection, tagging and recorded the percentage of mistagged vectors. The percentage of mistagged vectors was plotted as a function of information size. A network with twenty output nodes[7] was used as the target for fault injections. The experiment was repeated with the four experimental training sets (with six, ten, thirteen and seventeen optimal clusters) in order to obtain the trend in mistag percentage as the amount of hardware redundancy was decreased.

[7]In the clustering network, the number of output nodes matches the number of clusters in the vector set. Recall that four sets of vectors with six, ten, thirteen and seventeen clusters, respectively, were used. The network has to accommodate the set with the largest number of clusters, i.e., the set with seventeen clusters. Recall also that the clustering network is modeled according to Kohonen's Feature Map, which, as stated in Section 3.2, has P output nodes arranged in a rectangle of size K by L (where $K \times L = P$). Therefore, the smallest network which can accommodate seventeen clusters is one which has twenty nodes (arranged in a rectangle of size five nodes by four nodes).

6.1.2. Impact of hardware size

In this experiment, we investigated the impact of faults as the size of the minimal clustering network was increased. Again, the four experimental training sets were used in this experiment. For each vector set, a corresponding minimal clustering network was used as the target for the fault injections. The corresponding sizes of the clustering network were six, ten, thirteen and seventeen, respectively. The percentage of vectors mistagged as a result of the injected faults was measured and plotted as a function of increasing hardware size.

6.2. Experiment 2

In this experiment, the impact of faults in the classification network was investigated. Figure 5 shows the sequence of events during each run of the experiment. As before, in the interval (t_0, t_2), the vectors were clustered and tagged. The set of tagged vectors was presented to the classification network in the interval (t_2, t_3) for the training process. The training process was completed at time t_3 when a set of optimal link-weights for the classification network was obtained. A fault was then injected at time t_3, just before the network started to classify the new inputs. During the interval (t_3, t_4), the impact of faults on

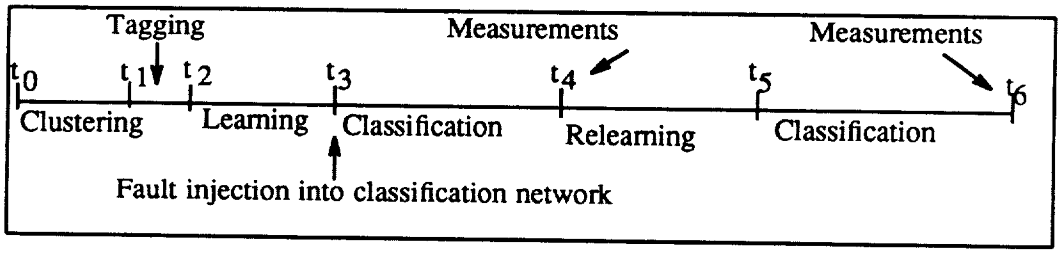

Figure 5. Sequence of events for Experiment 2.

the classification process was measured. In the interval (t_4, t_5), we attempted to retrain the faulty network by reapplying the set of two hundred training vectors. The number of misclassifications was again measured between times t_5 and t_6 to determine the improvement, if any, after the relearning process.

As in Experiment 1, twenty injections were made for each fault type and 340 injections were made for each training set (20 injections of 17 fault types). In all, 2720 ($2\times4\times340$) injections were made to investigate the effects of hardware redundancy and hardware size. The four experimental training sets in Experiment 1 were also used. The percentage of misclassifications, the learning duration (t_3–t_2), the relearning duration (t_5–t_4) and the steady state output values were measured The next two subsections provide a detailed description of the two procedures used to investigate the effect of hardware redundancy and hardware size, respectively.

6.2.1. Impact of hardware redundancy

In this experiment, the effect of decreasing hardware redundancy in the classification network was studied. The size of the classification network was fixed and the size of the information was varied. The percentage of misclassifications was then plotted against increasing information size.

The size of the network was fixed at ninety nodes: ten input nodes, thirty nodes in the first hidden layer, thirty three nodes in the second hidden layer and seventeen nodes in the output layer. This was the minimal network for the largest experimental training set (17 optimal clusters). The number of nodes in each layer was determined using Lippmann's method and verified via simulation. Again, the four experimental training sets, with six, ten, thirteen and seventeen clusters, respectively, were used. As the information size increased from six to seventeen, the amount of hardware redundancy in the ninety node network was decreased so as to determine the trend in the percentage of misclassifications as the hardware redundancy decreases.

6.2.2. Impact of hardware size

This experiment investigated the impact of faults as the size of the minimal classification network was increased. We injected faults into four minimal classification networks and measured the percentages of misclassifications. The percentages were plotted against increasing hardware size to illustrate the effect of increasing hardware size. The four minimal network configurations, shown in Table 1, corresponding to the four experimental training sets, one for each vector set, were used as target networks.

Table 1. Sizes of the minimal classification networks.

Classification Network					
No. of clusters training set	Input layer	Hidden layer 1	Hidden layer 2	Output layer	Total nodes
6	10	8	10	6	34
10	10	10	20	10	50
13	10	18	29	13	70
17	10	30	33	17	90

6.3. Experiment 3

This experiment investigated the effect of fault propagation between the two networks. In particular, we investigated the effect of a fault occurring in the clustering network during the tagging process. A fault was first injected into the clustering network and the percentage of mistagged vectors was measured. Then, the faulty set of tagged vectors was used to train the classification network and the percentage of misclassifications was then measured. The experiment was performed with increasing sizes of the minimal networks to determine the impact of hardware size. However, unlike Experiments 1 and 2, the impact of hardware redundancy is not studied because the focus of this experiment is on the worst case impact of fault propagation.

Figure 6 shows the sequence of events for a single run of this experiment. During times t_0 and t_1, the clustering network was used to cluster a set of training vectors. When the network has stabilized, a fault was injected and the vectors were then tagged with the result from the faulty network. The resulting set of training vectors were then used to train the classification network (between times t_2 and t_3). When the classification network has

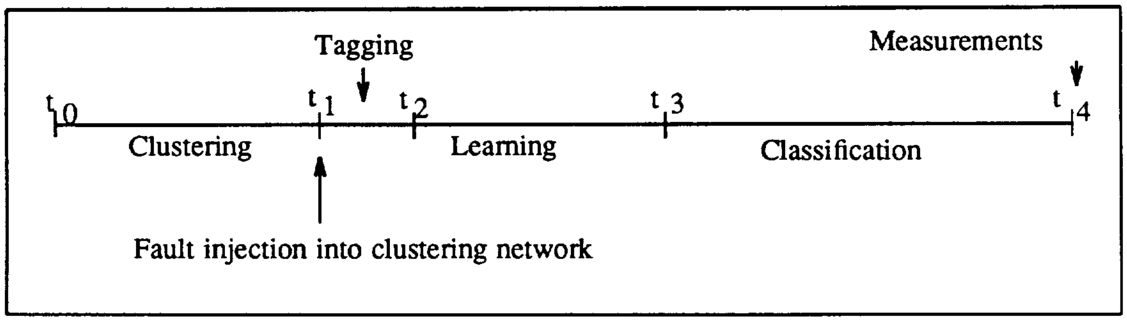

Figure 6. Sequence of events for Experiment 3.

stabilized, the accuracy of the classification was measured (between times t_3 and t_4). As in Experiments 1 and 2, the learning duration ($t_3 - t_2$), the output values, the percentage of mis-tagged vectors, and the percentage of misclassifications were measured. Each experiment consisted of 340 fault injections and the experiment was repeated with the four experimental training sets. In all, 1360 (4×340) fault injections were performed in this experiment.

CHAPTER 7.

ANALYSIS OF THE CLUSTERING NETWORK

This chapter reports the results obtained from Experiment 1 in which the impact of faults in the clustering network was studied. Section 7.1 investigates the impact of faults, focusing on the mistag percentages for the clustering network. Section 7.2 investigates the effect of hardware redundancy redundancy on the mistag percentages. Section 7.3 examines the trend in the mistag percentage with increasing hardware size.

7.1. Impact of Faults on the Clustering Network

Recall that in this experiment, faults were injected during the tagging phase. Table 2 shows the mistag percentages for all 17 fault types using a ten-node network (the results for the four clustering networks were all similar).

On the average, 25% of the training vectors were mistagged. Since the network is minimal, the results presented may be interpreted as the worst case, i.e., the mistag percent will only decrease, as the number of nodes is increased. Permanent node-input faults (noise and glitches at the node input) result in the largest mistag percentage (37%). The reason for this high percentage is the single-layered structure of the network. The components of each

Table 2. Impact of faults on the clustering network.

Clustering Network		
Faults		Percentage of erroneous tag
Link faults:	Transient	12%
	Permanent	10%
Node input:	Transient	27%
	Permanent	37%
Node memory:	Transient	23%
	Permanent	19%
Node offset:	Transient	32%
	Permanent	27%
	Byzantine	26%
Node response delay:		
	Transient	27%
	Permanent	22%
	Byzantine	26%
Node gain:	Transient	22%
	Permanent	24%
	Byzantine	21%
Full-node faults:	Transient	32%
	Permanent	29%
Average		25%

input vector are linked directly to the inputs of each output node. Thus, when a node input is perturbed, the clustering network "sees" a new input vector, and the possibility exists that the network may assign this new vector to another cluster.

The table also shows that permanent link faults resulted in only 10% of the input vectors being mistagged. The reason for this low percentage (cf., a broken link) is due mainly to the considerable link redundancy in the network. For example, if the link between the i^{th} input and the j^{th} output is broken, the information from the i^{th} input can still be passed to other output nodes via the other outgoing links of the i^{th} input node.

The importance of determining the mistag percentage lies in the fact that if training vectors are assigned incorrect tags, the classification network will not be trained correctly. Thus, a possibility exists that the failure diagnostic system will produce an incorrect classification of a new failure vector.

7.2. Impact of Hardware Redundancy

In this section, we examine the effect of hardware redundancy in the clustering network. A network with twenty output nodes, together with the four experimental training sets (with six, ten, thirteen and seventeen optimal clusters, respectively), was used to study the impact of decreasing hardware redundancy.

Figure 7 plots the percentage of mistagged vectors versus the number of clusters in the training set. The average percentages for both link and node faults are used in the figure. Two observations can be drawn from this figure. First, link faults have a lower impact than node faults. The maximum mistag percentage for link faults is 7.5% versus 26% for node faults. The minimum percentage for link faults is 2.5% versus 16% for node faults. This is clearly due to the considerable amount of link redundancy in the network.

Second, the mistag percentage increases as the number of optimal clusters in the training set increases. The rate of increase is smaller for link faults as opposed to node faults.

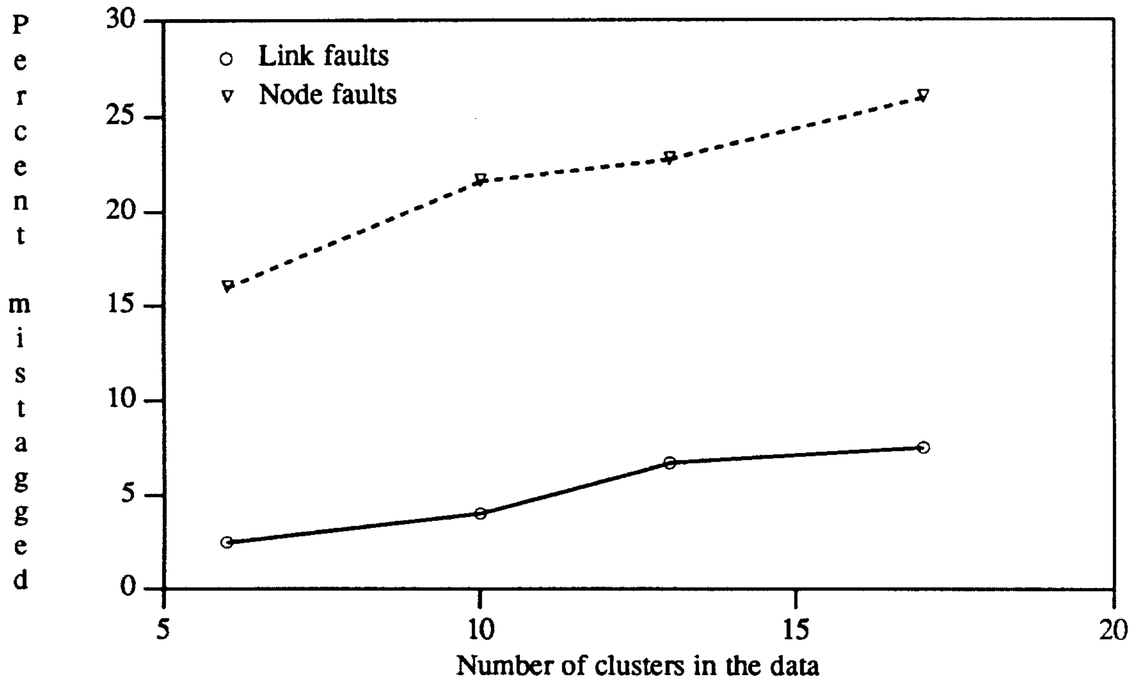

Figure 7. Effect of redundancy in the clustering network.

The average rate of increase for link faults was 0.56 (%/cluster) and the corresponding rate for node faults was 2.6 (%/cluster). The reason that the impact of link faults is relatively less than node faults is due to the higher degree of redundancy of the interconnection links. In the target clustering network, each node has ten incoming links. If a node fails, ten links will be affected and ten nodes at the other end of the link will receive incorrect information. However, if a single link fails, only one node will receive incorrect information.

Although the absolute value of the mistag percent increases with the information size, the ratio of the mistag percent to the information size is more or less constant for all information sizes. The linear relationship between the mistag percentage and the amount of hardware redundancy indicates that the penalty per unit decrease in hardware redundancy is constant.

7.3. Impact of Hardware Size

This experiment studied the trend in the mistag percentage as the size of minimal clustering networks increased. Four minimal networks corresponding to the four experimental training sets were used. Figure 8 plots the average percentage of mistagged vectors versus the number of nodes in the minimal clustering network.

The figure shows that the impact of link faults on the mistag percent is significantly less than that of node faults. The maximum mistag percent for link faults is 19% versus

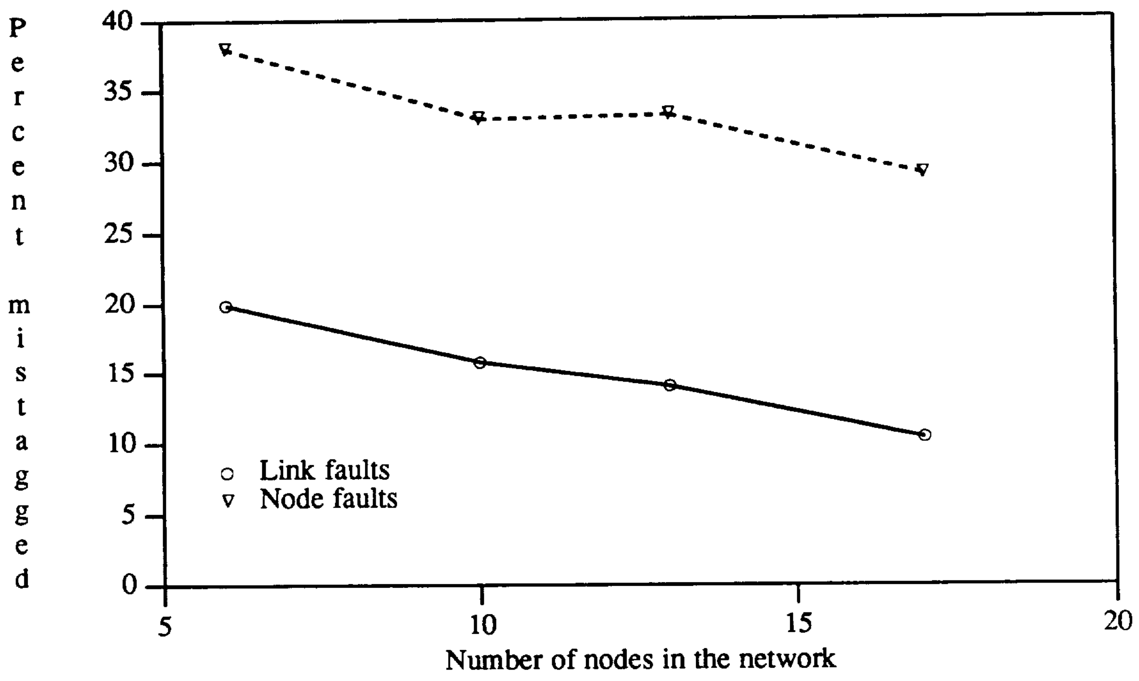

Figure 8. Impact of faults in minimal clustering networks.

36% for node faults. The minimum mistag percent for link faults is 10% versus 29% for node faults. The reason is again due to the high degree of link redundancy in the network.

Figure 8 shows a decreasing trend in the mistag percentage for both link and node faults, as the hardware size increases. The average rates of decrease of mistag percentage for link and node faults are 0.84 and 1.35 (%/node), respectively. We may explain the decreasing trend as follows: If there are P input vectors and K output nodes, then, on the average, each node represents P/K vectors. Thus, a faulty node affects an average of P/K input vectors. If the network size K is increased, the corresponding value of P/K decreases. For example, a node failure in a twenty-node network affects only 5% of the vectors. However, a similar failure in a five-node network affects 20% of the vectors.

The rates of decrease of mistag percent for both link and node faults are also approximately constant, i.e., increase in mistag percent per unit decrease in the hardware size is constant. Although the absolute mistag percent for a larger network is small, the relative ratio of the network size to the hardware size is the same for all networks.

CHAPTER 8.

ANALYSIS OF THE CLASSIFICATION NETWORK

This experiment investigated the impact of faults in the classification network. Section 8.1 evaluates the percentage of misclassifications in a faulty network. Section 8.2 examines the impact of faults on the relearning duration. Sections 8.3 and 8.4 focus on the effects of hardware redundancy and hardware size, respectively, on the misclassification percentage.

8.1. Impact of Faults on Misclassification

In this experiment, faults were injected during the recognition phase and the resulting misclassifications obtained from a three-layered classification network were measured. This section presents the results for a network consisting of ten input nodes and ten output nodes, with ten nodes in the first hidden layer and twenty nodes in the second. The results for the other three networks and the corresponding experimental training set are similar. Note that a misclassification is said to occur if a vector is assigned to an incorrect cluster (input misclassification) or if the difference δ is less than the threshold θ (output misclassification). The next two subsections discuss input and output misclassifications.

8.1.1. Input misclassification

Table 3 summarizes for each fault type, the percentage of input vectors misclassified before and after the relearning phase. Results show a clear decrease in misclassification after relearning. On the average, 24% of the input vectors were misclassified before relearning while only 11% of the vectors were misclassified after relearning.

Table 3. Percentage of input misclassifications in the classification network.

Classification Network			
Faults		Percent of misclassifications before relearning	Percent of misclassifications after relearning
Link faults:	Transient	7%	2%
	Permanent	6%	3%
Node input:	Transient	18%	12%
	Permanent	22%	12%
Node memory:	Transient	17%	9%
	Permanent	14%	4%
Node offset:	Transient	27%	9%
	Permanent	26%	11%
	Byzantine	28%	15%
Node response delay:			
	Transient	24%	14%
	Permanent	24%	12%
	Byzantine	29%	15%
Node gain:	Transient	20%	12%
	Permanent	19%	10%
	Byzantine	21%	12%
Full-node faults:			
	Transient	32%	16%
	Permanent	35%	21%
Average		24%	11%

It may be argued that, for some applications, the network behavior before the relearning phase can be ignored. Thus, once a fault is detected, the network outputs just prior to the fault detection may be discarded. If we consider only the post-relearning phase, Table 3 shows that both permanent and transient full-node faults cause the highest percentage of misclassifications. For permanent full-node faults, 21% of the inputs were misclassified after the relearning phase and for transient full-node faults, 16% of the inputs were misclassified. The reason for the high failure percentage for full-node faults is that the hardware redundancy (in comparison with the information or link redundancy) is reduced. Since minimal networks were used, a single node fault reduced the number of nodes to below that required for the specific experimental training set.

Link faults have a relatively low impact on misclassifications. Only 3% of the input vectors were misclassified after the relearning phase. This behavior is similar to that of the clustering network. Clearly, if a link (incoming or outgoing) from a node fails, the information can still be transmitted via other links.

Node memory faults, node input faults, node offset faults and node gain faults all have moderate impact on misclassification. For example, a transient node input fault causes 18% misclassification before and 12% after relearning. These faults, unlike full-node faults, do not cause reduction in hardware redundancy. The effect is only a perturbation in the interconnection link-weights for node memory faults, or the node output values for node input, offset and gain faults. Therefore, the impact is less severe than that of a full-node fault (21%

misclassifications). However, the faulty output value caused by a node fault is transmitted to more than one node. Hence, the impact is more severe than that caused by a link fault.

Intuitively, one might expect transient faults to have a significantly lower impact than permanent faults since the duration of the fault is shorter. However, the results in Table 3 indicate that some transient faults have higher misclassification percentages than permanent faults. A fault, transient or permanent, affects the link-weights of the network or the information flowing through the network. If the link-weights are disturbed, the network performs the classification with respect to a new set of cluster centroids. If the information is perturbed, the network "sees" a new vector, and this new vector may be assigned to another cluster. Thus, the cause of the misclassification can be attributed to cluster centroids being shifted or the information being distorted. It is usually not possible to determine which of these causes will predominate. For example, a permanent fault could disturb the centroid slightly or perturb the input slightly such that, only a few vectors are misclassified. However, a transient fault could disturb the centroid or the information drastically such that all the vectors in a specific cluster are misclassified. Therefore, a transient can have more of an effect than permanent faults. Another reason for this is due to the fact that the limit on the relearning duration was set to 600 cycles (chosen to be equal to the average learning duration). A longer relearning duration may allow the effects of the transient to die off. However, this is clearly inefficient since it would be better to issue a restart.

8.1.2. Output misclassification

This subsection presents the results concerning output misclassification. In this experiment, the threshold θ was initially chosen to be 0.8. This value provides a high degree of confidence in assigning the vector to the specific output node. Recall that the difference δ between the two highest output values indicates the degree of confidence of assigning a vector to the specific cluster.

Table 4 shows, for each fault type, the percentage of vectors with $\delta<0.8$. It can be seen that permanent full-node faults have the greatest potential of causing output misclassification (19% of the vectors cannot be confidently classified into a unique cluster after the relearning phase). Permanent link faults have the least impact; only 1% of the vectors were misclassified.

A comparison between Table 3 and Table 4 shows that about 90% of the misclassifications were due to the difference δ being less than 0.8. For example, after relearning, the average percentage of vectors misclassified was 11%, but 9% were due to $\delta<0.8$. Figure 9 shows the distribution of δ for this experiment.[8] It shows two modes, one centered around 0.9 and the other at 0.15. The low mode is not significant since only 1% of the misclassifications have $\delta<0.2$. In the second mode, 8% of the misclassifications have δ between 0.6 and 0.7 and the remaining 92% have δ greater than 0.8.

[8]In one set of experiments, seventeen trials were conducted. During each trial, twenty runs of the experimental sequence described in Section 6.2 were carried out. Since each run uses two hundred vectors, we have 68000 values for δ.

Table 4. Percentage of misclassifications due to output misclassification.

Classification Network			
Faults		Percent of misclassifications before relearning	Percent of misclassifications after relearning
Link faults:	Transient	6%	2%
	Permanent	3%	1%
Node input:	Transient	12%	12%
	Permanent	17%	9%
Node memory:	Transient	10%	9%
	Permanent	12%	2%
Node offset:	Transient	20%	9%
	Permanent	21%	7%
	Byzantine	18%	9%
Node response delay:			
	Transient	17%	10%
	Permanent	19%	9%
	Byzantine	19%	11%
Node gain:	Transient	12%	10%
	Permanent	19%	13%
	Byzantine	15%	12%
Full-node faults:			
	Transient	27%	14%
	Permanent	25%	19%
Average		26%	9%

It may be argued that even if the difference δ is less than the threshold of 0.8, the output can still be used if δ is sufficiently large (for example, if δ=0.6). This can be achieved by redefining the threshold [10]. This approach is somewhat optimistic and does not take into account issues such as the time taken for threshold redefinition and the electrical impact of low output values on the rest of the system. Assuming threshold redefinition is possible at little cost, the results in Figure 9 indicate a decrease in the percentage of misclassifications from 9% to 0.9% when the threshold is decreased from 0.8 to 0.6.

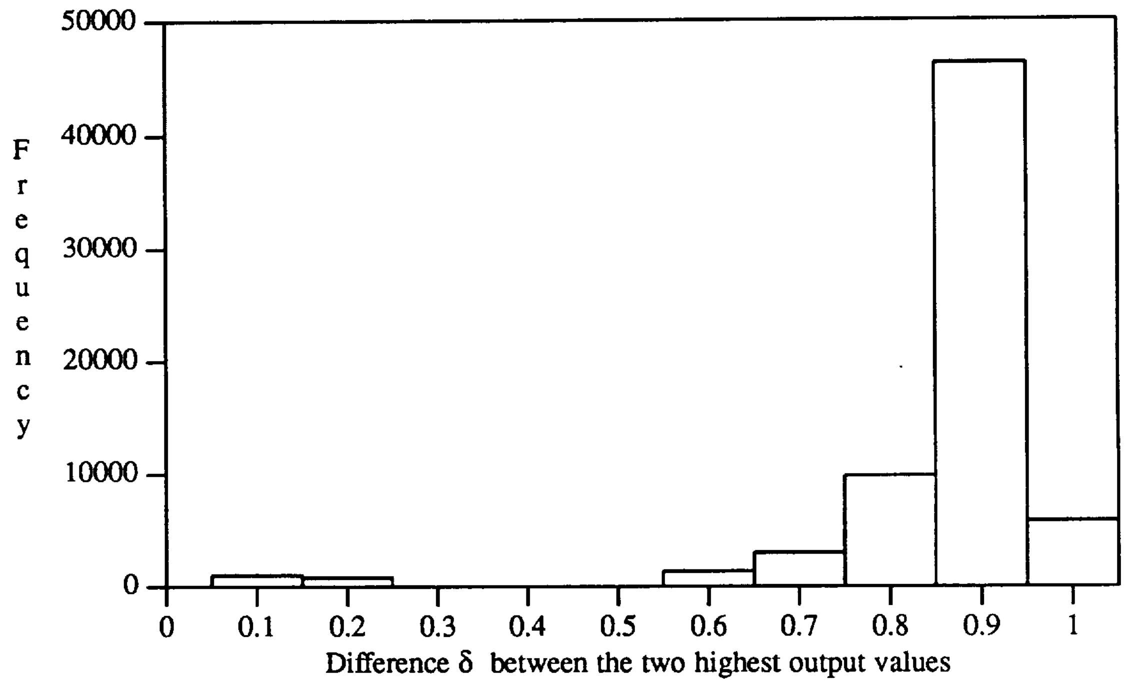

Figure 9. Distribution of the difference between the two largest output values.

8.2. Impact of Faults on the Relearning Duration

Figures 10 and 11 show the impact of a selected set of transient link and node faults on the relearning duration. The results excluded from Figures 10 and 11 are shown in Appendix D. The numerical values presented in the following, however, include the results of both Figures 10, 11 and the appendix.

The figures plot the number of simulation cycles required to stabilize the link-weights, versus the duration of the transient. On the average, as the duration of a transient increases, the trend is for an increase in the number of relearning cycles necessary to obtain stable link-weights. For both transient link faults and transient memory faults (random link-weight

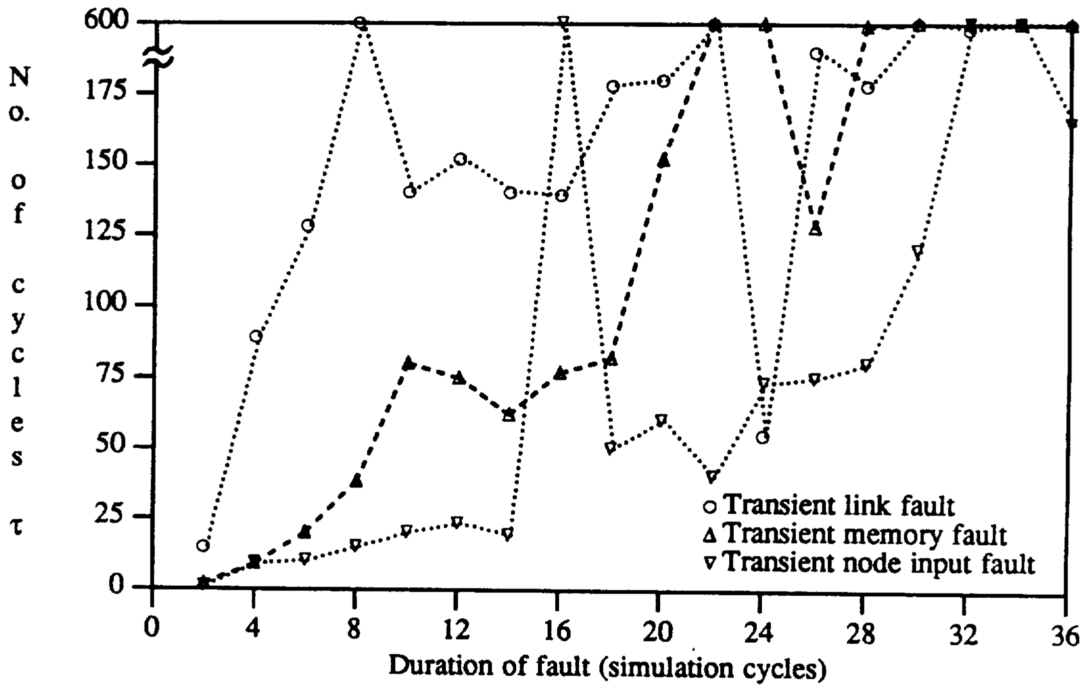

Figure 10. Impact of transient link, memory and node input faults on the relearning duration.

perturbation), the network becomes increasingly unstable as the duration of a transient increases (Figure 11). In general (considering all transient faults), the number of relearning cycles required after a transient is highly unpredictable, varying from a low of 0 (with a probability of 0.05) to a high of more than 600 cycles (with a probability of 0.23). On the average, 24% of the experiments failed to obtain a set of stable link-weights within 600 cycles during the relearning phase (21% for link faults and 27% for node output faults). We define a network which is unable to obtain a set of stable link-weights within 600 cycles to be unstable. This choice was made so as to be slightly greater than the average learning duration of about 550 cycles.

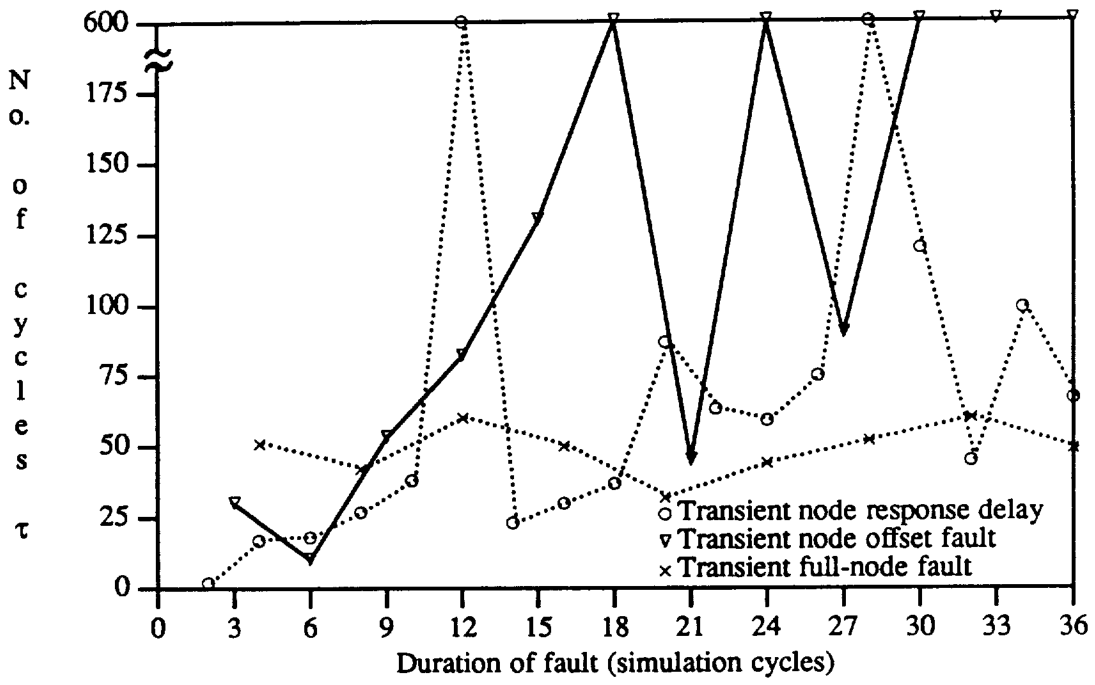

Figure 11. Impact of transient node faults on the relearning duration.

The main reason for the instability due to the transients lies in the manner in which these networks perform the optimization, i.e., error minimization. Transients sometimes drastically perturb the network and throw the minimization onto a totally different path leading to instability. However, a transient may also throw the minimization onto a path leading rapidly to a minimum. Results also indicate that 22% of the cases have transients of long durations but required relatively few relearning cycles. On the average, the network required about 50 relearning cycles.

Figure 12 shows the impact of permanent faults on the relearning duration. In this case, for a given set of input vectors, a fault was injected into a randomly selected location

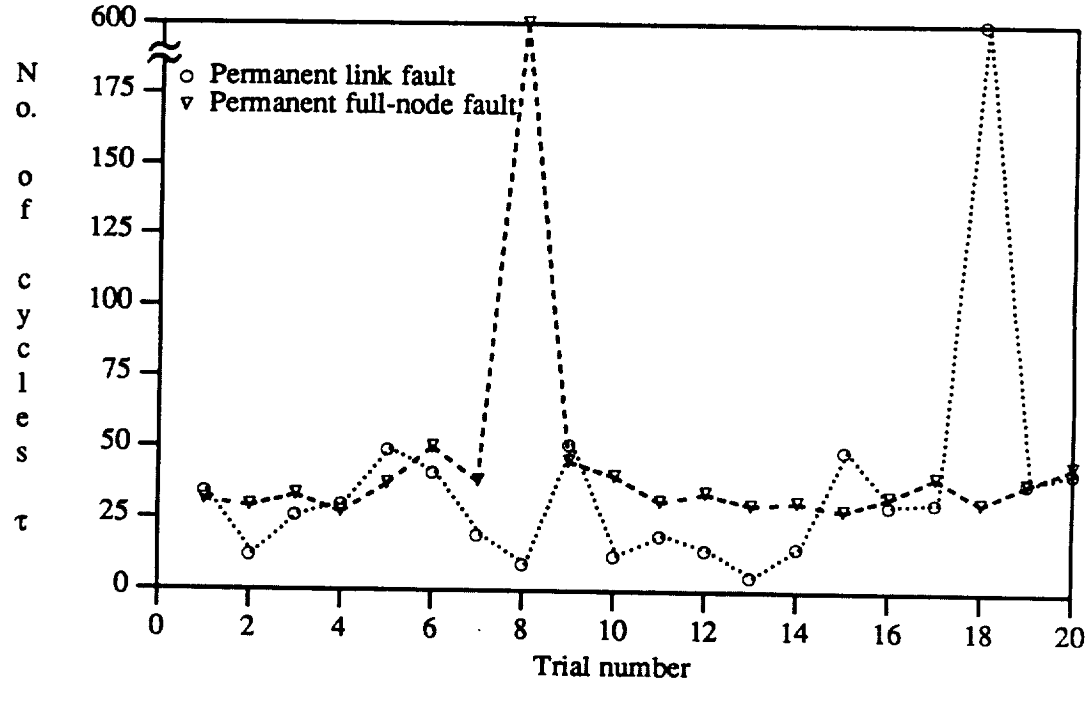

Figure 12. Impact of permanent link faults and full-node faults on the relearning duration.

(node or link) and the time to stabilize was measured. The figure plots the number of simulation cycles needed to stabilize the output for each of the twenty runs of two trials:[9] one for permanent link faults and the other for permanent full-node faults. The figure shows that permanent full-node faults and permanent link faults do not have a significant impact on the relearning duration. The relearning duration is more or less constant at about 35 cycles, except for two instances (one each for the link and node faults) where the network was unstable.

A comparison of the results between permanent and transient faults shows that the relearning phase for permanent faults ranges between 35 ±10 cycles, which is lower in comparison with a range of 45 ±20 cycles for transient node faults. Thus, the relearning duration after a permanent fault is far more predictable than the relearning duration after a transient fault. The difference in behavior between the transients and permanent faults is most likely due to the fact that, for a permanent fault, the system has to cope with only a single disturbance. For a transient, however, it has to cope with two relatively close disturbances (one when the fault is injected and the other when the transient dies off).

[9]In Section 6.2, we have defined a trial to consist of twenty runs of a certain experiment sequence with the injection of one type of fault.

8.3. Impact of Hardware Redundancy

In this section, we discuss the trend in misclassification as the hardware redundancy decreases in a fixed-size network. A network with 90 nodes was used as the target network. This is a minimal network for the experimental training set with 17 optimal clusters. Figure 13 shows the average percentage of misclassifications for link and node faults as the information size increases from six to seventeen.

The results are similar to those found in Section 7.2. A detailed analysis here would be repetitive. In summary, we found that link faults have a lower impact than node faults. The misclassification percentage for link faults ranges from 1.1% to 6.2%. For node faults,

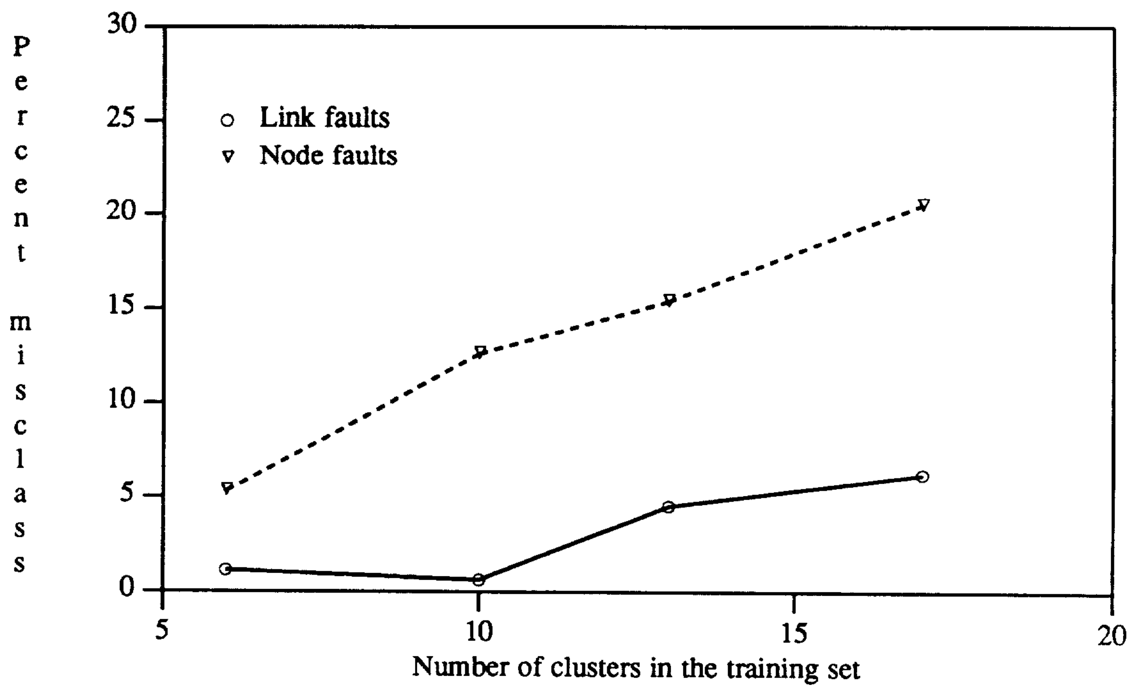

Figure 13. Effect of redundancy in the classification network.

the misclassification percentage ranges from 5.3% to 20.5%. This is clearly due to the high degree of link redundancy in the network. The rates of increase of the misclassification percentage with increasing information size for link faults and node faults were quite constant at 0.34 and 1.1 (%/cluster), respectively. Thus, the penalty for decreasing the amount of hardware redundancy is more or less constant.

8.4. Impact of Hardware Size

In this section, the impact of hardware size on misclassifications was investigated. Figure 14 plots the percentage of misclassifications versus the number of nodes in the four minimal classification networks (corresponding to the four experimental training sets).

The results obtained here are similar to those in Section 7.3. From the figure, the misclassification percentage decreases as the size of the network increases. For link faults, the percentages range from 2.7% to 9.5%, and for node faults, the percentages range from 12.7% to 24.4%. This result again indicates that link faults have a lower impact than node faults because of the high link redundancy. The rates of decrease of the misclassification percentage with increasing hardware size were found to be 0.15 (%/node) for link faults and 0.24 (%/node) for node faults. Thus, larger networks were found to have less impact on faults. However, the constant rates of decrease indicate that the improvement in misclassification percentage is constant.

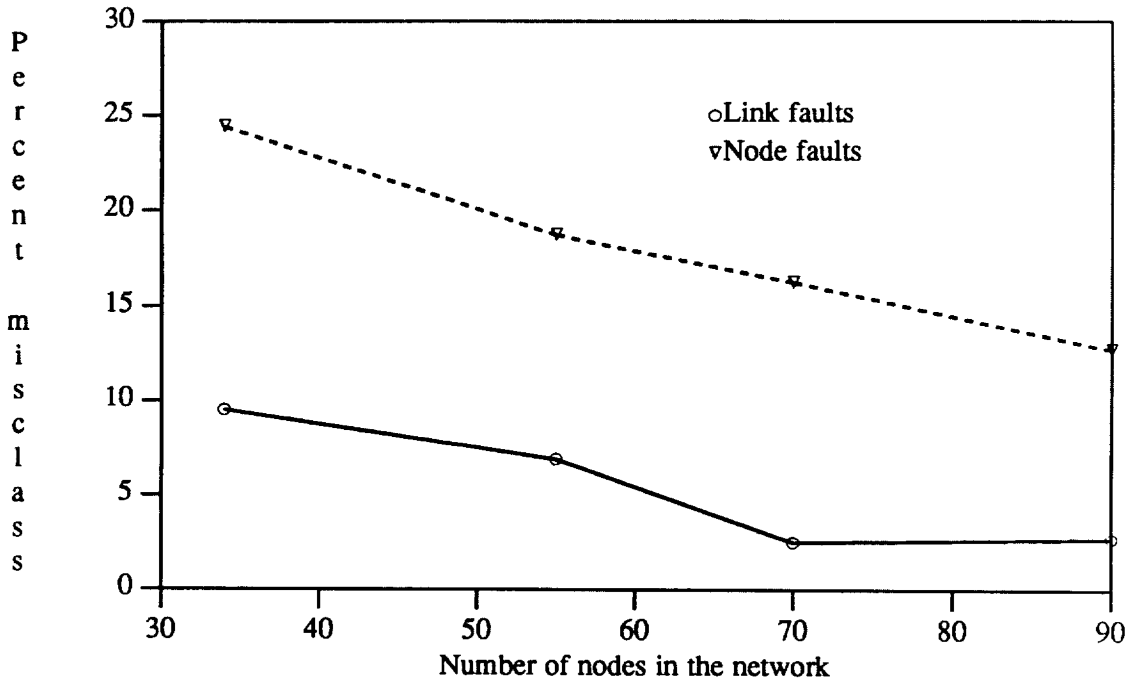

Figure 14. Effect of the size of the classification network.

CHAPTER 9.

ANALYSIS OF FAULT PROPAGATION

In Experiment 3, faults were injected into the clustering network, and the impact of faults on both the clustering network itself and on the classification network was investigated. First, faults were injected into the clustering network, and the percentage of vectors which were mistagged was measured. Second, the tagged vectors from the faulty clustering network were used to train the classification network, and the percentage of misclassifications was subsequently measured. Table 5 shows these percentages for each type of faults injected.

The average percentage of misclassifications was 17%. This is, however, lower than the average percentage of incorrectly tagged vectors (25%). Intuitively, one would expect the two percentages to be similar. In reality, a decrease of 8% in the percentage of misclassifications was observed. Although these results appear to suggest that a self-correction process exists between the two networks, the following results indicate otherwise.

Table 5. Impact of fault propagation between the clustering and the classification network.

Clustering Network and Classification Network			
Faults		Percent of erroneous tags	Percent of misclassifications in classification network
Link faults:	Transient	10%	7%
	Permanent	12%	10%
Node input:	Transient	27%	18%
	Permanent	37%	25%
Node memory:	Transient	23%	17%
	Permanent	19%	16%
Node offset:	Transient	32%	24%
	Permanent	27%	21%
	Byzantine	26%	17%
Node response delay:			
	Transient	27%	18%
	Permanent	22%	19%
	Byzantine	26%	21%
Node gain:	Transient	22%	14%
	Permanent	24%	20%
	Byzantine	21%	13%
Full-node fault:			
	Transient	32%	24%
	Permanent	29%	21%
Average		25%	17%

Four minimal clustering and classification networks were studied to observe the effect of increasing hardware size. Table 6 reports the average percentages of mistag and misclassified vectors obtained from the four experiments. Not all the results suggest the existence of a self-correction process between the two networks. For the network with 60 and 107 nodes, the misclassification percentage is lower than the mistag percentage. However, the other two cases indicate the opposite relationship. Although the individual mistag percentages and the misclassification percentages for the clustering and classification network, respectively, are consistent with the results presented in the previous chapters, these four experiments alone, were not able to show a significant trend in the impact of fault propagation between the two networks.

Table 6. Impact of fault propagation on different sized networks.

Clustering Network and Classification Network		
No. of nodes	Average percent of erroneous tags	Average percent of misclassifications in classification network
60	25%	17%
75	23%	25%
92	19%	24%
107	20%	18%

CHAPTER 10.

CONCLUSIONS

In this thesis, simulation experiments were conducted to quantify the effect of a variety of faults on a neural network. The target network consisted of a single-layered clustering network and a three-layered classification network. The percentage of vectors mistagged by the clustering network, the percentage of vectors misclassified by the classification network, the time taken for the network to stabilize, and the output values were all measured.

The results showed that both transient and permanent faults have a significant impact on the performance of the network. The corresponding mistag and misclassification percentages were typically within 5% to 10% of each other. In addition, transient faults were found to cause the network to be increasingly unstable as the duration of a transient was increased. This is because a network affected by a transient has to cope with two disturbances, one during the onset of the fault and the second, during the period when the fault dies off.

The average mistag percentage and the average misclassification percentage were both about 25%. After relearning, the percentage of misclassifications was reduced to 9%. The impact of the link faults was relatively insignificant in comparison with node faults (1% versus 19% misclassified after relearning). This is because of the considerable amount of information redundancy provided by the massive interconnections.

A study of the impact of hardware redundancy showed a linear increase in the mistag and misclassification percentages with decreasing hardware redundancy. The penalty per unit decrease in the hardware redundancy remained constant. In comparison with link faults, node faults resulted in three times the mistag and misclassification percentages. A study of the impact of hardware size showed that the mistag and misclassification percentages linearly decreased as the minimal hardware size was increased. Thus, large networks resulted in lower mistag and misclassification percentages than small networks. Node faults were again found to cause about three times the percentage of mistags or misclassifications as compared to link faults.

In this thesis, the fault characteristics of the neural network are assumed to be independent of the learning algorithm. However, this is not necessarily true. The learning algorithm and the network paradigm are important factors contributing to the fault tolerance of neural networks. Future research should investigate various learning algorithms and network structures to determine if the above results are generally true. In addition, with neural networks being implemented as VLSI chips, it is also important to study the networks from the electrical point of view. Current simulators could be extended to simulate nodes and links as physical components such as amplifiers and metal interconnects on VLSI ANN implementations.

APPENDIX A.

KOHONEN'S FEATURE MAP

The clustering network is a single-layered network, with a plane of nodes connected as shown in Figure 15. This is a simplified model of Kohonen's Feature Map [5]. The following is a brief account of the algorithm used in this clustering network.

Each of the nodes in the network represents a cluster centroid. The incoming link-weights to each node represent the coordinates of the cluster centroid. The function of each node computes the Euclidean distance between the input vector and the cluster centroid, i.e., between the link-weight and the input vector.

During the clustering phase, the set of training vectors is repeatedly presented to the network. For each presentation, the output node with the smallest Euclidean distance is chosen to be the active node. This is equivalent to assigning the input vector to the closest cluster. Weights to this node are adjusted in a fashion similar to that in a statistical clustering algorithm such as the k-means algorithm [20]. The weights are adjusted to accommodate the new input vector. This is similar to recalculating the cluster centroid after every iteration in the k-means clustering algorithm.

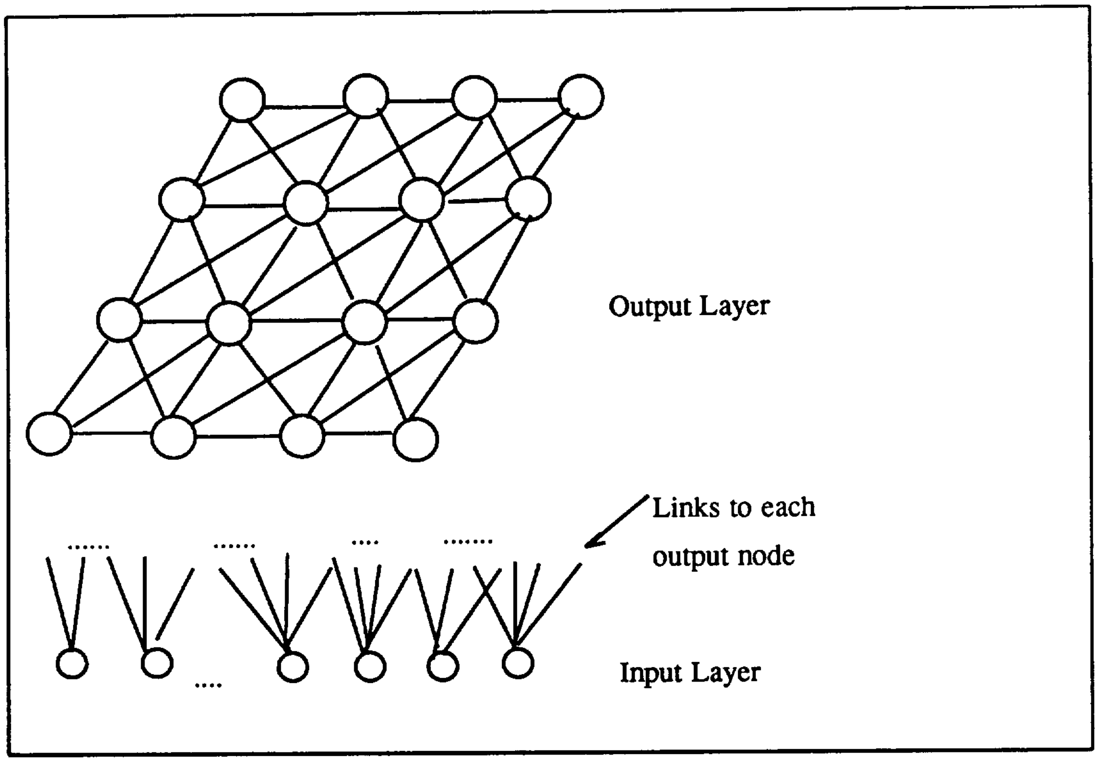

Figure 15. The clustering network implemented as Kohonen's Feature Map.

The clustering algorithm consists of five steps:

Step 1: Construct a neural network of size (K, N) where K denotes the number of clusters and N represents the number of attributes in a vector. Let w_{ij} denote the weight of the interconnection link between output node i and the input node j. Initialize the weights w_{ij} to small random values. This is similar to the choosing of a starting point in a clustering algorithm [20].

Step 2: Present a new vector x_i to the input nodes (i denotes the ith observation vector).

Step 3: Compute the distance d_j between the input vector and all the output nodes.

$$d_j = \sum_{i=0}^{K-1} (x_i - w_{ij}(t))^2$$

Step 4: Select the output node j^* with the minimum d_j and modify the link weights by

$$w_{ij}(t+1) = w_{ij}(t) + \alpha(t)(x_i - w_{ij}(t)) \quad \text{for j} \in NE_j^*(t)$$

$NE_j^*(t)$ is a Euclidean distance neighborhood around node j^* which decreases with time. The gain term $0 \le \alpha(t) \le 1$ decreases with time. The gain term increases the rate of change of the weights to achieve faster convergence.

Step 5: Goto Step (2)

The network adaptively changes the centroids in Step (4) until an optimal or local value is found. In this experiment, we use this network to cluster sets of two hundred training vectors. Once the clustering is completed (link-weights are stablized), each vector is tagged with a number denoting the cluster to which it belongs. For example, the tagged vector (0.24, 0.87, 0.65, 0.77 1) denotes that the vector (0.24, 0.87, 0.65, 0.77) belongs to cluster 1. These tagged vectors are then used during the training phase on the classification network.

APPENDIX B.

THREE-LAYERED PERCEPTRON MODEL

The classification network is a three-layered perceptron network [6]. It consists of an input layer, two hidden layers and an output layer. The number of nodes in the input layer equals the number of attributes in the training vector. The number of output nodes equals the number of clusters present in the training set. The number of nodes in each hidden layer depends on the training set.

In general, the worst-case number of nodes in the second hidden layer is equal to the number of disconnected clusters in the vector space [2]. The number of nodes in the first hidden layer is about three times that of the second hidden layer. In our experiment, Lippmann's method was used to initially determine the size of the network. A considerable amount of simulation was then conducted to obtain the configuration which minimizes the number of nodes.

The classification network is trained using the back-propagation algorithm along with the generalized delta rule [6]. The network is trained so that the output node representing the desired cluster produce a value of one when an input vector belonging to that cluster is

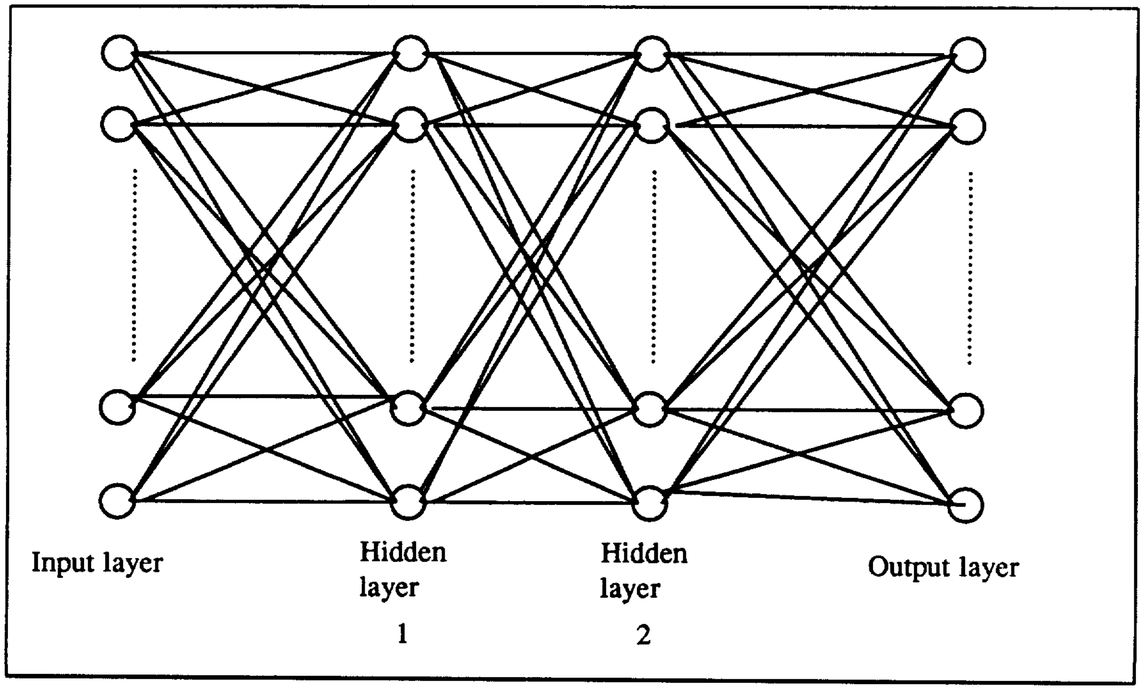

Figure 16. A three-layered perceptron model.

presented. The rest of the output nodes should produce a value of zero. For example, in a system with four attributes and six clusters, the output node would be (0, 0, 1, 0, 0, 0) if the input vector belonged to cluster 3. However, the minimization process leads to some output that is close to one or zero. Hence the above-mentioned output vector might look like (0.07, 0.1, 0.95, 0.1, 0.05, 0.11).

The back-propagation training algorithm using the generalized delta rule has five steps:

Step 1: Initialize the weights to small random values.

Step 2: Present input x_i and specify the desired output d_j which is given by the tag to each vector. The desired output node is set to 1 and the rest are set to 0.

Step 3: Using a sigmoid function, calculate the actual output y_j .

Step 4: Recursively change the link-weights between the nodes of the successive layers (this is the back-propagation step). The weights are adjusted by the following

$$w_{ij}(t+1)=w_{ij}(t)+\eta\delta_j x_i^*+\alpha(t)(w_{ij}(t)-w_{ij}(t-1))$$

If the node i is an output node, then

$$\delta_j=y_j(1-y_j)(d_j-y_j)$$

If the node j is a hidden node, then

$$\delta_j=x_j^*(1-x_j^*)\sum_k \delta_k w_{ij} \ ,$$

where k is all nodes above layer j and x_j^* denotes an input to a node not at the input layer.

The momentum $0\le\alpha(t)\le 1$ is used for faster convergence. As usual, it decreases with time.

Step 5: Repeat Step (2)

This method is analogous to the discriminant analysis approach in multivariate statistics. Specifically, in classical discriminant analysis classifiers, a discriminant matrix can be obtained such that every x_i is mapped into an optimal cluster. [21] has shown that the back-propagation training algorithm produces a matrix, which is equivalent to the discriminant analysis matrix, that transforms x_i to a specific cluster.

APPENDIX C.

FANS: A FAULT ANALYSIS TOOL FOR NEURAL NETS

C.1. Introduction

Computer simulation is the most common method used to implement and study artificial neural networks. Although analog VLSI (Very Large Scale Integration) technology has been used to fabricate some artificial neural networks, the size of these networks have been too small to realize any practical applications. Hence, neural network simulators remain a major tool for investigating these networks. As physical device implementation of artificial neural networks increases in number with the advancement of analog VLSI techniques, it is essential to study the impact of physical limitations and hardware faults in neural networks. Thus, a simulation environment for neural network simulation, as well as for fault injection, is necessary in the study and design of artificial neural networks.

This appendix describes the features, operations and design of FANS (the acronym for a Fault Analysis tool for Neural Networks). The simulator is designed to operate under the

UNIX[10] operating system and is written in the C programming language. Currently, it is customized to simulate the multilayered perceptron network [6] and Kohonen's Feature Map [5]. The fault models available are discussed in Chapter 4. The simulator can be easily enhanced to simulate other network and fault models. Performance measures such as the network training duration, the accuracy of a classifier and the output levels are automatically monitored.

A major objective of FANS is to provide the user with flexibility in specifying and studying the impact of faults in neural networks. The user must specify the network structure, the learning process and the fault injection process. The user does not need to intervene until the simulation is completed and the measurements are taken. The user specifications, along with the necessary training vectors, serve as the input to the simulator. Measurements include network parameters, such as link-weights and node activation levels, and FANS is able to display them off-line via graphs. This is particularly useful in studying the real-time behavior of the link-weights and the node values.

This appendix is organized as follows. Section C.2 provides an overview of related neural network simulators. Section C.3 presents an overview of the simulator structure. Section C.4 is the user's guide and Section C.5 presents two examples which use FANS; one simulating a three-layered perceptron network and the other, a Kohonen's Feature Map.

[10]UNIX is a trademark of AT&T Bell Laboratories.

C.2. Related Research

The recent surge of interest in neural networks has spurred the development of many simulators. These simulators come in a wide spectrum of capabilities and performance. For example, MacBrain, which is a neural network simulator on the Macintosh personal computer, provides users with excellent graphics to display network topologies and simulations. However, most of the simulators designed for personal computers have specific functions. For example, BrainMaker, developed by California Scientific Software, supports up only five types of processing nodes. These simulators are suitable for the novice who wants to see what a neural network computational model has to offer.

A more complex simulator is the Rochester connectionist simulator [22] developed by Goddard, Lynne and Mintz. The tool is designed to aid in specification, construction and simulation of connectionist networks. It provides graphics display via X-windows and allows a great degree of flexibility in network design simulation. However, no fault injection capabilities are provided.

The emphasis of the neural network simulators designed thus far has been solely on fault-free simulations. Studies on the hardware limitations and failure characteristics of neural networks are also important, given that a number of neural networks of increasing complexity are being fabricated [3]. Thus, a neural network simulation environment with fault analysis capabilities appears essential. In particular, a simulator which allows the user to inject a variety of faults i.ıto the network is needed.

C.3. Overview

Neural network simulation consists of two phases: the learning phase and the operational phase. The learning phase for Kohonen's Feature Map corresponds to the clustering process, whereas for the three-layered perceptron model, it corresponds to the training process. The operational phase in Kohonen's Feature Map refers to the tagging process, and for the three-layered perceptron network, the operational phase refers to the classification of new input vectors.

To use FANS, the user provides a specification file, which indicates the network architecture, the fault models, the learning parameters, the operation parameters and the output format. After the specifications have been made, no user intervention is required until the simulation is completed. The simulator outputs the input pattern, the output pattern, the highest value of the output node and the percentage of mistags or misclassifications. In addition, the user can also sample network values, such as link-weights and node output values, at preset intervals to investigate the real-time network behavior.

Figure 17 shows the structure of the neural network simulator. There are four parts: (1) the user specification file, (2) the network simulator, (3) the fault injector and (4) the output module. The simulator reads the specification file and constructs the network accordingly. The learning phase and the operation phase are then executed in succession. The user can use the fault injector to inject faults into the network during any of the two phases.

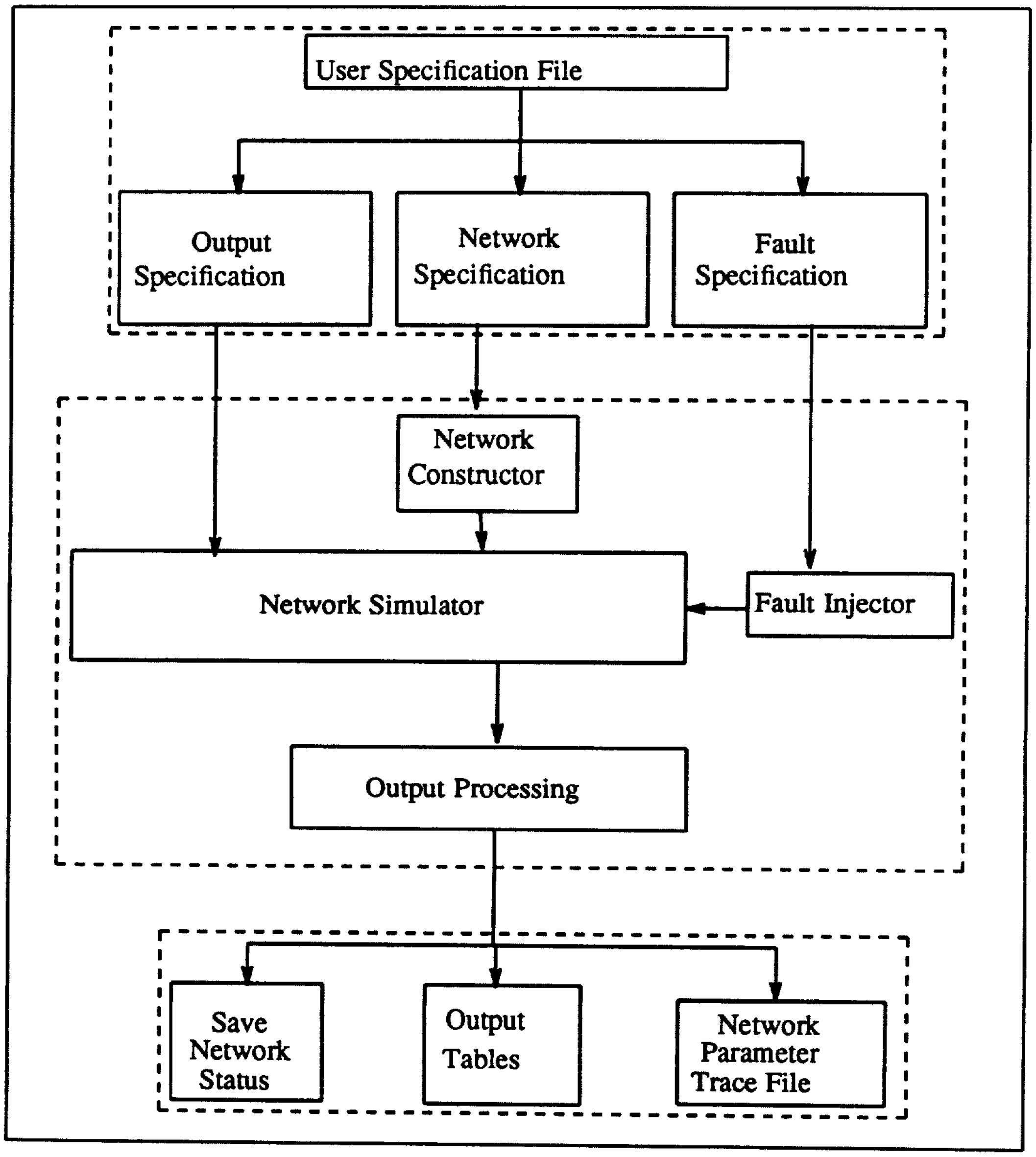

Figure 17. Block diagram of the simulator.

C.3.1. User specification file

The user specification file consists of the network specification, the output specification and the fault specification. The network specification is needed to construct the network and perform the learning process. The user is required to indicate the name of the network, the type of network model, the number of nodes in each layer of the network, the learning duration and the training patterns.

The output specification indicates the type of information to output. There are two types of output: (1) sampled network values and (2) a summary table. To sample network values, the user specifies the sampling interval and the parameters to be sampled. The simulator will then output the specified network values and the sampling time, to a specified file output file. Currently, there is only one kind of summary table. This table outputs the clustering or classification results for each vector in the input set. For the purpose of this thesis, the output table also includes the percentage of mistagged or misclassified vectors.

The fault specification indicates the number of faults to be injected, the location of the faults and their fault types (as specified in Chapter 4 of this thesis). The faults can occur during the learning or the operational phase. Each fault specification has a unique identifier. The learning phase and the operational phase invoke the fault injector by calling the specific fault specification identifier. This allows the user to define several fault models and inject them as required.

C.3.2. Network simulator

As shown in Figure 17, the network simulator consists of four modules: (1) the network constructor, (2) the network simulator, (3) the fault injector and (4) the output module. The network constructor interprets instructions from the network specification given by the user. It creates the specified number of nodes and interconnects them according to the chosen network model. By default, the link-weights are initialized to small random numbers and the node values to zeros. If a *load file* is specified, the network constructor will initialize the link-weights and nodes to the values stipulated in the file. This allows the user to restore a previous network status.

After the network is constructed, the learning phase is initiated. The simulator will perform the forward and backward propagation for the number of cycles specified by the training duration. The simulator sequentially processes each node in the network, from the first node in the input layer to the last node in the output layer. The link-weights of each node are iteratively modified according to the appropriate network paradigm. The change in each link-weight during subsequent iterations is used to determine if the network has stabilized. The simulator is currently set so that the learning process terminates if each link-weight is within 1% of its previous value. A user specified maximum number of simulation cycles is used to limit the learning process if the link-weights do not stabilize.

During the operational phase, the network receives new input patterns and produces the required output patterns. As with the learning phase, the fault injector will be invoked if a fault specification is included. The user can also compare the output patterns produced by a faulty network with that produced by a fault-free network. To do this, an *answer file* which specifies the output patterns from a fault-free network is provided by the user.[11] The simulator compares the output patterns with those in the *answer file* and computes the number of matches.

C.3.3. Fault injector

The fault models provided by FANS are described in Chapter 4. The fault injector can be invoked during the either the learning phase or the operational phase of the simulation. During each cycle, the simulator checks the fault injector flag. If the flag is on, the fault injector is invoked.

The fault injector keeps a list of the faults to be injected. For transient faults, a counter is used to monitor the transient duration. The flag above will remain set and faults will be injected until the counter has expired. For permanent faults, the flag remains set so that during every simulation cycle, the faulty components are perturbed by a user-specified constant. For example, to simulate a node offset fault, the fault injector adds the output value of the faulty node to the constant offset indicated by the user. For a node memory fault, the specified link-weight is perturbed by a random amount. However, for permanent link faults and permanent full-node faults, the physical component is removed from the network. This is simply implemented in the program by removing the pointer to the link or the node structure.

[11]In our experiment, the output patterns are obtained from the fault-free network and stored in the *answer file*. Next, a new simulation is performed, whereby a fault is injected, with the same input vectors. The output produced is compared with the corresponding output patterns in the *answer file*. The percentage of mistags or misclassifications is measured by the number of mismatch between the faulty output patterns and those in the *answer file*.

C.3.4. Output module

There are three kinds of output available: (1) network status file, (2) summary table and (3) sampled outputs. A *network status file* contains all network parameters such as link-weights, node values, node function and network architecture. It is generated at the end of the simulation and may be used at a later time to restore the network status.

The summary table has been customized for the purposes of this thesis. The percentages of mistags and misclassifications are reported for a clustering network and a classification network, respectively. The table also lists the desired and actual output patterns and the corresponding highest output value.

The third type of output consists of sampled network values. Each file lists the sampling time and the corresponding network value. This file can be used as an input to the makefile called *makegr* to plot the values on a PostScript printer. This makefile uses *grap* to convert the input data into a *pic* file, which is later fed to the *ditroff* text processor and a PostScript generator to produce a graph on the laser printer [23].

C.3.5. Running FANS

The executable file for the simulator *sim* can be found in the *examples* directory of FANS. The user can also obtain the executable file by calling *make* in the directory *FANS/src*. The *makefile* compiles all the source codes and produces the executable file *sim*. The simulator can be invoked by typing the command *sim*, followed by the name of the user specification file, in C-Shell. For example, the command *sim example1* will invoke the simulator to simulate the network specified in the file *example1*.

C.4. User's Guide

The neural network simulator developed in this thesis runs and injects faults with minimal intervention from the user. The user only has to provide a specifications program in order to use the customized features of the simulator. It consists of three types of specifications: one for network, output and fault injection.

Although the user is able to use the simulator by writing C programs and calling the functions provided, this is a time-consuming process. The user specification commands allow the user to use the functions provided without writing C programs . Although the simulator is customized to two network paradigms, the modular structure makes it easy for the user to enhance the simulator for other networks. This section describes the commands used in the user specification file. An example is presented to illustrate the usage of each of the commands.

C.4.1. Network declaration

Syntax: network *<identification number>*

<network name (max 50 alphanumeric characters)>

Example: network 12

Three-layered perceptron learning network

The *network* command marks the beginning of the user specification file. The command is immediately followed by a number, which is a unique identification number for the network. The next line consists of the name of the network. The user can provide a short and meaningful phrase describing the purpose of the simulation.

C.4.2. Model specification

Syntax: model = <*network model index*>

Example: model = backprop

The *model* command specifies the network model to be used. Since the simulator is customized to the multilayered perceptron model with the back-propagation learning algorithm and the Kohonen's Feature Map, the descriptors *backprop* and *kohonen* are used to identify them, respectively.

C.4.3. Input nodes

Syntax: input = <*number of input nodes*>

Example: input = 10

The command *input* declares the number of input nodes in the network. In the above example, the user requests ten input nodes. The number of input nodes in a network must be greater than one. All input nodes have, by default, a sigmoidal activation function and an identity node function.

C.4.4. Hidden nodes

Syntax: hidden = <*number of layers*> <*number of nodes in each layer*>

Example: hidden = 2 10 15

The command *hidden* specifies the number of hidden layers and the number of nodes in each of those layers. In the above example, the network has two hidden layers, with ten nodes in the first and fifteen nodes in the second. The network can have an arbitrary number of hidden layers. The node activation function is sigmoidal and the node function is chosen according to the model specification. For example, if the *model* = *kohonen*, the node computes the Euclidean distance between the inputs and the link-weights.

C.4.5. Output nodes

Syntax:	output = <*number of output nodes*>
Example:	output = 10

The command *output* specifies the number of output nodes in the network. In the above example, the network has ten output nodes. The node activation function is sigmoidal and the node function depends on the selected network model.

C.4.6. Load network values

Syntax:	load = <*name of file containing network parameters*>
Example:	load = demo.save

The command *load* instructs the simulator to initialize the network parameters such as the link-weights and the node activation levels from the values given in the specified file. This command overrides the default initializing process, which sets all the node values to zero and all the link-weights to a small random value.

C.4.7. Picking a node

Syntax: node *<num>* = *<node type> <node layer> <node identifier>*

Example: node 2 = 1 0 4

In the simulator, a node is identified by three parameters: a node type, a layer number and a node number. Input nodes, hidden nodes and output nodes are identified by the constants one, two and three, respectively. In the example above, the user picks the fourth input node in the zeroth input layer. In this simulator, only one input layer and one output layer is supported. Thus, the layer number is always zero for node types one and three. The node number is an identifier given by the user. This command is used together with the command *outspec* and *profile* to select a node to monitor or to inject faults, respectively.

C.4.8. Picking a link

Syntax: link *<num>* = *<source node layer> <source node layer> <source node identifier> <dest node type> <dest node layer> <dest node num>*

Example: link 1 = 1 0 4 2 1 4

In the simulator, a link is identified by two pairs of triplet. The first triplet represents the node origin of the link and the second triplet represents the destination node of the link. In the example above, the specified link is between the fourth input node and the fourth node in the second hidden layer. The link number is an identifier given to this link by the user. This command is used together with the command *outspec* and *profile* to select a link to monitor and to inject faults, respectively.

C.4.9. Output specifications

Syntax: outspec *<identifier>*

outfile = *<name of output file>*

grain = *<sampling interval>*

Picked link or node

outspecend

Example: outspec 3

```
outfile = mydemo.out
grain   = 10
link 1  = 1 1 4 2 1 2
link 2  = 2 1 4 3 1 2
outspecend
```

The set of commands above selects a link or a node to be monitored. The simulator samples the link-weights or the node activation levels and records the values in an output file specified by the command *outfile*. If no filename is given, the default is *net.out*. The command *grain* specifies the sampling interval in a number of simulation cycles. In the example above, the user wants to sample two link-weights every tenth simulation cycle and record the values in the file *mydemo.out*. Each *outspec* command allows the user to monitor either links or nodes.

C.4.10. Learning specifications

Syntax:

```
learn
  inputfile = <filename of input vectors>
  teachfile = <filename of desired output patterns>
  duration  = <duration of learning in simulation cycles>
  fault     = <fault specification identifier>
  outspec
       Body of output specifications
  outspecend
learnend
```

Example:

```
learn
  inputfile = demo.input
  teachfile = demo.teachfile
  duration  = 500
  fault     = 4
  outspec
     outfile = mydemo.out
     grain   = 10
       node 1  = 1 0 5
       node 2  = 3 0 1
  outspecend
learnend
```

The set of commands above allows the user to specify the learning process. The command *inputfile* specifies the name of the file containing the training patterns. The command *teachfile* specifies the name of the file containing the desired output patterns. This command is used only when the user selects *backprop* as the network model. The command *duration* sets the number of simulation cycles for the learning process. The command *fault* indicates the index of the fault specifications if a fault is to be injected during the learning duration.

The user can also specify certain network links or nodes to be monitored during the learning process. In the example above, the user trains the network with the input patterns in *demo.input* and the desired output pattern in *demo.teachfile*. The learning duration is 500 simulation cycles and the fault specification number four is used for the fault injection process. In addition, the user also wants to monitor the activation levels of two nodes in the network every tenth cycle.

C.4.11. Fault injector specifications

Syntax: fault *<identifier>*

<name of the fault injection process>

inject = *<number of faults>*

profile *<identifier>*

location = *<faulty component> <number of locations>*

Component specifications

duration = *<duration type> <number of cycles>*

mode = *<nature of fault>*

```
            type = <fault type> <parameters>
        profilend
        outspec
              Body of specification
        outspecend
    faultend
```

Example:

```
    fault 4
        Transient link faults
        inject = 1
        profile 1
            location = rannode
            duration = permanent
            mode = constant
            type = complete
        profilend
        outspec
            outfile = mydemo.out
            grain   = 10
            link 1  = 1 0 3 2 1 2
        outspecend
    faultend
```

The set of commands above is used to describe an injected fault. The fault specification block begins with the command *fault* and an identifier. This identifier is used in the learning phase or the operation phase to inject faults. A character string of 50 characters is provided to the user to describe the process. The command *inject* specifies the number of faults to be injected. For each fault, there is a *profile* description of the fault. In the *profile* description block, the location, duration, perturbation, mode and fault type are specified.

The command *location* specifies the location of the fault. It is followed by a descriptor which represents the location type. There are four descriptors available: (1) *rannode* indicates a randomly selected faulty node, (2) *ranlink* indicates randomly selected faulty link, (3) *link* indicates that the links specified in the next lines will be the targeted faulty components and (4) *node* indicates that the nodes given in the next lines are the faulty nodes. The number of faulty locations is specified if the descriptor *link* or *node* is used.

The command *duration* specifies whether the fault is transient or permanent. The descriptors *transient* and *permanent* are used. If the fault is transient, the transient duration, in number of simulation cycles, follows the descriptor. The command *mode* indicates if the perturbation is constant or Byzantine. The descriptors *constant* and *byzantine* are used to describe the mode. Finally, the fault type is specified by the command *type.* The descriptor *complete* indicates that the component fails completely and produces no response. The descriptors *offset*, *delay* and *gain* are used to specify node offset, node response delay and node gain faults, as described in Chapter 4. The descriptor *random* indicates that the perturbation is random noise. In the example above, the user injects a node fault into a randomly selected node. The fault is permanent and the fault type is a full-node fault. The user also specifies a link-weight to be sampled every 10 simulation cycles.

C.4.12. Operation phase

Syntax:

```
operation
   testfile  = <file containing input patterns>
   ansfile   = <file containing desired output patterns>
   fault     = <fault specification index>
   threshold = <lowest acceptable output value>
   outspec <identifier>
        Body of output specifications
   outspecend
operationend
```

Example:

```
operation
   testfile  = mydemo.test
   threshold = 0.8
   fault     = 3
   outspec 1
        node 1 = 3 0 0
        node 2 = 3 0 1
   outspecend
   outspec 2
     link 1 = 1 0 0 2 0 1
```

```
        outspecend
    operationend
```

The set of commands above specifies the usage of a trained network. The commands *operation* and *operationend* bracket the block of codes for execution specifications. The user has to supply the file containing the input patterns with the command *testfile*. If the user has the desired output patterns corresponding to the input patterns, the command *ansfile* could be used to specify the file name. The simulator will then compare the actual and desired output patterns and compute the percentage of matches. As in the learning process, the user can inject faults during the execution phase. The command *fault* associates the operation phase with the specified fault injection specifications. For the purpose of this thesis, the *threshold* command is used to specify the lowest acceptable output value. This is used in the thesis to study the impact of low output values of the multilayered perceptron network. The user is also able to monitor certain network values by using the *outspec* command. In the example above, the threshold is 0.8. Network values can also be monitored using the *outspec* command to sample network parameters during the operation phase.

C.4.13. Save network status

Syntax:	save = <*filename*>
Example:	save = mydemo.save

The command *save* allows to the user to save the current network values in the specified file. If no filename is provided, the simulator will provide a default name of *net.save.*

C.4.14. Summary tables

Syntax:	summary = <type identification number>
Example:	summary = 1

The simulator is customized to study the impact of faults on neural networks. One summary report is currently available. It is identified by the constant *1* in the simulator. It presents the actual and the desired output patterns, the highest output value and the total number of matches between the actual and desired output patterns. Section C.5 shows two examples of the table. The user can easily enhance the simulator with other output tables and summary reports.

C.4.15. Running FANS

Syntax:	sim <specification filename>
Example:	sim example1

The simulator can be executed by calling *sim* in C-shell, followed by the name of the specification file. The executable file *sim* can be found in the directory *examples* of FANS. An executable can also be generated by using the *Makefile* in the directory *FANS/src*.

C.5. Sample Specification Files

This section describes two sample specification files: the first simulates a three-layered perceptron network, and the second simulates Kohonen's Feature Map. These are two of the specification files used in the experiments described in this thesis. They can be executed by the commands *sim example1* and *sim example2* respectively. More examples, along with the training vectors can be found in the *examples* directory of the simulator.

C.5.1. Three-layered perceptron network

Figure 18 shows the specification file to simulate a three-layered perceptron network. The network has ten input nodes, twenty output nodes and two hidden layers, with five nodes in the first and fifteen nodes in the second hidden layer. The input patterns are given in the file *clust.dat* and the desired output patterns in the file *clust.teach.20*. The maximum learning duration is set to 600 simulation cycles.

During the operation phase, the input vectors are taken from the file *clust.dat*. Fault is injected at the start of the operation phase and *fault 1* is the fault specification file. In the example, the fault injector will inject a single link fault at a randomly chosen link. The fault is a permanently broken link.

In order to measure the classification accuracy of the network, the command *ansfile* is used to specify that the file *clust.teach.20* contains the desired output patterns. Since the user requested *summary* output format, the actual output patterns and the desired output patterns from *ansfile* are listed.

```
% more example1
network 1
        Back propagation network
        model   = backprop
        input   = 10
        hidden  = 2 5 15
        output  = 20
        learn
                inputfile = clust.dat
                teachfile = clust.teach.20
                duration  = 600
        learnend
        operation
                fault = 1
                testfile  = clust.dat
                ansfile   = clust.teach.20
                threshold = 0.8
        operationend
        summary = 1
        save    = clust.save.20.1
networkend

fault 1
        Fault testing for classifier
        inject  =  1
        profile 1
                location  = ranlink 1
                duration  = permanent
                mode      = constant
                type      = complete
        profilend
faultend

end.
%
```

Figure 18. Specification file for the three-layered network.

Figure 19 shows a portion of the script file with the output table. The simulation is initiated by the command *sim example1*. The output lists the observation number, the output node with the highest value, the corresponding fault-free output node and the node output value. The percentage of mistag or misclassification is computed and printed at the end of the table. Misclassifications can be identified by comparing the *desired* and *actual* columns.

For example, observations 5, 6 and 7 are misclassifications because the desired output node is number 11 but the actual active output node is 1. The threshold of the output value is set at 0.8 and is used as part of a decision factor for misclassification, as discussed in Chapter 5. After the operation phase is completed, the network parameters are saved in the file *clust.save.20.1* for future use.

C.5.2 Kohonen's Feature Map

Figure 20 shows the user specification file to simulate Kohonen's Feature Map. The network has ten input nodes and seven output nodes. Since Kohonen's Feature Map is a single-layered network, hidden layers are not specified. The learning process is unsupervised and thus, the user specifies only the input patterns. The learning duration is limited to 500 simulation cycles. The user also specifies that the network status be restored according to the values in the file *example2.save.*

During the operation phase, the input patterns are taken from the file *dataset1* in the directory *testcase*. A fault is injected at the start of the operation phase. The faulty component is a randomly selected node and the fault is transient, with a duration of 40 cycles. The perturbation is Byzantine and the fault type is a full-node fault. The output patterns and the activation levels will be displayed in an output table, since the command *summary = 1* is specified.

Figure 21 shows a portion of the script file and the output table. The simulation is initiated by the command *sim example2*. The output table is similar to that in the previous example.

```
% sim example1 &
%

Output for back propagation network
===================================

Classification results :
No.    Desired          Actual          Output Activation
=====================================================
1         5               5                 0.921302
2         5               5                 0.859132
3         5               5                 0.859342
4         14              1                 0.891788
5         11              1                 0.720834
6         11              1                 0.775482
7         11              1                 0.775501
8         11              11                0.820430
.          .               .                .
.          .               .                .
.          .               .                .
192       4               4                 0.992499
193       16              16                0.996142
194       6               6                 0.990695
195       7               7                 0.942462
196       2               5                 0.921201
197       12              5                 0.862375
198       8               8                 0.927681
199       18              5                 0.806878
200       4               4                 0.992499

-----------------------------------------------------

  Threshold                 = 0.800000
  Number of test patterns = 200
  Misclassification %       = 29.500000
%
```

Figure 19. Output for the three-layered network.

```
% more example2
network 1
        Kohonen self organising network
        model    = kohonen
        input    = 10
        output   = 7
        learn
                inputfile = testcase/dataset1
                duration  = 500
        learnend
        operation
                fault = 1
                testfile = testcase/dataset1
        operationend
        summary = 1
        load = example2.save
networkend

fault 1
        Fault testing for clusterer
        inject  =  1
        profile 1
                location   = rannode 1
                duration   = transient 40
                mode       = byzantine
                type       = complete
        profilend
faultend

end.
%
```

Figure 20. Specification for Kohonen's Feature Map.

```
% sim example2 &
%

Output for kohonen self organising network
==========================================

Clustering results :
No.    Desired        Actual        Output Activation
=====================================================
1        2              2              13.989765
2        3              3              9.623331
3        3              3              7.521379
4        3              3              16.624250
5        5              4              10.557803
6        5              6              13.041821
7        3              3              9.010196
8        3              3              16.731127
.        .              .              .
.        .              .              .
.        .              .              .
192      3              3              21.263313
193      3              3              44.500603
194      3              3              9.466000
195      3              3              16.487846
196      3              3              20.921387
197      3              3              13.697389
198      3              3              36.723930
199      3              3              13.194621
200      3              3              27.134039

-----------------------------------------------------

  Number of test patterns = 200
  Mistag %                = 22.500000

%
```

Figure 21. Output for Kohonen's Feature Map.

APPENDIX D.

IMPACT OF FAULTS ON THE RELEARNING DURATION

The following figures show the number of relearning cycles for the classification network when the specific faults were injected.

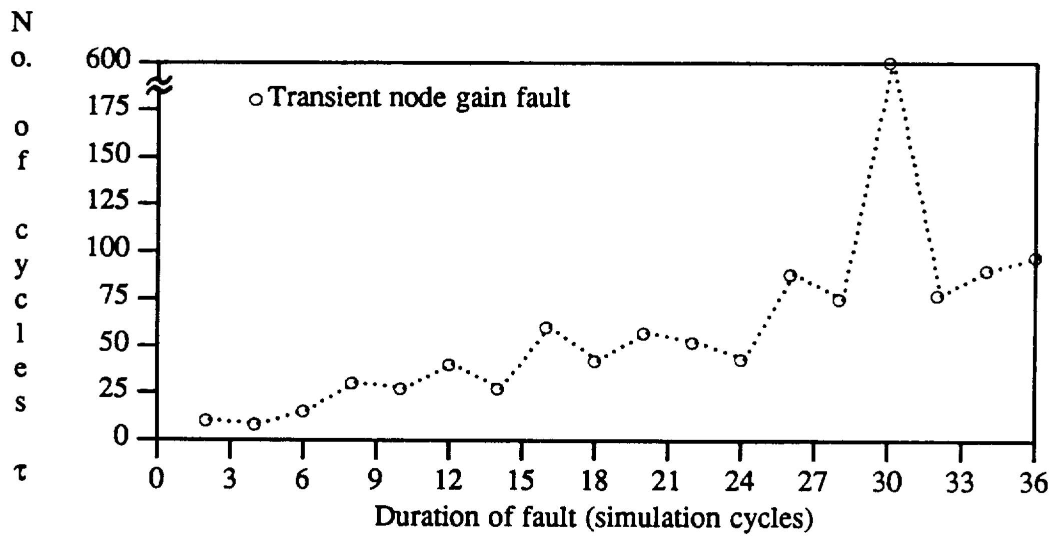

Figure 22. Impact of transient node gain fault on relearning duration.

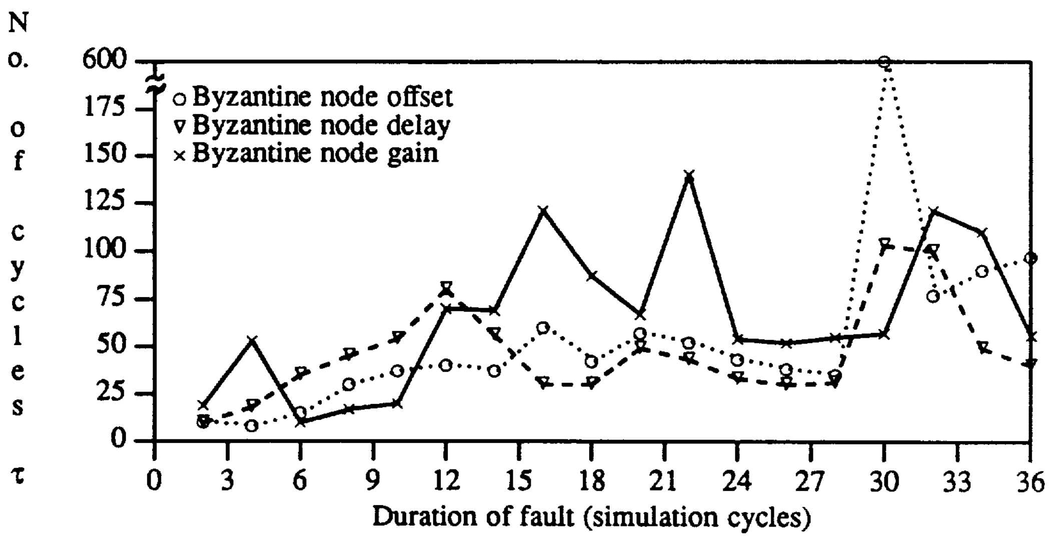

Figure 23. Impact of Byzantine faults on relearning duration.

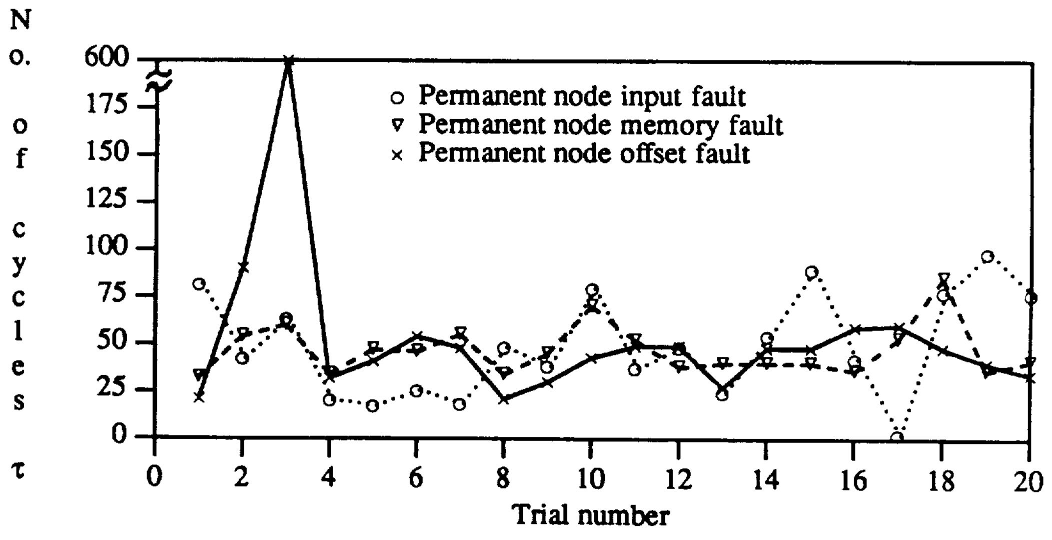

Figure 24. Impact of permanent node faults on the relearning duration.

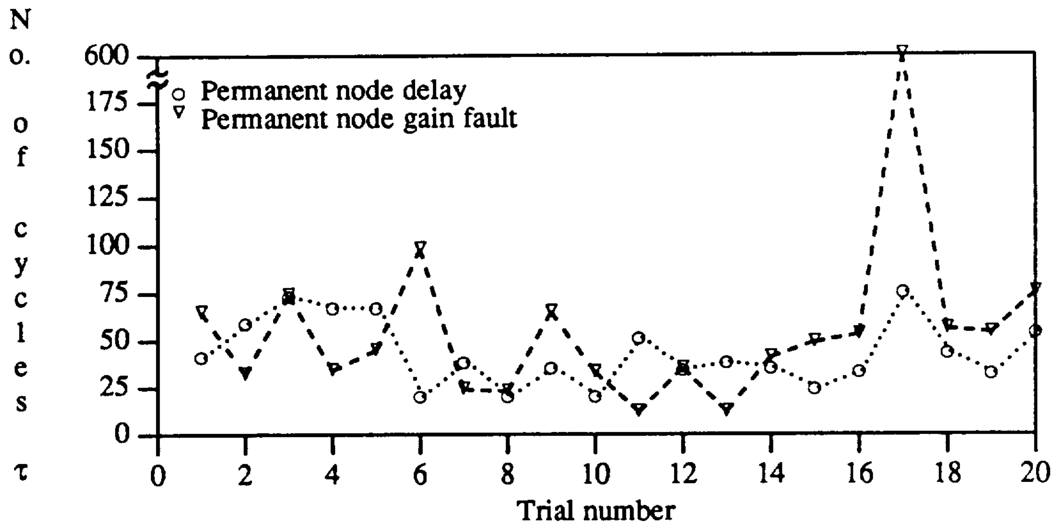

Figure 25. Impact of permanent node delay and node gain faults on the relearning duration.

REFERENCES

[1] J. A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research.* Cambridge, MA: MIT Press, 1988.

[2] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4 - 22, April 1987.

[3] M. W. Roth, "Neural network technology and its applications," *Journal of Knowledge Engineering*, vol. 2, pp. 46 - 62, March 1989.

[4] M. F. Tenerio and C. S. Hughes, "Real time noisy image segmentation using artificial neural networks," *Proceedings IEEE International Conference on Neural Networks*, vol. 4, pp. 357 - 364, 1987.

[5] T. Kohonen, *Self Organising and Associative Memory.* Berlin: Springer-Verlag, 1988.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Cambridge: MIT Press, pp. 318 - 362, 1986.

[7] T. Kohonen, "The Neural Phonetic Typewriter," *IEEE Computer*, vol. 21, pp. 11 - 22, March 1988.

[8] N. M. Nasrabadi and Y. Feng, "Vector Quantization of Images Based Upon the Kohonen Self-Organizing Feature Maps," *Proceedings IEEE International Conference on Neural Networks*, vol. 1, pp. 101 - 108, 1988.

[9] J. Mann, R. P. Lippmann, B. Berger, and J. Raffel, "A Self-Organizing Neural Net Chip," *Proceedings IEEE Custom Integrated Circuits Conference*, pp. 10.3.1 - 10.3.5, 1988.

[10] K. Saito and R. Nakano, "Medical diagnostic expert system based on PDP model," *Proceedings IEEE International Conference on Neural Networks*, vol. 1, pp. 255 - 262, 1988.

[11] D. G. Bounds, P. J. Lloyds, B. Mathew, and G. Waddell, "A Multilayer Perceptron Network for Diagnosis of Low Back Pain," *Proceedings IEEE International Conference on Neural Networks*, vol. 2, pp. 481 - 490, 1988.

[12] A. Rajavelu, M. T. Musavi, and M. V. Shirvaikar, "A Neural Network Approach to Character Recognition," *Neural Networks*, vol. 2, pp. 387 - 393, October 1989.

[13] H. Yang and C. C. Guest, "Performance of Backpropagation for Rotation Invariant Pattern Recognition," *Proceedings International Conference on Neural Networks*, vol. 4, pp. 365 - 370, June 1987.

[14] R. K. Elsley, ''A Learning Architecture for Control Based on Back-propagation Neural Networks,'' *Proceedings IEEE International Conference on Neural Networks*, vol. 2, pp. 587 - 594, 1988.

[15] H. Weschsler and G. L. Zimmerman, ''Fault tolerant recognition using DAM's,'' *Proceedings IEEE International Conference on Neural Networks*, vol. 2, pp. 719 - 726, 1987.

[16] J. A. G. Nijhuis and L. Saanenburg, ''Fault tolerance of neural associative memories,'' *IEE Proceedings*, vol. 136, pp. 389 - 394, September 1989.

[17] Jos Nijhuis, Bernd Hofflinger, Andre van Schaik, and Lambert Spaanenburg, ''Limits to the Fault-Tolerance of a Feedforward Neural Network with Learning,'' *Proceedings International Symposium on Fault-Tolerance Computing*, pp. 228 - 235, 1990.

[18] L. F. Pau, *Failure Diagnosis and Performance Monitoring.* New York: Marcel Dekker, 1981.

[19] M. Pease, R. Shostak, and L. Lamport, ''Reaching agreement in the presence of faults,'' *J. ACM*, vol. 27:2, pp. 228 - 234, April 1980.

[20] H. Spath, *Cluster Analysis Algorithms.* West Sussex: Ellis Horwood, 1979.

[21] P. Gallinari, S. Thiria, and F. F. Soulie, ''Multilayer perceptrons and data analysis,'' *Proceedings IEEE International Conference on Neural Networks*, vol. 1, pp. 391 - 399, 1989.

[22] N. H. Goddard, K. J. Lynne, and T. Mintz, ''Rochester connectionist simulator,'' Technical Report 233, University of Rochester, Rochester, New York, 1988.

[23] J. F. Ossanna, *NROFF/TROFF User's Manual.* Murray Hill, New Jersey: Bell Laboratories, 1976.